ACE Working Group
Internet-Draft
Intended status: Standards Track
Expires: December 31, 2015

L. Seitz
SICS
G. Selander
Ericsson
M. Vucinic
STMicroelectronics

June 29, 2015

Authorization for Constrained RESTful Environments
draft-seitz-ace-core-authz-00

Abstract

This memo defines a framework for authorization in constrained-node networks, i.e. networks where some devices have severe constraints on memory, processing, power and communication bandwidth. The main goal is to offload constrained devices by providing them access control support from trusted parties that are less constrained. The approach is based on RESTful requests to dedicated access management resources, supporting different authorization schemes and communication security paradigms.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

Authorization is the process of deciding what an entity ought to be allowed to do. Authorization management and processing access control policies in order to reach an authorization decision is often a heavyweight task, involving e.g. database lookups, credential verification and matching large sets of values against each other. Constrained devices are ill-equipped to handle this on their own, therefore the logical approach is to offload authorization management and part of the decision process to a less constrained, trusted third party.

This memo describes an approach to authorization in constrained-node networks using dedicated resources for access management. The various entities in the architecture handle authorization information by making RESTful requests (GET/PUT/POST/DELETE) to these resources and thereby establish the necessary security contexts for interactions between endpoints.

The approach is based on the use cases from [I-D.ietf-ace-usecases] and on the architecture and problem description from [I-D.gerdes-ace-actors]. It introduces protocols for requesting authorization information from a trusted third party, encodings for such authorization information and protocols for transferring it to the affected device(s). We assume that CoAP [RFC7252] is the preferred application-layer protocol and that constrained devices implementing these mechanisms can be as resource constrained as class 1 according to the definitions in [RFC7228].

An important goal in the design of this framework is to preserve end-to-end security in the presence of untrusted intermediate nodes in the communication of requests, responses and related authorization information.

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].

Certain security-related terms are to be understood in the sense defined in [RFC4949]. These terms include, but are not limited to, "authentication", "authorization", "confidentiality", "(data) integrity", "message authentication code", and "verify".

RESTful terms including "resource", "representation", etc. are to be understood as used in HTTP [RFC7231] and CoAP [RFC7252].

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

Terminology for entities in the architecture is defined in [I-D.gerdes-ace-actors], such as the constrained nodes Client (C) and Resource Server (RS), and the less constrained nodes Client Authorization Server (CAS) and Authorization Server (AS).

Since this draft focuses on the problem of access control to resources it is for simplicity assumed that the Client Authorization Server functionality is not stand-alone but subsumed by either the Authorization Server or the Client (see section 2.2 in [I-D.gerdes-ace-actors]).

We use the term "authorization token" for authorization information that is protected with object security.

2. Overview

This section contains an overview of the authentication and authorization problem and the solution described in this memo. We begin with two generic examples.

2.1. Example A: Access to an Actuator

Consider an actuator, e.g. in an industrial control system. Set in the terms of the architecture [I-D.gerdes-ace-actors], the actuator is hosted in one endpoint, acting RS, and another endpoint, acting C, is requesting access to the actuation resource.

Security requirements of special importance are:

o RS needs to verify that the requesting Client is authorized to access the resource.

o Integrity and replay protection of request. Assuming CoAP, integrity protection applies to Payload but also to other fields such as for example the CoAP option Uri-path.

o C must be able to verify the integrity of the response, and that the received message is the response to the previously made request, in order for C to verify that the actuation was performed.

o It must be possible to confidentiality protect the request and response, e.g. for privacy reasons.

2.2. Example B: Multiple Devices Accessing Sensor Data

Consider a constrained sensor performing measurements consumed by multiple devices. Since the communication of a duty-cycled sensor with each consuming device would require energy expensive radio receptions and transmissions, it is favorable to minimize the number of devices that directly interact with the sensor. Therefore the sensor publishes (e.g. sends notifications with) measurements to a message broker, which forwards the measurements to subscribing devices. Access control is performed by only giving plaintext access to authorized subscribers.

In the terminology of [I-D.gerdes-ace-actors] this means that the sensor acts RS and the subscribing devices are Clients. In this case, the RS may neither be able to verify integrity nor replay of individual client requests, nor authorize individual clients, since it isn't necessarily interacting directly with any clients. For the same reason the client is not able to match a publication against a request. The main relevant security requirements here are:

o Access to plaintext publications must be restricted to authorized Clients.

o A Client needs to verify the origin of the publication, and that it is not a replay of an old publication.

2.3. The Authentication and Authorization Problem

Both examples in the previous subsections are encompassed by the problem statement described in [I-D.gerdes-ace-actors] but their security requirements are quite different. On a high level, the authentication and authorization problem can be broken down into three sub-problems:

1. The policy for resource access as defined by the Resource Owner (RO) and managed by the Authorization Server (AS) needs to be translated into authorization information following an Authorization Information Format (AIF) that the RS can parse and act upon.

