CoRE Working Group
Internet-Draft
Intended Status: Informational
Expires: August 18, 2014

G. Selander
M. Sethi
Ericsson
L. Seitz
SICS Swedish ICT

February 14, 2014

Access Control Framework for Constrained Environments
draft-selander-core-access-control-02

Abstract

The Constrained Application Protocol (CoAP) is a light-weight web transfer protocol designed to be used in constrained environments. Transport layer security for CoAP has been addressed with a DTLS binding for CoAP. This document describes a generic and dynamic access control framework suitable for constrained devices e.g. using CoAP and DTLS. The framework builds on well known paradigms for access control, externalizing authorization decision making to unconstrained nodes while performing authorization decision enforcement and verification of local conditions in constrained devices.

Status of this Memo

Table of Contents

1. Introduction

The Constrained Application Protocol (CoAP) [I-D.ietf-core-coap] is a light-weight web transfer protocol, suitable for applications in embedded devices used in services such as smart energy, smart home, building automation, remote patient monitoring etc. Due to the nature of the these use cases including critical, unattended infrastructure and the personal sphere, security and privacy are critical components. Authentication and authorization aspects of such use cases are discussed in [I-D.seitz-ace-usecases].

CoAP message exchanges can be protected with different security protocols. The CoAP specification defines a DTLS [RFC6347] binding for CoAP, which provides communication security services including authentication, encryption, integrity, and replay protection.

The CoAP specification sketches an approach for authorization and access control - i.e. controlling who has access to what - using static access control lists, which are assumed to have been provisioned to the devices and which contain lists of identifiers that may start DTLS sessions with the devices.

There are some limitations inherent to such an approach:

1. By restricting the scope of access control to the granularity of identifiers of requesting clients, it is not possible to give different privileges to different entities that are allowed to access the same device. For example, it may be desirable to give some clients the right to GET resources but others the right to POST or PUT resources to the same device; or to give the same client different access rights for different resources on the same device.

2. There are use cases [I-D.seitz-ace-usecases] where the granularity of GET/PUT/POST/DELETE is not sufficient to specify the relevant access restrictions. For example, an access policy may depend on local conditions of the device such as date and time, proximity, geo-location, detected effort (press 3 times), or other aspects of the current state of the device.

3. It is not defined how to change access privileges except by re-provisioning. How such changes would be authorized is also unclear.

This document proposes a framework that allows fine-grained and flexible access control, applicable to a generic setting including use cases with constrained devices [I-D.ietf-lwig-terminology].

1.1 Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [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", "access control", "confidentiality", "credential", "encryption", "sign", "signature", "data integrity", and "verify".

Terminology for constrained environments is defined in [I-D.ietf-lwig-terminology]. These terms include, but are not limited to, "constrained device", "constrained network", and "device class".

Authorization terminology is taken from OAuth 2.0 [RFC6749].

Resource Server (RS): The constrained device which hosts resources the Client wants to access.

Client (C): A device which wants to access a resource on the Resource Server. This could also be a constrained device.

Resource Owner (RO): The subject who owns the resource and controls its access rights.

2. Scope and Requirements

This section defines the scope and gives an overview of the requirements that form the basis for the proposed Access Control Framework.

2.1 Resource Authorization and Protocol Authorization

Access control is protection of system resources against unauthorized access. There are different kinds of "system resources" that needs protection and different kinds of protection mechanisms.

For the purpose of this memo, we distinguish between two types of authorization: "Resource Authorization" and "Protocol Authorization".

- o Resource Authorization (RA) deals with the question whether the server should allow a client to request GET/PUT/POST/DELETE to a resource (where "resource" is as defined in RFC 2616).

- o Protocol Authorization (PA) deals with the question whether the server should engage in a protocol initiated by the client.

Where RA is mainly about protecting the resource, PA is also about protecting the server that hosts the resources. By only granting authorized clients the right to run a protocol, only those clients are able to interact with resources on the server. This also avoids unnecessary protocol processing, thus saving battery and computing resources, and reducing the effect of certain DoS attacks.

