CoRE Working Group
Internet-Draft
Intended Status: Informational
Expires: April 24, 2014

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

October 21, 2013

Access Control Framework for Constrained Environments
draft-selander-core-access-control-01

Abstract

The Constrained Application Protocol (CoAP) is a light-weight web transfer protocol designed to be used in constrained nodes and constrained networks. Communication security support for CoAP, including authentication, encryption, integrity protection, is specified by means of a DTLS binding for CoAP, but authorization and access control are not described in detail.

This document describes a generic and dynamic access control framework suitable for constrained environments e.g. using CoAP. The framework builds on standards and 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

http://www.ietf.org/shadow.html

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. Use cases for CoRE security are discussed in [I-D.seitz-core-sec-usecases].

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

Authorization and access control - i.e. controlling who has access to what - is addressed with 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-core-sec-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".

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".

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

- Protocol Authorization (PA) deals with the question whether the server should allow a client to run a certain protocol with it.

RA is mainly about granting only authorized requests for a resource, and protecting resource related communication between authorized client and server

PA is mainly about protecting the device hosting the resource and avoiding unnecessary protocol processing e.g. to save battery / computing resources or to protect against certain DoS attacks.

Although there are overlapping and degenerate cases, we believe this distinction is useful to understand the purpose of authorization.

There may be dependencies between PA and RA:

- RA typically imply 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 (say DTLS) with the server for accessing the resource.

- PA access does not necessarily imply RA: Just because a client is authorized to execute a protocol (say DTLS) 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 successfully executing the protocol.

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

2.2 Requirements

The Access Control Framework SHALL support

- o access control in a constrained environment with constrained devices or networks, in particular,
  - additional messages exclusively for performing access control SHOULD be kept at a minimum to reduce power consumption on the constrained devices.

- o access control applicable to a variety of use cases and access purposes, in particular
  - differentiated access rights for different requesting entities,
  - access control at least at the granularity of RESTful resources,
  - access policies based on local conditions (e.g. state of device, time, position),

- o changes to access policies without re-provisioning, and

- o interoperability between different system components and providers, which is best achieved by the use of state-of-the-art security standards and best practices (in particular end-to-end security between resource and authorized client).

The Access Control Framework SHALL be compatible with a large variety of client-server authentication methods, message protection mechanisms (communication/object security), device key management

procedures and trust anchors (secret keys, raw public keys, certificates).

3 Outline of Access Control Framework

In this section we present the rationale leading to our access control framework, the resulting roles, the message flow, and the access control procedure.

3.1 Rationale

Consider a generic setting where a CoAP client wants to access a resource hosted on a CoAP server, which is potentially a constrained device, and where the access rights are determined by the owner of the resource. In this section we introduce some of the terms and procedures and provide some rationale for those.

Managing and evaluating arbitrary access control policies is in general too heavyweight for constrained devices. As a consequence the main authorization decision is externalized to a less constrained node, called the "authorization server", acting on behalf of the resource owner.

On the other hand, 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:

- Transporting information about local conditions in the device to an authorization server for each policy decision (or on a regular basis) introduces delays and/or adds additional messages exclusively for the purpose of performing access control.

- Local conditions may have changed at the time of enforcement.

We therefore target enforcement and local decisions to take place in

the constrained device hosting the resource, or in a proxy-type device offloading a severely constrained device hosting the resource.

We express local conditions as constraints under which an externally granted authorization decision is valid, and which are verified at the time and location of enforcement.

In order to convey the authorization decisions (including local conditions) from the authorization server to the device where access control is enforced, we use integrity protected data objects, which we call "access tokens" (or just "tokens").

Access tokens are acquired from the authorization server and used to gain access to the device. We denote by "access manager" the function of requesting and receiving access tokens from an authorization server. Constrained clients may need support to acquire tokens, in which case the access manager is implemented on a separate node.

NOTES

1. The authorization server must be trusted by all involved parties, in particular by the resource owner. We assume that this trust relationship is manifested through trusted keys established a priori in the constrained device and the authorization server.

2. The existence of such a trust relationship, once introduced to support authorization and access control, can be utilized to optimize key establishment, authentication and message protection between a client and the origin server.

3.2 Roles

The relevant roles involved in this access control framework are:

- A Resource Owner specifying the policies for access to the resources.

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

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

- o A potentially constrained Origin Client (OC) wishing to access to a resource. As there may be client intermediaries, e.g. forward proxies, the actual CoAP client requesting the RS may be different from the origin client. When there are no other clients to confuse with, we refer to the origin client simply as "the client".

- 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 OC.