2. Similarly, the Client may need to acquire keys/credentials or other information from the AS to securely access the resource. Such client information is encoded following a Client Information Format (CIF).

3. In order for Client and RS to authenticate and authorize according to policy, the communication of the various information elements needs to be protected appropriately end-to-end: from AS to C, from AS to RS, and between C and RS.

To access an actuator, as in example A, the authorization information could be an authorization decision that C is granted access to a certain resource, or attributes of C that RS can match against internal access policies. The client information could be cryptographic keys which C could use to establish secure communication with RS.

In the publish-subscribe scenario of example B, the authorization information (if present) could be cryptographic keys that must be used for protection of published resource representations. The client information could be cryptographic keys/credentials with which publications of certain resource representations can be verified and decrypted.

Communication security could be addressed with a session security based protocol such as DTLS [RFC6347], or an object security solution based on e.g. COSE [I-D.schaad-cose-msg] (see Section 4).

The other sub-problems listed in this section, AIF and CIF, are further detailed in Section 6 and Section 7, respectively.

2.4. Solution Overview

Considering the large variety of use cases, it may be difficult to address the authentication and authorization problem with one single protocol instance. In this section we list some guiding principles for a solution. The remainder of the document provides solution components based on the principles listed in this section. In Section 8 the detailed processing steps are presented. In Appendix A examples of the proposed solution are given.

PRINCIPLE 1: Allow different order of information flows

The information flows, including AIF, CIF, and Resource Request/ Response (RRR) (Figure 1), may take place in different order see Section 3.

```
                    +-------------------------------+
                    |          Authorization        |
                    |             Server            |
                    +-------------------------------+
                         |                       |
                        CIF                     AIF
                         |                       |
                         V                       V
                    +--------+             +-----------+
                    | Client |<--- RRR --->| Resource  |
                    |        |             |  Server   |
                    +--------+             +-----------+
```

Figure 1: Information flows

PRINCIPLE 2: Build upon REST

The information flows should leverage the RESTful architecture. For RRR this is already assumed, but RESTful administrative resources for managing access to e.g. AIF and CIF need to be defined. The support for REST should not preclude optimizations for particular protocols, for example some flows may be embedded within the DTLS handshake.

PRINCIPLE 3: Allow security at different layers

Leave to the application to decide about session based or object based security for any given information flow.

3. Information Flows

This section gives an overview of information flows between the entities in the architecture, and the use of access management resources.

3.1. Message Sequences

The information flows in scope between the nodes (constrained to constrained and constrained to less constrained) are described in [I-D.gerdes-ace-actors]. Taking [RFC2904] as a starting point, there are three authorization schemes, defining the message sequences for the flow of request, response, and authorization data:

1. In the Push scheme, C first obtains authorization information from AS and then transmits this data to RS together with its request, and finally RS sends its reply. In this model RS does not communicate with AS directly.

2. In the Pull scheme, C sends its request to RS directly, RS then contacts AS in order to obtain authorization information, AS provides authorization information, and finally RS sends back the reply. In this model C does not communicate with AS directly.

3. In the Agent scheme, C sends its request to AS, which evaluates C's access rights and, if authorized, executes the request on C's behalf, transmitting RS's response back to C. In this model C does not communicate with RS directly.

Depending on use case, different schemes may be more appropriate than others. From a constrained-node network point of view we note the following:

- The Push and Agent scheme requires C to connect to AS.
- The Pull and Agent scheme requires RS to connect to AS.
- The Pull scheme requires RS to handle two secure connections at the same time (RS - AS and RS - C) in order to perform the real time authorization check.

Considering processing and memory requirements of RS, the Push scheme is more favorable than the Pull scheme.

The Agent scheme assumes constant connectivity with both C and RS and is therefore not applicable to the constrained setting.

However, the publish-subscribe case from Section 2.2 is not covered by any of the schemes so this memo defines a new scheme, called Client-Pull.

4. In the Client-Pull scheme, RS first provides unconditional access to a protected resource representation. C obtains this protected resource representation and then contacts AS to obtain authorization information (in this case cryptographic keys) for access and verification of the resource representation. In this model RS does not communicate with AS directly, and C does not communicate with RS directly.

The Client-Pull scheme is only useful for GET access. However, it offloads the RS considerably, since the RS neither receives any requests from clients nor performs any access control verifications. Furthermore it enables use cases, such as caching and publish-subscribe, that can not be covered by the other schemes in a satisfactory manner.

3.2. Access Management Resources

This section describes an approach for transferring authorization information between the various entities in the architecture by making RESTful requests to dedicated resources for access management. The structure or naming of such "access management resources" is left for a future version of this memo, and for simplicity we denote any such resource "/authorize".