In order to enforce authorization the server must be able to verify some property of the requesting client, e.g. its identity or a group membership. PA may e.g. be applied to DTLS as suggested in the CoAP specification or in [I-D.seitz-core-security-modes].

o RA typically implies some PA: If a client is authorized to access a resource hosted on a server, then the client should be allowed to run a protocol (e.g. DTLS) with the server when accessing the resource.

o PA access does not necessarily imply RA: Just because a client is authorized to execute a protocol with the server, the client is not necessarily authorized to access any resources hosted on the server.

The CoAP Security Modes [I-D.ietf-core-coap] and the Additional Security Modes for CoAP [I-D.seitz-core-security-modes] define Access Control Lists with information about what clients are allowed to run DTLS with an origin server. This is by definition Protocol Authorization. However, PA can be used to define RA: For example, by allowing access to all resources for all clients allowed to execute (and successfully complete) an authentication protocol.

The scope of the Access Control Framework defined in this draft is targeting RA, but as is noted above, RA implies that complementing PA needs to be defined.

2.2 Requirements

The Access Control Framework (ACF) for constrained environment as described in this memo shall support the requirements in [I-D.seitz-ace-usecases] and take into account the design considerations in [I-D.seitz-ace-design-considerations]. In particular the ACF should

- o support differentiated access rights for different requesting entities,
- o provide access control at least at the granularity of RESTful resources,

- o allow access rights depending on local conditions (e.g. state of device, time, position),
- o include procedures for authorizing changes and revocation of access rights
- o keep transmission and reception at a minimum in order to reduce energy consumption in constrained devices.

3. Static and Dynamic Access Control

Consider a generic setting where a Client wants to access a resource hosted on a Resource Server, which is potentially a constrained device, and where the access rights are determined by the Resource Owner. (The Client may also be constrained, we return to this in section 4.)

3.1 Static Access Control

3.1.1 ACL for Protocol Authorization

If there are no restrictions on which Client is allowed to access a certain resource, there is no need to perform access control, nor to authenticate the Client. If it does matter which Client is allowed access, then the Resource Server must authenticate some properties (e.g. identity, group membership) of the Client, and also be able to determine if the Client is authorized based on these properties.

One possible access control scheme is that each Resource Server keeps a list of identifiers of authorized clients. The CoAP Security Modes [I-D.ietf-core-coap] Pre-Shared Key and Raw Public Key mention Access Control Lists (ACLs) with information about what clients are allowed to run DTLS with a server, and subsequently access any resource on the server (cf. PA, Section 2.1).

3.1.2 ACL for Resource Authorization

In a more elaborate scheme, the right to access a resource on a server could depend on more parameters, e.g.

- o what resource is requested,

- o what request method (e.g. GET, PUT, DELETE) is used, or

- o local/temporal conditions at the time of the request.

This kind of authorization information can be encoded into an ACL stored in the Resource Server and used to determine if a request should be granted.

3.1.3 Static ACLs

Both schemes described in previous sections require ACLs (access rights) to be provisioned to the Resource Server at one time, and used at a later time to grant resource access. A common assumption is that an ACL is provisioned during deployment and remains valid for the lifetime of the device. We refer to this as Static Access Control.

Static Access Control is adequate for a number of use cases, e.g. when the access rights remain constant throughout the lifetime of the device or when manual provisioning of new access rights after deployment of the device is feasible.

Static Access Control does not address how ACLs can be changed or revoked remotely, nor how such an update would be authorized. In particular for embedded devices this requires special considerations, for example due to

- o the lack of physical access to the device (e.g. due to devices built into infrastructure), and/or
- o the infeasibility of manual provisioning procedures (e.g. due to the large quantity of devices).