3.3 Message flow

The default procedure for resource access is described in Figure 1, and works as follows:

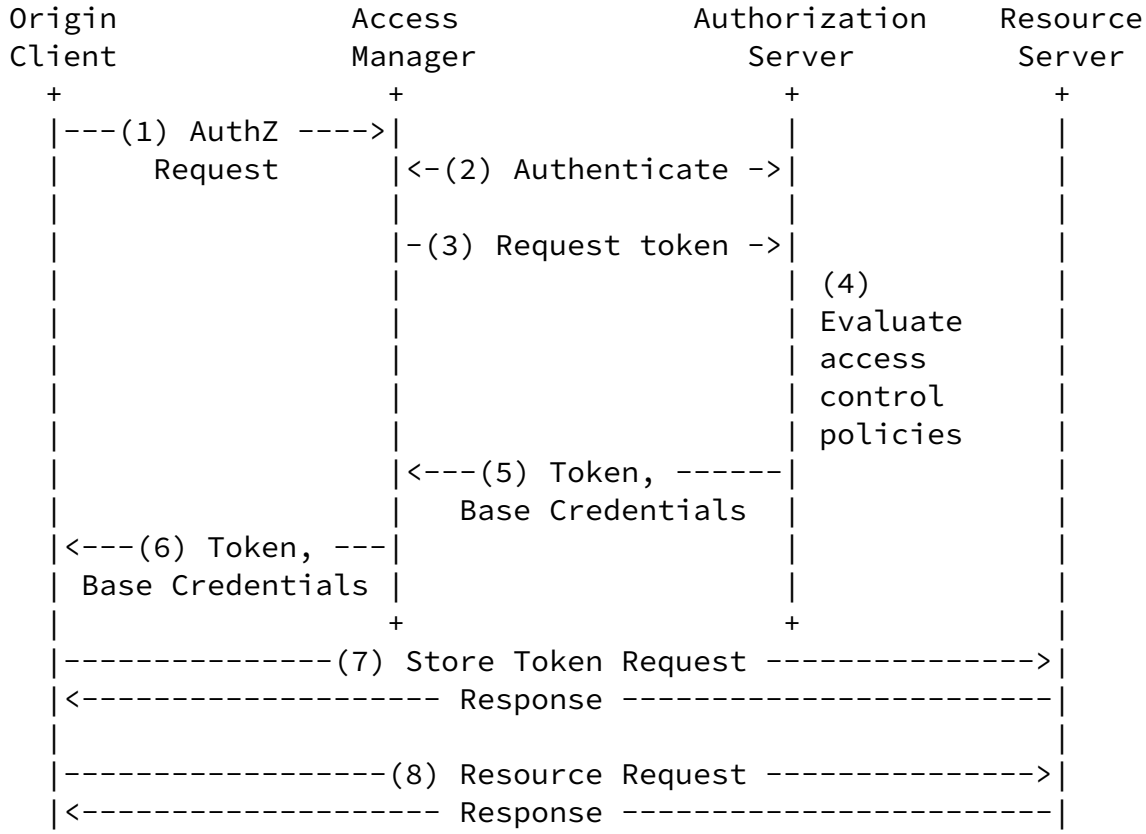

Figure 1: Roles and access control procedure

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

The AM authenticates to the AS (2) on behalf of the OC. The AM then

requests an access token and optionally base credentials for a specific security mode (3). The request contains the OC'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 OC's subject identifier. Optionally it also sends Base Credentials to be used in the message exchange between OC and RS. A base credential may e.g. be the public key of the RS, a public key certificate generated for the OC, 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 stores the token on the RS (7). The RS provides a well-known resource for submitting and storing tokens used to authorize future requests. After the token is verified by the RS it is stored and the RS responds appropriately to the OC.

The OC submits Resource Request(s) (8), which are verified against the stored tokens by the RS. If the RS finds a matching token, and all local conditions are met, the request is processed and a response is sent.

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 the message protection scheme, and protocol authorization for communication security establishment.

4. Access Tokens

The access token is a secure object containing authorization information passed from the AS via the AM and OC to the RS. In this section the content, protection and transport of an access token is discussed.

4.1 Requirements

In order to enable the RS to enforce the authorization decision, the access token MUST provide the following information:

- Which resource does the decision apply to.
- Which action (GET, PUT, POST, DELETE) does the decision apply

to.

- Which OC does the decision apply to (subject identifier), and how can this OC be authenticated (if necessary).
- Which AS has created this access token (issuer). This information MAY be implicit from the signature of the token.
- Under what other conditions is the access token valid (local conditions evaluated by the resource server at access time, e.g. expiration, number of uses).

The access token SHALL include a sequence number which together with the issuer, is unique for a given RS. The token MAY state a specific allowed payload value, where applicable (e.g. for PUT/POST requests).

The access token MUST be integrity protected by the AS such that it can be verified by the RS using a trusted key. An access token MAY be protected with a message authentication code. The corresponding symmetric key is then called the Access token Key (AK). An access token MAY be signed with a private key in an asymmetric signature scheme, for example the AS's private key (PrivK_AS). The RS, or other nodes verifying the access token, MUST have access to the relevant key (AK or PubK_AS) at the time of performing the verification.