In this section we illustrate various message sequences for transferring authorization information by means of requesting access management resources. We assume, as in section 3.2 of [I-D.gerdes-ace-dcaf-authorize], that the need for transferring authorization information may be triggered by a request from C to RS to access a resource. Figure 2 shows C making an access request (GET /PUT/POST/DELETE) which cannot be granted by the RS because of missing authorization information.

```
                      C                           RS
                      |                            |
                      |    Request  /resource      |
                      | -------------------------> |
                      | <------------------------- |
                      |    Unauthorized            |
                      |                            |
```

Figure 2: Access request unauthorized.

Starting with the Pull scheme, if the RS can access the AS, then any missing authorization information could be requested on the fly, see Figure 3. The RS sends an authorization request in a specific Authorization Request Format (ARF) to the /authorize resource at the AS as detailed in Section 8 and receives authorization information in a specific Authorization Information Format (AIF) in the response. RS updates the stored authorization information based on the new information and executes the client's resource request accordingly. Note that C and RS need be mutually authenticated using pre-established credentials in this scheme.

```
        C                           RS                            AS
        |                           |                             |
        |    Request  /resource     |                             |
        | ------------------------> |                             |
        |                           |    POST /authorize ARF      |
        |                           | ------------------------> |
        |                           | <------------------------ |
        |                           |             AIF             |
        |                           |                             |
        | <------------------------ |                             |
        |    Granted                |                             |
        |                           |                             |
```

Figure 3: Pull Scheme

In use cases where the RS does not have connectivity with AS, or if the RS cannot handle two secure connections at the same time, the Push scheme provides an alternative, see Figure 4. C receives, in the negative response to its resource request, information about where to request the missing authorization information. In this case it is C that requests the resource /authorize at the AS and receives both authorization information, and client information in a specific Client Information Format (CIF). The latter will be used by C to establish communication security with RS. Next, C pushes the authorization information to the /authorize resource at the RS. RS updates the stored authorization information based on the new data and confirms the change, after which C can repeat the resource request to the RS with the new client information and authorization information in place.

```
        AS                          C                             RS
        |                           |                             |
        |                           |    Request  /resource       |
        |                           | ------------------------> |
        |                           | <------------------------ |
        |                           |    Unauthorized             |
        |    POST  /authorize ARF   |                             |
        | <------------------------ |                             |
        | ------------------------> |                             |
        |       AIF, CIF            |                             |
        |                           |                             |
        |                           |    POST /authorize  AIF     |
        |                           | ------------------------> |
        |                           | <------------------------ |
        |                           |    Changed                  |
        |                           |                             |
        |                           |    Request  /resource       |
        |                           | ------------------------> |
```

```
        |                           | <------------------------ |
        |                           |    Granted                |
        |                           |                           |
```

Figure 4: Push Scheme

Note that in case there is connectivity between RS and AS, but RS has not implemented the Pull scheme or cannot handle two secure connections at the same time, then AS could POST the authorization information to the /authorize resource of RS directly instead of using C as intermediary node.

Also note that the authorization information needs to be integrity protected during the transport between AS and RS. This is because the client has access to it and would otherwise be able to change its permissions. How this is done is described in Section 4.

Finally the Client-Pull scheme, which only requires C to request a Protected Resource Representation from the RS, and client information (e.g. cryptographic keys) from the AS.

```
        AS                          C                           RS
        |                           |                           |
        |                           |    GET  /resource         |
        |                           | ------------------------> |
        |                           | <------------------------ |
        |                           |   Protected Resource      |
        |                           |     Representation        |
        |                           |                           |
        |    POST /authorize ARF    |                           |
        | <------------------------ |                           |
        | ------------------------> |                           |
        |    CIF                    |                           |
        |                           |                           |
```

Figure 5: Client-Pull Scheme

Note that the order of the information flows is different for the different authorization schemes. The authorization scheme is defined at design time, so an RS would typically support one specific message sequence suitable for the particular application.

4. Communication Security

In this section we address the third sub-problem of Section 2.3, the authentication and communication security problem, based on the assumptions made in [I-D.gerdes-ace-actors]:

- o AS and RS are assumed to have established security contexts (keys, credentials, security protocols, parameters). How this was established is out of scope for this work.

- o AS is assumed to have a policy for how C is allowed to access the resources of RS, including information for how to authenticate C, such as a shared secret key, public key or other credential.

- o Both session based and object based security solutions shall be supported.

Since CoAP is the default communication protocol, DTLS [RFC6347] is the default session based security protocol. DTLS MUST NOT be used with untrusted intermediary nodes. For object security in constrained environments, COSE [I-D.schaad-cose-msg] is the main candidate. OSCOAP [I-D.selander-ace-object-security] defines a profile of COSE, suitable for securing individual CoAP messages, and two Modes of operation:

- o Mode:PAYL for (CoAP) Payload confidentiality, integrity and replay protection.

- o Mode:COAP for additional protection of CoAP Headers (Code) and Options (Uri-Path, Uri-Query etc.) and for securing request-response pair.