3.3 Dynamic Access Control

In this section we address use cases for which Static Access Control is not sufficient [I-D.seitz-ace-usecases], e.g. to grant access to new Clients or change access rights some time after deployment.

3.3.1 Rationale

The flexibility required by a Resource Owner in assigning access rights implies that static ACLs need to be replaced by more general access control policies. However, managing and evaluating arbitrary access control policies is typically too heavyweight for constrained devices. As a consequence we assume that the policy management and the authorization decision making is externalized to a less constrained node, called the Authorization Server (AS), acting on behalf of the Resource Owner who defines the access control policies governing the decisions of the AS. The AS may potentially be implemented in many different kinds of physical nodes, e.g. as a server in the cloud or a relatively unconstrained portable device such as a smartphone.

While authorization decision and policy management is outsourced to the AS, access control enforcement should be performed in a trusted environment associated to the resource and as close to the resource as possible, in order to provide end-to-end security between resource and authorized client.

Moreover, verifications of any local conditions should be performed in conjunction with accessing the resource for the following reasons:

- o Transferring information about local conditions in the Resource Server to the Authorization Server for each policy decision adds to the communication costs for the Resource Server, and unnecessarily so if the decision is "not granted".

- o The local conditions in the Resource Server may have changed at the time of access, so the decision would be based on outdated information.

We therefore suggest that access control decision enforcement and verification of local conditions should take place in the Resource Server, or in a proxy-type device offloading a severely constrained device hosting the resource. Local conditions may be expressed as constraints under which an externally granted authorization decision is valid, and which are verified at the time and location of access.

We use the term Dynamic Access Control refer to the setting where information about authorization decisions and/or access policies is transferred from the Authorization Server to the Resource Server.

Authorization decisions (potentially including local conditions) are conveyed from the Authorization Server to the Resource Server in Access Tokens, which are objects containing authorization information related to a client. Access tokens are produced by an Authorization Server and consumed by a Resource Server, which processes the access token and caches or stores information about the access rights.

NOTE

The terminology "Authorization Server" and "Access Token" is taken from OAuth 2.0 [RFC6749]. The feasibility to implement the access control in constrained environments using OAuth is for further study.

3.3.2 Access Tokens

There may be different types of authorization decision content in an access token, we consider two cases:

- o An access token may be a Capability Token, i.e. a list of one or more resources and associated request methods (GET/PUT/POST/DELETE) which the client is granted. See Appendix A for an example of format of a capability list-based access token (also expressing a local condition). Other examples of formatting capability lists can be found in [I-D.bormann-core-ace-aif].

- An access token may be an assertion about a group membership of the client (a Group Membership Assertion), for which the access rights are specified in form of a Group ACL on the Resource Server, see 3.3.3. For an example of a group membership assertion see [I-D.gerdes-core-dcaf-authorize].

Transfer of access tokens, potentially via intermediary nodes, is discussed later in this document.

3.3.3 Group ACLs

One purpose of the AS is to outsource policy management from the RS. However, for frequently recurring requests requiring a common set of access rights it is beneficial to store in the RS local access policies which can be compactly represented and easily evaluated, such as ACLs.

In order to avoid identity management at the level of the RS, such ACLs should refer to groups (or "roles") instead of specific subject identifiers. We refer to these ACLs as Group ACLs, since they contain group identifiers as subjects rather than client identifiers. When there is no risk for confusion we will simply call them ACLs.

A group ACL is used in conjunction with a group membership assertion (see 3.3.2) on the RS. Together they associate a Client to a resource access permission associated with the group which the Client is member in.

Furthermore, group ACLs themselves should be represented as resources on the RS which can be accessed by the AS. Updates of ACLs should be performed by the AS only, and should be implemented by PUT or POST to the ACL resources on the RS.

3.3.4 Trust model

The Authorization Server must be trusted by all involved parties, in particular the Resource Owner must trust the AS to enact the access policies as specified. The Resource Server must trust the access tokens to express rights given by the Resource Owner, and that updates on ACLs performed by the AS are done on behalf of the Resource Owner.