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 PubK_AS to the intermediary nodes, rather than AK.

4.2 Access Token Protection

Since access tokens are to be consumed by constrained devices, the protection of the access token must be lightweight and compact. This specification RECOMMENDS the use of JSON Web Signatures (JWS) [I-D.ietf-jose-json-web-signature] as a means of signing access tokens. It is furthermore RECOMMENDED to use the JWS Compact Serialization in order to further reduce the size of the protected access token object.

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].

4.3 Access Token Transport

The access token can be transported from the OC to the RS in

different ways.

A. One possibility is to extend the communication security establishment protocol (e.g. using TLS authorization extensions [RFC5878] in the DTLS/TLS handshake).

B. Another possibility is to use the application protocol (e.g. CoAP) and send the tokens as regular requests.

In either case the access token is verified upon reception, and if it is valid (see 4.4), stored for being used in a subsequent resource request. If the access token is not valid (see Section 4.4) 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 SHALL at least support the transport of access tokens in the application protocol. For this to work, there needs to be a well-known location on the RS to which the OC can send the access token. This resource SHALL have the local URI-path './well-known/core/authz/'. Writing to this location SHALL NOT

require Resource Authorization (i.e. no access token is required).

4.4 Access Token Reception

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

- Verify if the token is revoked
- Verify if the token is from a trusted issuer (i.e. the AS known to the RS)
- Verify the signature of the token

In order to support 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.

4.5 Access Token Enforcement

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

- If there is information about expiry, verify if the stored token has expired
- Verify that the stored token is bound to the requesting subject

- Verify that the stored token authorizes the received request (including local conditions)

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 CoAP request is originating from the OC referenced in the received access token. This has implications on the access token and message protection profiles.

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 MUST be signed by the OC in addition to the AS integrity protection. The RS can not authenticate the OC directly, but it can infer from a correctly signed valid and fresh access token that the OC is authorized and has an 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

presented in the original interest request 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. The OC would then have to send a new CoAP request with an observe option expressing interest and a new access token for demonstrating that it is allowed access.

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 has 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

<TBD: IANA considerations text>

8. Acknowledgements

The authors would like to thank Stefanie Gerdes, Mats Naeslund, and John Mattsson 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-core-sec-usecases]
Seitz, L., Gerdes, S., and Selander, G., "Use cases for CoRE security", draft-seitz-core-sec-usecases-00 (work in progress), September 2013.

[I-D.ietf-lwig-terminology]
Bormann, C., Ersue, M., and Keranen, A., "Terminology for Constrained Node Networks", draft-ietf-lwig-terminology-05 (work in progress), July 2013.

[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-17 (work in progress), October 2013.

[I-D.ietf-jose-json-web-signature]
Jones, M., Bradley, J., and Sakimura N., "JSON Web Signature (JWS)", draft-ietf-jose-json-web-signature-17

(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), Octobe 2013.

[I-D.bormann-cbor] Bormann, C., and Hoffman P., "Concise Binary Object Representation (CBOR)", draft-bormann-cbor-09 (work in progress), September 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.

[RFC5878] Brown, M. and R. Housley, "Transport Layer Security (TLS) Authorization Extensions", RFC 5878, May 2010.

[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.

```
01 {
02   "SN": "081d5ff7bb2c2d08",
03   "IS": "6f",
04   "SI": "435143a1b5fc8bb70a3aa9b10f6673a8",
05   "LCO": {
```

```
06      "NB":"09:00:00Z",
07      "NA":"17:00:00Z"
08   },
09   "ACT": "POST",
10   "VAL": "open",
11   "RES": "node346/doorLock"
12 }
```

Token element	Encoding
Sequence number	SN
Issuer	IS

Subject identifier	SI
Local conditions	LCO
Action	ACT
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 action specifies the RESTful action (DELETE, GET, POST, PUT) that this token authorizes, while the resource specifies the URI host and URI path from the CoAP requests.

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 [I-D.bormann-cbor] to reduce the token size.

Appendix B. Changelog

Changes from -00 to -01:

- The draft is significantly shortened, content is moved to separate drafts and much informational content has been removed.
- The limited use case descriptions are greatly expanded and moved into a separate draft [I-D.seitz-core-sec-usecases].
- The key provisioning schemes are generalized to alternate CoAP security modes and described in a separate draft [I-D.seitz-core-security-modes]

- The ACL categories are replaced by the distinction between protocol authorization and resource authorization.
- The Access Manager functionality originally defined in draft-gerdes-core-dcaf-authorize-00 is introduced.
- The communication security profile description is removed. For a detailed DTLS based access control setting see [I-D.draft-gerdes-core-dcaf-authorize].
- 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