4.1. AS - RS

The main purpose of this communication is to provide authorization information from AS to RS. There are three communication patterns:

1. RS: POST /authorize to AS (ARF in request, AIF in response)

2. AS: POST /authorize to RS (AIF in request)

3. C: POST /authorize to AS (ARF in request, AIF in response) followed by C: POST /authorize to RS (AIF in request)

Pattern 1. and 2. SHALL be protected by DTLS or COSE Mode:COAP, leveraging the pre-established security context. In pattern 3., since C is an untrusted intermediary node, the authorization information SHALL be protected end-to-end from AS to RS with COSE Mode:PAYL, leveraging the pre-established credentials.

Pattern 2. with COSE Mode:COAP includes the case of POST via an intermediary untrusted Forward Proxy, which in particular could be the Client, if the Client supports Forward Proxy functionality.

In pattern 3. the use of transport layer security (DTLS) or CoAP message security (Mode:COAP) is optional, since Mode:PAYL protected authorization information is in itself a confidential, integrity and replay protected token.

If the subsequent communication between C and RS is going to be protected by DTLS, then an OPTIONAL optimization of the message exchange is to replace "C: POST /authorize to RS (AIF in request)" with a transfer of the protected authorization in the DTLS Handshake, see Section 4.4.

4.2. AS - C

The main purpose of this communication is to provide client information from the AS to the C. The communication pattern is:

- C: POST /authorize to AS to (ARF in request, CIF in response)

This pattern SHALL be protected by DTLS or COSE Mode:COAP, leveraging out of band established credentials.

4.3. C - RS

This is the original resource request/response problem. Since there may be no pre-established keys between C and RS, it is the purpose of AIF and CIF to provide information for authentication and communication security. There are two communication patterns:

1. The Client makes a request (GET/PUT/POST/DELETE /resource) to RS and receives a response.

2. The Client makes a GET request to the RS, or a cache, or message broker and unconditionally receives a protected resource representation

Pattern 1. SHALL be protected by DTLS or COSE Mode:COAP, leveraging the established keys or credentials in authorization information and client information. Pattern 2. SHALL be protected with COSE Mode:PAYL.

4.4. Authorization Information transfer in the DTLS Handshake

In certain situations, a session security protocol like DTLS [RFC6347] can be used directly to send the authorization information to the RS and optimize the message exchange. The authorization information MAY be embedded in the DTLS handshake. In this case the authorization information SHOULD be transferred using the TLS supplemental data extension [RFC4680].

Figure 6 illustrates this approach for a DTLS handshake. The messages marked with * are optional depending on the type of handshake.

```
       Client                                       Resource Server
       ------                                       ---------------

       ClientHello (with extensions) ------->

                                      <-------  HelloVerifyRequest

       ClientHello (with extensions) -------->

                                                      ServerHello
                                                     Certificate*
                                               ServerKeyExchange*
                                              CertificateRequest*
                                    <--------     ServerHelloDone

       SupplementalData (AIF)
       Certificate*
       ClientKeyExchange
       CertificateVerify*
       [ChangeCipherSpec]
       Finished                     -------->
                                               [ChangeCipherSpec]
                                    <--------            Finished
       Application Data             <------->    Application Data
```

Figure 6: Authorization information Transfer in Handshake

The SupplementalDataType value SHOULD be set to 16386 (authz_data). The Resource Server MUST verify the validity of the received authorization information before accepting any application data from the client.

The RS MUST verify that the authorization information transmitted in the SupplementalData message is bound to the same key as the one the Client used in the handshake (see Section 6 for the subject binding in the AIF). Note that this also enables AS-asserted authorization of the DTLS handshake.

5. Authorization Request Format

Authorization requests are represented in the Authorization Request Format (ARF). They are produced either by the Client (Push, Client-Pull) or the RS (Pull) and consumed by the AS. The ARF allows to specify requests for several actions on several resources in a single message.

5.1. ARF Information Model

The general information that needs to be contained in the ARF is the following:

- o The identifier of the subject of this authorization request (usually the Client).

- o The identifier of the host that provides the resources that are requested in this authorization request (CoAP option URI-host). Note that the ARF does not allow to specify more than one host.

- o The requested resources and actions on these resources.

The resources can be represented by the "path-absolute" part of the URI (In CoAP this would be the Uri-Path options).

5.2. ARF Data Model

For representing the ARF discussed in Section 5.1 the following steps are performed:

- o The subject identifier is encoded in a map that maps the type of the identifier to its value. The type could e.g. be a raw public key, or the key identifier of a secret key, or a X.509 distinguished name.

- o The host identifier is encoded as second parameter.

- o The third parameter is array of resource-actions tuples. The structure of the entries is specified as follows:

  * Requests that affect the same resource are merged into a single entry that specifies the union of the permitted actions.

  * The actions GET, POST, PUT, DELETE are represented as integer 0, 1, 2, and 3 respectively.

* The set of numbers is converted into a single number by taking each number to the power of two and computing the inclusive OR of the binary representations of all the numbers.