In order to secure the access token transport and to be able to authenticate requests from the AS, we assume that the Resource Server has established a shared secret key or authentic public key of the AS. How this key is established is out of scope for this memo.

The Authorization Server being a Trusted Third Party can also support authentication between Client and Resource Server, by means of e.g. key distribution functionality (cf. Kerberos [RFC4120]). The feasibility to implement access control in constrained environments using authorization extensions to Kerberos is for further study.

4 Access Control Framework

The Access Control Framework detailed in this section targets Dynamic Access Control for Resource Authorization.

4.1 Entities

The relevant entities are:

- o An Authorization Server (AS) performing the authorization decision making, based on the access control policies, and sharing one or more trusted keys from the Resource Server.

- o A potentially constrained Resource Server (RS) hosting resources and provisioned with one or more trusted keys from the AS.

- o A potentially constrained Client (C) wishing to access a resource. As there may be intermediaries, e.g. forward proxies, the actual CoAP client requesting the RS may be different from the Client. When we want to emphasize the original source of the request we use the term "Origin Client" (OC).

- o An Access Manager (AM) which requests and receives access tokens from an AS. The AM may be a standalone node or integrated/co-located with the C. Constrained clients may need support to acquire access tokens, in which case the Access Manager is implemented on a separate node.

4.2 Message flow example

One example procedure for resource access is shown in Figure 1 and described below. The setting is a Client wishing to access a resource for which it is authorized, but which the RS is not aware of. Once the RS has stored a new access token, the message flow reduces to step 8.

```
                         Access            Authorization    Resource
      Client             Manager               Server        Server
        +                  +                     +              +
        |---(1) AuthZ ---->|                     |              |
        |     Request      |<-(2) Authenticate ->|              |
        |                  |                     |              |
        |                  |-(3) Request token ->|              |
        |                  |                     | (4)          |
        |                  |                     | Evaluate     |
        |                  |                     | access       |
        |                  |                     | control      |
        |                  |                     | policies     |
        |                  |<---(5) Token, ------|              |
        |                  |   Base Credentials  |              |
        |<---(6) Token, ---|                     |              |
        | Base Credentials |                     |              |
        |                  +                     +              |
        |---------------(7) Store Token Request --------------->|
        |<-------------------- Response ------------------------|
        |                                                       |
        |------------------(8) Resource Request --------------->|
        |<-------------------- Response ------------------------|
```

Figure 1: Roles and access control procedure

The C sends an authorization request to the AM (1).

The AM authenticates to the AS (2) on behalf of the C. The AM then requests an access token, and optionally Base Credentials for a specific security mode (3). The request contains the C’s subject identifier which is used to evaluate the access control policies.

The AS makes the authorization decision on behalf of the Resource Owner (4) and, if granted, responds (5) to the AM with an access token bound to the C’s subject identifier. Optionally it also sends Base Credentials to be used in the message exchange between C and RS. A Base Credential may e.g. be the public key of the RS, a public key certificate generated for the C, or a derived key bootstrapping the trust relation between the AS and RS [I-D.seitz-core-security-modes].

The AM forwards the access token and Base Credentials to the OC (6).

The OC sends the access token (see 4.3.3) to the RS (7). After the token is verified by the RS (see 4.3.4) its content is stored and the RS responds appropriately to the OC.

The OC submits Resource Request(s) (8), which are verified against the stored access token content (and potentially Group ACLs) by the RS. If the RS finds a matching grant, and all local conditions are met, the request is processed and a response is sent. Steps (7)-(8) could potentially be combined in one request-response.

Communication security is not detailed in this message flow and depends on several factors. E.g. if the Base Credentials are secret keys, then the communication between C and AM, and between AM and RS must be confidential.

Request and Response messages need to be protected, either using communication security, such as DTLS [RFC6347], or object security, such as JWE [I-D.ietf-jose-json-web-encryption] and JWS [I-D.ietf-jose-json-web-signature]. The Base Credentials that AS optionally provides, can be used to establish the cryptographic keys for and object security scheme, or Protocol Authorization for, say, DTLS.

A detailed proposal can be found in [I-D.gerdes-core-dcaf-authorize].

4.3 Access Tokens

In 3.3.2 we listed two alternative access tokens: capability token and group membership assertion. In this section we discuss the content, protection, transfer, reception, and storage of these kinds of access tokens.

4.3.1 Requirements for Access Tokens

Access tokens must be integrity protected by the AS such that it can be verified by the RS using a trusted key (see 4.3.2), and furthermore they should enable the RS to enforce the authorization decision. Hence the access token should provide the following information:

- o Which OC does the decision apply to (subject identifier), and how can this OC be authenticated (if necessary).

- o Which AS has created this access token (issuer). This information may be implicit from the signature of the token.

- o A sequence number which, together with the issuer, is unique for a given RS.

The token can also specify under what other conditions it is valid (local conditions evaluated by the resource server at access time, e.g. expiration, number of uses).

In addition to this, a capability list also needs to specify:

- o Which resources does the decision apply to.
- o Which request methods (GET, PUT, POST, DELETE) does the decision apply to.

A capability token may state specific allowed values, for PUT and POST methods (e.g. if the client is only allowed to set values 1 and 2 not 0 and 3 for certain actuator).

4.3.2 Access Token Protection

Since access tokens are to be consumed by constrained devices, the protection of the access token must be lightweight and compact. For example JSON Web Signatures (JWS) [I-D.ietf-jose-json-web-signature] can be used as a means of signing access tokens, specifically with the JWS Compact Serialization.

In an object security setting, where the token may be transferred over an insecure channel, it can be encrypted and integrity protected using JWE [I-D.ietf-jose-json-web-encryption].

An alternative, potentially more compact encoding format would be CBOR [RFC7049], however it would require corresponding signature and encryption schemes.

Using an asymmetric signature scheme is recommended if intermediary nodes, between OC and RS, are expected to verify the access token, since it is less security critical to provision public keys to the intermediary nodes, rather than symmetric keys. This allows an intermediary to discard certain invalid requests (expired/spoofed access tokens, etc.) without sharing a secret key with the RS.

4.3.3 Access Token Transfer

The access token can be transferred from the OC to the RS in different ways.

A. One possibility is to extend the communication security establishment protocol (e.g. using TLS Handshake Message for Supplemental Data [RFC4680] in DTLS).

B. Another possibility is to use the application protocol (e.g. CoAP) and send the access tokens as regular requests, i.e. PUT the access token to a dedicated token storage resource.

In either case the access token is verified upon reception, and if it is valid (see 4.3.4), its content is stored (see 4.3.5) for being used in a subsequent resource request (see 4.3.6). If the access token is not valid the RS aborts the corresponding protocol to avoid unnecessary processing. This saves resources in the case A above, since the communication between RS and OC is still in a very early stage. However, early abort of communication establishment can also be achieved by protocol authorization, see e.g. [I-D.seitz-core-security-modes]. Moreover one drawback with case A is that a new session has to be established if the same OC needs to submit a new access token to the RS.

For these reasons implementations should at least support the transfer of access tokens in the application layer protocol. For this to work, the C needs to know the token storage resource on the RS. This information can be provided by the AS in step 5 of figure 1. Writing to this location should not require Resource Authorization. Instead, there are verifications of the access token done on reception as is discussed in the next section.

4.3.4 Access Token Reception

Upon receiving an access token which is not already stored the RS shall perform the following processing:

- o Verify if the token is revoked
- o Verify if the token is from a trusted issuer (i.e. an AS known to the RS)
- o Verify the Message Authentication Code or signature of the token using a trusted AS key

In order to support access token revocation the RS shall maintain a list of sequence numbers per issuer, specifying the revoked tokens. If the access token passes the verifications, we denote it ’valid’. The RS shall only store valid access tokens. Revoked tokens shall be removed from storage.