* Each entry is an array, containing the resource identifier, and the number representing the actions.

This information can then be represented in CBOR [RFC7049] or JSON [RFC7159] as an array of the elements listed above. An example JSON representation of the ARF is given in Figure 7.

```
          [{"subjectKeyId":"someKeyId1"}, "rs.example.com",
           [["sensors/tempC",1], ["conf/sleepInterval", 5],
            ["conf/sleepDuration", 5]]]
```

Figure 7: Example JSON representation of an access request

6. Authorization Information Format

Addressing the first sub-problem of Section 2.3, this section defines an Authorization Information Format (AIF) for encoding authorization information. AIF instances are either consumed directly by an RS (Pull), or packaged into a COSE Mode:PAYL token and sent to the Client, who forwards the token to the RS (Push).

This memo defines two different types of AIF that can be used by an AS:

o One or more access control decisions, listing one or more tuples of actions and resources similar to a capability list. This is a simple super-set of the ARF.

o Group memberships for the Client. This type of AIF is used in a configuration where the RS stores access control lists (ACLs) corresponding to a number of groups. The RS will resolve a specific group to a specific set of ACLs which are then applied to the request.

6.1. AIF Information Model

The general information that needs to be contained in the AIF is the following:

o The identifier of the subject of this authorization information (usually the Client)

- o The identifier of the host that provides the resources for which this authorization information applies (CoAP option URI-host). Note that the AIF does not allow to specify more than one host.

If the authorization information represents one or more group memberships, the only other information needed is the group names. If the authorization information represents access control decisions, the affected resources and the actions are needed.

Authorization decisions can also MAY contain local conditions, that can only be evaluated by the RS at access time (e.g. "NotBefore='09:00', NotAfter='18:00'"). This memo does not define the format of these local conditions, as it is expected that these will be application specific. The RS MUST reject any authorization information that contains local conditions that it does not understand.

6.2. AIF Data Model

For representing the AIF discussed in Section 6.1 the following steps are performed:

- o The subject identifier is encoded in a map that maps the type of the identifier to its value. The type could e.g. be a raw public key, or the key identifier of a secret key, or a X.509 distinguished name. Section 4.3 defines how this is used to bind the authorization information to a specific client.

- o The host identifier is encoded as second parameter.

- o The third parameter is either an array of access control decisions, or if group memberships are asserted, a map, mapping the key 'grp' to the array of group names. The structure of the entries in the access control decision array is the same as for the ARF, except that is may also contain local conditions represented as strings.

- o Each entry is an array, containing the resource identifier, the number representing the actions, and optionally the local conditions.

This information can then be represented in CBOR [RFC7049] or JSON [RFC7159] as an array of the elements listed above. An example JSON representation of the AIF with access control decisions is given in Figure 8, while an example of the AIF containing group membership assertions is shown in Figure 9.

```
   [{"SubjectKeyId":"someKeyId2"}, "rs.example.com",
    [["actuators/doorLock", 5, "not-before:'07:00';not-after:'18:00'"]]]
```

Figure 8: Example JSON representation of access control decisions

```
     [{"SubjectPublicKey":"MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEJG4m93
     ED8tPK2CpkhrrtKNx1vXDbml4Fun1628Qkl1U_aIB5zUfqPwaacznbqoMJ6vQVZ7
     4X9HpfynouLK_ujw"}, "rs.example.com",
     {"grp":["admin","user"]}]
```

Figure 9: Example JSON representation of group memberships

Note that different subject identifiers might be used in instances of the ARF and the AIF in the same Push scheme. This is due to the fact that the credential the Client uses towards the AS may not be the same as the one later used towards the RS.

7. Client Information Format

This section addresses the second sub-problem of Section 2.3, and describes a format for client information. Two types of client information are defined in this section:

1. Client Information related to the Push scheme, which encodes keys and related parameters, that allow the Client to communicate securely with the RS and prove that it is the legitimate subject of some authorization information.

2. Client Information related to the Client-Pull scheme represent keys and related parameters that allow the Client to access and verify a resource representation protected with object security.

For the Push scheme, the CIF must be able to encode instructions to the Client on how secure the connection between Client and RS, and how to transfer the authorization information to the RS.

7.1. CIF for the Push scheme

The encoding of the client information in a Push scheme is an array with the following elements:

1. The type of this client information, here the string 'push'.

2. The method to be used for securing the transfer. Either the string 'coaps' (for CoAP over DTLS) or the string 'oscoap' (for CoAP with object security).

3. The method of transferring the authorization information. Either the string 'suppl' (for supplemental data) or the URI of the resource on the RS to which the authorization information should be transferred.

4. The type of key to be used for coaps of oscoap. This can either be a raw public key denoted by the string 'rpk', or a pre-shared key denoted by the string 'psk'.

5. If the key type was 'psk', then the next element is the identifier of this key. If the key type was 'rpk' this element is the raw public key of the RS. The encoding here should follow the SubjectPublicKeyInfo defined in RFC 7250 [RFC7250].

6. If the key type was 'psk', then the next element is the pre-shared key that the Client should use either for DTLS or object security. The encoding of this key depends on the representation format. For CBOR it is raw bytes, for JSON it is Base64 encoded bytes.

Figure Figure 10 shows an example representation of the CIF for a Push scheme, using JSON.

```
["push", "oscoap", "/authorize", "psk",
"someKeyId", "Z4LXBNbOeeOJOYglBLb4pg"]
```

Figure 10: CIF example for the Push scheme

7.2. CIF for the Client-Pull scheme

The encoding of the CIF for a Client-Pull scheme is an array with the following elements:

1. The type of this client information, here the string 'cpull'.

2. The public key of the RS used to sign the resource representation. The encoding here should follow the SubjectPublicKeyInfo defined in RFC 7250 [RFC7250].

3. The secret key used to encrypt the resource representation. The encoding of this key depends on the representation format. For CBOR it is raw bytes, for JSON it is Base64 encoded bytes.

Note that further parameters are provided in the secure object itself, such as e.g. algorithm identifiers and initialization vectors.

Figure Figure 11 shows an example representation of the CIF for a Client-Pull scheme, using JSON.

```
       ["cpull", "MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEJG4m93
       ED8tPK2CpkhrrtKNx1vXDbml4Fun1628Qkl1U_aIB5zUfqPwaaczn
       bqoMJ6vQVZ74X9HpfynouLK_ujw", "eyJhbGciOiJIUzI1NiIsIm"]
```

Figure 11: CIF example for the Client-Pull scheme

8. Message Processing

This section puts together the pieces from previous sections and specifies more in detail the processing steps for the authorization schemes defined in Section 3.

8.1. Unauthorized Access Attempt

All schemes can start with the Client requesting access at the RS without any established authorization information. Depending on which authorization scheme (Push, Pull, Client Pull) the RS has implemented, the RS can either:

- Query the AS using the Pull scheme.

- Instruct the Client to use the Push scheme.

- In case the Client-Pull scheme, provide an encrypted representation of the resource that was requested, without performing any access control.

If the RS wants the Client to use the Push scheme, the RS MUST respond with an error message using the response code 4.01 (Unauthorized). The error message MUST contain information that allows the Client to locate the AS responsible for the requested resource, and other information needed to make an Authorization Request. For example:

```
               4.01 Unauthorized
               Content-Format: application/json
               ["oscoap", "as.example.com/authorize"]
```

The response payload MAY be protected, e.g. signed with the private key of RS, using COSE Mode:PAYL. The response MAY include authentication information of AS, such as the (hash of the) public key of AS. The public key of RS or AS be retrieved in different ways e.g. from a Resource Directory

The initial exchange described in this section may be omitted, and the client can start with the Authorization Request to AS as described in Section 8.2 if the Client has obtained information about the relevant AS in some other way, e.g. using a Resource Directory.

If the RS has access to authorization information about the Client, but it does not apply to the requested resource, the RS MUST answer with an error message using the response code 4.03 (Forbidden). If the RS has authorization information for the Client that applies to the requested resource, but that does not cover the requested action, the RS MUST reply with an error message using the response code 4.05 (Method Not Allowed).

8.2. Authorization Request

Either the Client (Push scheme) or the RS (Pull scheme) can POST an request to the /authorize resource at the AS, specifying the Client's request(s) for authorization using the ARF defined in Section 5.

Upon receiving a POST request to /authorize, the AS MUST perform the following steps:

- o Ensure that the request was received over a secure channel (DTLS) or uses object security. If object security was used the AS MUST perform the necessary verifications.

- o Ensure that the requesting party is authenticated. This is either the Client (as described in Section 4.2) or the RS (as described in Section 4.1). For DTLS this is supposed to have happened during the handshake, for object security this is accomplished by a signature or MAC over the request, produced by the requesting party.

- o Control that the requesting party is authorized to submit this kind of authorization request

If the integrity verification fails, the AS MUST respond with an error message using the response code 4.00 (Bad Request)

If the requesting party is not correctly authenticated, the AS MUST respond with an error message using the response code 4.01 (Unauthorized).

If the requesting party is not authorized to perform this request for authorization information, the AS MUST respond with an error message using the response code 4.03 (Forbidden).

If the evaluation is successful the AS MUST respond with a 2.05 (Content) message code. The response can contain one of the following payloads:

- o If any part of the access requested by the Client was not authorized, the payload MUST be empty.

- o If all parts of the access requested by the Client were authorized, the payload MUST be either in AIF, CIF or both, depending on the authorization scheme:

1. If the Authorization Scheme is Pull, then the payload MUST be in AIF, as defined in Section 6.

2. If the Authorization Scheme is Client-Pull, then the payload MUST be in CIF as defined in Section 7.

3. If the Authorization Scheme is Push, then the payload MUST be an array containing first CIF, then AIF, as defined in Section 7 and Section 6 respectively.