Optionally the RS can use the sequence number of the token, to enforce token expiration. This can be done by rejecting sequence numbers that are significantly lower than the highest sequence number the RS has received so far.

Optionally the RS can use the time lapse since received to enforce token expiration. This can be done by storing together with the token the local time as measured by the RS upon reception.

4.3.5 Access Token Storage

If the received access token is valid its content should be stored. Independently of case A or B in section 4.3.3, the content of the token should be handled in the same way.

The token should be stored in a dedicated token storage resource, the signature should be removed from the token before storage. Expired or revoked tokens should be purged from the token storage.

4.3.6 Access Token Enforcement

Upon receiving a request, the RS shall perform the following processing on the relevant stored token:

- o If there is information about expiry, verify if the stored token has expired
- o Verify that the stored token is bound to the requesting subject
- o Verify that the stored token authorizes the received request (including local conditions), this may include matching group memberships specified in the token to group ACLs on the RS.

If no matching token is found, the request must be rejected using the response code 4.03 Forbidden.

Keys or identifiers established in the communication security protocol can be used to support subject binding verification. Table 1 shows examples of token subject identifiers based on different CoAP security modes (see also section 9 of [I-D.ietf-core-coap], [RFC4279] and [I-D.seitz-core-security-modes]).

CoAP security mode	Token subject identifier
PreSharedKey	psk_identity
RawPublicKey	public key fingerprint
Certificate	Subject DN
DerivedKey	psk_identity
AuthorizedPublicKey	public key fingerprint

Table 1: DTLS parameters as token subject identifiers

5. Intermediary processing and notifications

This section describes the security implications of intermediary processing and notifications for access control.

5.1 Intermediary nodes

There may be intermediary nodes between OC and RS, including forward proxies, reverse proxies, cross-proxies, gateways, etc. From an access control point of view the RS should be able to verify that a received request is originating from the OC referenced in the received access token. This has implications on the access token and message protection.

We distinguish between the end-to-end security setting where no intermediary nodes need be trusted and the hop-by-hop security setting where at least one intermediary node must be trusted.

DTLS generally needs to be hop-by-hop in case of proxies, this requires some degree of trust in a proxy which may not be acceptable for some applications. A RS sending back the response via the forward proxy trusts the forward proxy with the plain text response (e.g. a GET response) and that the proxy has established secure communication with the OC.

In the hop-by-hop case, neither DTLS nor CoAP offers any means for RS to authenticate the OC.

If the RS has established DTLS with a forward proxy which proxies requests from an OC, then the access token can be signed by the OC in addition to the AS integrity protection. Though the RS can not authenticate the OC directly, it can infer from a correctly signed valid and fresh access token that the OC is not only authorized but also has the intent to perform the request.

5.2 Mirror Server

The access control framework can also be applied to the scenario where a mirror server as defined in [I-D.vial-core-mirror-proxy] is present. In such a scenario, each RS behaves as a client of the mirror server. The access control enforcement in this case, would be made at the mirror server instead of in a constrained RS, and the trusted AS keys would have to be provisioned to the mirror server. However, to a client wishing to access a resource, the mirror server behaves as any other RS and is indistinguishable (transparent), thereby requiring no change for the communication between client and the mirror server. The communication between the mirror server and the constrained RS may or may not be secured, and is oblivious to the protocols used between the client and the mirror server.

5.3 Observe

The access control framework can also be applied, as it is, in the case where the CoAP observe option [I-D.ietf-core-observe] is used. With the observe option, clients can register an interest in a particular resource by sending a CoAP request containing the observe option to a RS. The RS would in this case maintain the state information for this expressed interest and send responses on state changes only as long as the access token and local conditions in the ACL are valid. The local conditions may need to be verified at each state change. Once the access token expires, the RS will remove any state information for the interest expressed. Also, the RS will notify the OC by sending a notification with 4.01 (Unauthorized) response code and the notification will not include an Observe Option. The OC would then have to transfer a new access token demonstrating that it is allowed access and send a new CoAP request with an observe option expressing interest.