8.3. Receiving Authorization Information

The RS may receive authorization information in a response to a POST to an Authorization Request as described in the previous section.

As an alternative this section describes how to push this information to the RS. This is done with a POST of the authorization information to the /authorize resource on the RS. This request may come from the Client in the Pull scheme, as defined in Pattern 3 in Section 4.1. The request may also come from the AS directly.

Figure 12 illustrates the successful execution of such a request.

```
                   C/AS                            RS
                    |                              |
                    |    POST /authorize AIF       |
                    | ------------------------>    |
                    | <------------------------    |
                    |    2.01 Created              |
                    |                              |
```

Figure 12: POST AIF to /authorize

A third OPTIONAL method is to include the authorization information in a DTLS handshake as supplemental data, as described in Section 4.4.

Irrespective of how the authorization information is received by the AS, upon receiving it, RS MUST verify that it is valid, by doing the following:

- Check that the authorization information comes from a trusted AS
- Check that the host in the authorization information corresponds to itself
- Check that the authorization information is not expired or revoked, if the object security encapsulation allows this
- Verify the integrity of the authorization information

If the validation succeeds, RS MUST store the authorization information and use it to determine the authorization of future requests.

If the authorization information is invalid, RS MUST respond with an error message using the error code 4.03 (Forbidden).

8.4. Receiving Client Information

The Client receives client information in a response to a POST of an authorization request to an AS as described in Section 8.2. The Client MUST make sure that it is communicating with a trusted AS. If the communication is protected with COSE Mode:COAP, the Client MUST verify the authenticity and integrity of the client information, before using it.

8.5. Resource Request and Response

The final resource request is handled differently based on which authorization scheme is implemented. In the Pull scheme this is an unauthorized request as described in Section 8.1. For resources configured to use the Client-Pull scheme for GET access, the RS MUST unconditionally provide an encrypted and signed representation of the requested resource to any requesting Client.

In the Push scheme the RS MUST perform the following steps:

- If the communication is secured with DTLS, verify that the handshake included client authentication. If the communication is protected with Mode:COAP, verify the object security of the request.
- Verify the present authorization information to see if it has data that matches the Client, the requested resource, and the actions

requested on that resource. Note specifically that the binding between the authorization information and the Client is provided either by comparing with the keys used in the DTLS handshake, or with the keys used for object security of the request.

- o If some matching authorization information contains local conditions, verify that these are fulfilled.

If any of these verifications fail, the RS treat the request as specified in Section 8.1.

If all of these verifications succeed, the RS MUST process the request as required by the underlying application.

If the request was protected with Mode:COAP, the RS MUST follow the response protecting scheme for this mode.

9. Security Considerations

The entire document is about security. Security considerations applicable to authentication and authorization in RESTful environments provided in OAuth 2.0 [RFC6749] apply to this work, as well as the security considerations from [I-D.gerdes-ace-actors].

10. IANA Considerations

This document has no actions for IANA yet.

11. Acknowledmgents

Some of the ideas of this document were originally described in a now expired Internet Draft: draft-selander-core-access-control-01. The authors would also like to thank Carsten Bormann, Stefanie Gerdes and Olaf Bergmann for the inspiration drawn from draft-bormann-core-ace-aif [I-D.bormann-core-ace-aif] and [I-D.gerdes-ace-dcaf-authorize].

This work has further drawn inspriation from [OSCAR] which presents another scheme for securing resource request and response using object security and where access key transport is performed by means of RESTful requests to dedicated resources.

12. References

12.1. Normative References

[I-D.schaad-cose-msg]
Schaad, J., "CBOR Encoded Message Syntax", draft-schaad-cose-msg-00 (work in progress), June 2015.

[I-D.selander-ace-object-security]
Selander, G., Mattsson, J., and L. Seitz, "March 9, 2015", draft-selander-ace-object-security-01 (work in progress), March 2015.

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

[RFC4680] Santesson, S., "TLS Handshake Message for Supplemental Data", RFC 4680, October 2006.

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

[RFC7250] Wouters, P., Tschofenig, H., Gilmore, J., Weiler, S., and T. Kivinen, "Using Raw Public Keys in Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", RFC 7250, June 2014.

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

12.2. Informative References

[I-D.bormann-core-ace-aif]
Bormann, C., "An Authorization Information Format (AIF) for ACE", draft-bormann-core-ace-aif-02 (work in progress), March 2015.

[I-D.gerdes-ace-actors]
Gerdes, S., Seitz, L., Selander, G., and C. Bormann, "An architecture for authorization in constrained environments", draft-gerdes-ace-actors-05 (work in progress), April 2015.

[I-D.gerdes-ace-dcaf-authorize]
Gerdes, S., Bergmann, O., and C. Bormann, "Delegated CoAP Authentication and Authorization Framework (DCAF)", draft-gerdes-ace-dcaf-authorize-02 (work in progress), March 2015.