6. Security Considerations

The present framework aims to protect the resources on RS, the servers themselves, and the services offered. The means proposed to protect these assets is to enforce granular access restrictions on accessing the devices. Due to the setup of the framework, there is also a need to protect the authorization decisions and the keys used to protect the entire resource access procedure.

The AS is a Trusted Third Party from the point of view of the resource owner. If the AS is compromised, it could e.g. issue access tokens to unauthorized parties.

Since the AM requests tokens on behalf of the OC, the AS must be able to verify that it really represents the OC.

In order to enforce a policy decision, the RS must authenticate the OC, and match the identifier of the authenticated entity with the subject identifier of the access token.

While DTLS offers bundled encryption and integrity protection of both payload and headers, an object security approach allows for a trade-off between protection against performance. Depending on the trust model, access token and payload may need to be encrypted because eavesdropping will reveal information about the OC’s request, which may be privacy sensitive. Wrapping of the payloads as secure objects allows differentiated protection of the content based on its sensitiveness.

A typical access token may have a size in the order of hundreds of bytes. If tokens can be sent to the RS by unauthenticated clients, care must be taken to prevent that the processing and storage of the token opens for Denial of Service attacks.

7. IANA Considerations

This document has no actions for IANA.

8. Acknowledgements

The authors would like to thank Stefanie Gerdes, Mats Naeslund, John Mattsson and Sumit Singhal for contributions and helpful comments.

9. References

9.1 Normative References

[I-D.ietf-core-coap]
Shelby, Z., Hartke, K., Bormann, C., and B. Frank, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-18 (work in progress), June 2013.

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

9.2 Informative References

[I-D.seitz-ace-usecases]
Seitz, L., Gerdes, S., and Selander, G., "ACE use cases", draft-seitz-ace-usecases-00 (work in progress), February 2014.

[I-D.ietf-lwig-terminology]
Bormann, C., Ersue, M., and Keranen, A., "Terminology for Constrained Node Networks", draft-ietf-lwig-terminology-07 (work in progress), February 2014.

[I-D.seitz-ace-design-considerations]
Seitz, L., and Selander, G., "Design Considerations for Security Protocols in Constrained Environments", draft-seitz-ace-design-considerations-00 (work in progress), February 2014.

[I-D.seitz-core-security-modes]
Seitz, L., and Selander G., "Additional Security Modes for CoAP", draft-seitz-core-security-modes-00 (work in progress), October 2013

[I-D.ietf-jose-json-web-encryption]
Jones, M., Rescorla, E., and Hildebrand J., "JSON Web Encryption (JWE)", draft-ietf-jose-json-web-encryption-20 (work in progress), January 2014.

[I-D.ietf-jose-json-web-signature]
Jones, M., Bradley, J., and Sakimura N., "JSON Web Signature (JWS)", draft-ietf-jose-json-web-signature-20 (work in progress), January 2014.

[I-D.bormann-core-ace-aif]
Bormann, C., "An Authorization Information Format (AIF) for ACE", draft-bormann-core-ace-aif-00 (work in progress), January 2014.

[I-D.gerdes-core-dcaf-authorize]
Gerdes, S., Bergmann, O., and Bormann, C., "Delegated CoAP Authentication and Authorization Framework (DCAF)", draft-gerdes-core-dcaf-authorize-01 (work in progress), October 2013.

[I-D.vial-core-mirror-proxy]
Vial, M., "CoRE Mirror Server", draft-vial-core-mirror-proxy-01 (work in progress), July 2012.