[I-D.ietf-ace-usecases]
Seitz, L., Gerdes, S., Selander, G., Mani, M., and S. Kumar, "ACE use cases", draft-ietf-ace-usecases-04 (work in progress), June 2015.

[OSCAR] Vucinic, M., Tourancheau, B., Rousseau, F., Duda, A., Damon, L., and R. Guizzetti, "OSCAR: Object security architecture for the Internet of Things", Ad Hoc Networks Vol. 32, September 2015.

[RFC2904] Vollbrecht, J., Calhoun, P., Farrell, S., Gommans, L., Gross, G., de Bruijn, B., de Laat, C., Holdrege, M., and D. Spence, "AAA Authorization Framework", RFC 2904, August 2000.

[RFC4949] Shirey, R., "Internet Security Glossary, Version 2", RFC 4949, August 2007.

[RFC6749] Hardt, D., "The OAuth 2.0 Authorization Framework", RFC 6749, October 2012.

[RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, October 2013.

[RFC7159] Bray, T., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, March 2014.

[RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, May 2014.

[RFC7231] Fielding, R. and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, June 2014.

Appendix A. Examples

The following examples show an overview the different authorization schemes presented in Section 3. The notation is as follows:

```
                A -> B     :  POST      {blah}   (request)
               \__ __/       \_ _/      \__ __/  \___ ___/
                  V            V           V         V
              Communication   RESTful   Payload    Type of
              partners and    method               payload
              direction of
              communication
```

Messages protected with COSE Mode:COAP are denoted by adding 'CoseC(Key)', where 'Key' is the key used to protect the message.

Payload protected with COSE Mode:PAYL is denoted by 'CoseP(Key){payload}', where 'Key' is the key used to protect the payload.

Note that we use POST to request authorization information instead of GET, since with the latter, the request would need to be transported in CoAP options (e.g. Uri-Query), and it could potentially become larger than the maximum CoAP packet size. Since CoAP options can not be fragmented this would be extremely problematic.

Example A (using the Push scheme)

1. C -> RS : GET /sensors/tempC {} (request)

2. RS -> C : 4.01 Unauthorized {"oscoap", "as.example.com/ authorize"} (response)

3. C -> AS : POST /authorize CoseC(Key_C-AS) { [{"subjectKeyId ":"Key_C-AS"}, "rs.example.com", [['sensors/tempC', 1(= GET)]]] } (ARF)

4. AS -> C : 2.05 CoseC(Key_C-AS) { ["push", "oscoap", "/authorize", "psk", "someKeyId", "Z4LXBNbOeeOJOYglBLb4pg"], CoseP(Key_AS-RS){ [{"SubjectKeyId":"someKeyId"}, "rs.example.com", [["sensors/ tempC", 1(= GET)]]] } } (CIF, AIF)

5. C -> RS : POST /authorize CoseC(someKeyId) { CoseP(Key_AS- RS){[{"SubjectKeyId":"someKeyId"}, "rs.example.com", [["sensors/ tempC", 1(= GET)]]]} } (AIF)

6. C -> RS : GET /sensors/tempC CoseC(someKeyId) {} (request)

7. RS -> C : 2.05 CoseC(someKeyId) {37.5} (response)

Example B (using the Pull scheme)

1. C -> RS : POST /actuators/doorLock CoseC(someClientKey) {open} (request)

2. RS -(DTLS)-> AS : POST /authorize { [{"subjectKeyId":"someClientKey"}, "rs.example.com", [['actuators /doorLock', 2(= POST)]]] } (ARF)

3. AS -(DTLS)-> RS : 2.05 { [{"subjectKeyId":"someClientKey"}, "rs.example.com", [['actuators/doorLock', 2(= POST)]]] } (AIF)

4. RS -> C : 2.04 (response)

Example C (using the Client-Pull scheme)

1. C -> RS : GET /sensors/tempC {} (request)

2. RS -> C : 2.05 { CoseP(someKey){39.2} } (response)

3. C -> AS : POST /authorize CoseC(Key_C-AS) { [{"subjectKeyId ":"Key_C-AS"}, "rs.example.com", [['sensors/tempC', 1(= GET)]]] } (ARF)

4. AS -> C : 2.05 CoseC(Key_C-AS) { ["cpull", "RS_PublicKey", "someKey] } (CIF)

The authorizon schemes defining these flows are introduced in Section 3, and detailed in Section 8. The message structures for ARF, AIF and CIF are introduced in Section 5, Section 6, and Section 7.

Authors' Addresses

Ludwig Seitz
SICS
Scheelevaegen 17
Lund 223 70
SWEDEN

Email: ludwig@sics.se

Goeran Selander
Ericsson
Faroegatan 6
Kista 164 80
SWEDEN

Email: goran.selander@ericsson.com

Malisa Vucinic
STMicroelectronics
850 Rue Jean Monnet
Crolles 38920
FRANCE

Email: malisa.vucinic@st.com