[I-D.ietf-core-observe]
Hartke, K., "Observing Resources in CoAP", draft-ietf-core-observe-11 (work in progress), October 2013.

[RFC4949] Shirey, R., "Internet Security Glossary, Version 2", FYI 36, RFC 4949, August 2007.

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

[RFC4120] Neuman, C., Yu, T., Hartman, S., and K. Raeburn, "The Kerberos Network Authentication Service (V5)", RFC 4120, July 2005.

[RFC6749] Hardt, D., Ed., "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.

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

[RFC4279] Eronen, P., Ed., and H. Tschofenig, Ed., "Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)", RFC 4279, December 2005.

Appendix A. Example Token Syntax

In this section we give an example of an access token using a compact

JSON notation. The intent with this example is mainly to demonstrate potential content and structure of a token.

```
{
  "SN": "081d5ff7bb2c2d08",
  "IS": "6f",
  "SI": "435143a1b5fc8bb70a3aa9b10f6673a8",
  "LCO": {
     "NB":"09:00:00Z",
     "NA":"17:00:00Z"
  },
  "MET": "POST",
  "VAL": "open",
  "RES": "node346/doorLock"
}
```

Token element	Encoding
Sequence number	SN
Issuer	IS
Subject identifier	SI
Local conditions	LCO
Request method	MET
Allowed payload value	VAL
Resource	RES

Table 2: Token elements encoding

In this example the issuer is identified by a single byte, this is possible because the token is for a specific RS, which is not expected to have more than 256 distinct trusted AS.

The subject identifier is a public key fingerprint binding the token to the corresponding public key, which in turn could be used to establish a DTLS connection to the RS using the RawPublicKey security mode (see section 9 of [I-D.ietf-core-coap]).

The local condition specifies a time frame during which the token is valid (NB = not before, NA = not after). The syntax and semantics of such conditions must be pre-defined on the consuming RS so that it can parse and enforce them.

The RESTful request method (DELETE, GET, POST, PUT) that this token authorizes is specified in the MET element, while the resource specifies the URI host and URI path from the CoAP requests. We do not consider it useful to specify the scheme (coap, coaps) or the query parts of a resource URI, the latter since queries are very resource dependent and it is probably difficult to write meaningful access policies on specific query values.

For actions including a payload (typically PUT and POST), the token can specify a restriction on the allowed payload value.

Note that JSON is used here because it gives a human readable token format, for production deployments one should consider using a more compact representation format such as CBOR [RFC7049] to reduce the token size. Other examples of access token formats are provided in [I-D.gerdes-core-dcaf-authorize].

Appendix B. Changelog

Changes from -01 to -02:

- o Further shortening of the draft by referencing separate drafts.
- o Distinction between Static and Dynamic Access Control
- o Discussion of ACLs and groups

Changes from -00 to -01:

- o The draft is significantly shortened, content is moved to separate drafts and much informational content has been removed.
- o The limited use case descriptions are greatly expanded and moved into a separate draft [I-D.seitz-ace-usecases].
- o The key provisioning schemes are generalized to alternate CoAP security modes and described in a separate draft [I-D.seitz-core-security-modes]
- o The ACL categories are replaced by the distinction between protocol authorization and resource authorization.
- o The Access Manager functionality originally defined in [I-D.gerdes-core-dcaf-authorize] is introduced.
- o The communication security profile description is removed. For a detailed DTLS based access control setting see [I-D.gerdes-core-dcaf-authorize].

o The object security profile is planned for a future draft.

Authors' Addresses

Goeran Selander
Ericsson
Farogatan 6
16480 Kista
SWEDEN

EMail: goran.selander@ericsson.com

Mohit Sethi
Ericsson
Hirsalantie 11
02420 Jorvas
FINLAND

EMail: mohit.m.sethi@ericsson.com

Ludwig Seitz
SICS Swedish ICT AB
Scheelevagen 17
22370 Lund
SWEDEN

EMail: ludwig@sics.se