ACE Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 21, 2016

O. Bergmann
S. Gerdes
Universitaet Bremen TZI

October 19, 2015

# Using The Delegated CoAP Authentication and Authorization Framework (DCAF) With CBOR Encoded Message Syntax
draft-bergmann-ace-dcaf-cose-00

## Abstract

This specification defines a profile for the Delegated CoAP Authentication and Authorization Framework (DCAF) that facilitates client authentication and authorization in a constrained environment using the CBOR Encoded Message Syntax.

## Status of This Memo

## Copyright Notice

Table of Contents

## 1. Introduction

The Delegated CoAP Authentication and Authorization Framework (DCAF) is designed to be agnostic of the actual mechanism being used to secure the communication between the ACE actors. While the original specification [I-D.gerdes-ace-dcaf-authorize] defines how to use DCAF messaging for establishing a Datagram Transport Layer Security (DTLS) [RFC6347] channel between actors on the constrained level (cf. [I-D.ietf-ace-actors]), this document specifies a binding of DCAF to the CBOR Encoded Message Syntax, COSE [I-D.ietf-cose-msg].

To reduce confusion, we use "DTLS DCAF" to refer to DCAF based on DTLS security, and "COSE DCAF" to refer to DCAF as defined in the present document.

DCAF defines authorized access to a resource hosted on a resource Server (S) based on a security context that is established between the requesting Client (C) and S. In DTLS DCAF, this security context is tied to a DTLS channel which allows for end-to-end integrity protection and confidentiality of communication. In the presence of intermediaries such as, e.g., CoAP proxies, channel security may not be applicable for all configurations. If this is the case, the exchanged information must be protected at the application level to help achieving the principals’ security requirements. The IETF working group CBOR Object Signing and Encryption (COSE) has defined a Concise Binary Object Representation (CBOR) [RFC7049] representation for signed and encrypted objects as well as message authentication codes.

This specification uses this CBOR Encoded Message Syntax [I-D.ietf-cose-msg] to protect the DCAF protocol flow on the application level. The features of this DCAF profile are:

o Authenticated exchange of authorization information.

o Simplified authentication on constrained nodes by handing the more sophisticated authentication work over to less-constrained devices.

o Support of secure constrained device to constrained device communication.

o Authorization policies of the principals of both participating parties are ensured.

o Simplified authorization mechanism for cases where implicit authorization is sufficient.

o Can be made to work just using symmetric encryption on the constrained nodes.

o Enable delivery of piggybacked protected content as discussed in [I-D.gerdes-ace-dcaf-authorize].

## 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].

Readers are expected to be familiar with the terms and concepts defined in [I-D.gerdes-ace-dcaf-authorize].

## 2.  Overview

This specification retains the most important features of DCAF by utilizing the same basic messaging mechanism.  DCAF ensures that protected information is accessible only by authorized entities, i.e. access must be authenticated and the principal that oversees the particular piece of information must permit the requested action. The DCAF specification cryptographically ties this authorization to a DTLS session setup between the communicating Client and Server.  The DTLS key material used for creating this session hence defines the security context between the communicating parties.  By supplementing DCAF with the notion of a context identifier, the same mechanism can be used with application level security as well.

### 2.1.  Sending Authorized Requests

In general, every request that C sends to S must be treated by S within a particular security context with C.  [_1] If S is not able to otherwise identify the security context from the message context, the context identifier must be transferred within the respective message.  An example of a request containing an explicit context identifier is shown in Figure 1 using the CBOR diagnostic notation as defined in Section 6 of [RFC7049] to describe the actual data represented in CBOR.

```
PUT /r
Content-Format: application/cose+cbor
[ h’a10300’,                      # protected { content_type: text/plain }
  { alg: HMAC 256/256,            # unprotected
    kid: h’3233386473613239’      # context identifier: "238dsa29"
  },
  h’48656c6c6f20576f726c6421’,  # payload: "Hello World!\n"
  h’....’,                  # tag: HMAC(options+protected+payload, secret)
  [ [ h’’, {}, h’’ ] ]            # recipients
]
```

Figure 1: Example for CoAP Request with Explicit Context Identifier

Figure 1 shows a PUT request from C to S for resource r containing a payload of type ’application/cose+cbor’ that carries a COSE_Mac structure to integrity-protect the request using the MAC key from a previously established security context with identifier ’238dsa29’. As the security context can be determined from the context identifier, an empty COSE_recipient structure is used.  Note that the integrity protection not only covers the message payload but also the content type and various sensitive CoAP options such as Uri-Path that will be passed to the MAC creation functions as canonicalized external_aad as described in Section 6.

Note1: Where confidentiality is required, a COSE_encryptData structure will be used instead of the COSE_Mac structure.

Note2: COSE_enveloped may be used instead of COSE_encryptData when dynamically generated session keys should be used, e.g. with protected piggybacked content.

Note3: As transferring COSE objects as the CoAP message payload is not always possible (e.g. in GET requests), this specification defines two new CoAP options ’Authorization’ and ’Authorization-Format’ that can be used to convey the authorization information.

To retrieve a resource representation using the request method GET, the authorization information is conveyed in an Authorization attribute as shown in Figure 2.

```
GET /r
Authorization: [ h’’,            # protected (empty)
  { alg: HMAC 256/256,           # unprotected
    kid: h’3233386473613239’     # context identifier: "238dsa29"
  },
  nil,                           # payload (empty)
  h’....’,                       # tag: HMAC(options+protected, secret)
  [ [ h’’, {}, h’’ ] ]           # recipients
]
```

Figure 2: Example for CoAP Request with Authorization Option

The request in Figure 2 uses the default Authorization-Format ’application-cose’ for the contents of the Authorization option which is a COSE_Mac structure. As in Figure 1 the MAC key from a previously established security context with identifier ’238dsa29’ is used and an empty COSE_recipient structure is used. The integrity protection for this request not only covers the message payload but also the content type and various sensitive CoAP options such as Uri-Path that will be passed to the MAC creation functions as external_aad. The external_aad MUST be constructed as CBOR bytes containing a canonicalized CoAP message as specified in Section 6.

## 2.2. Responding to an Authorized Request

A response to an Authorized Request that uses this DCAF profile MUST be protected according to the principals’ security objectives covered by the existing security context between C and S. Usually, this means that a resource representation returned by S in the response is wrapped into a COSE_encryptData or COSE_enveloped structure. A protected response to an authorized GET request is depicted in Figure 3.

Note: For AEAD ciphers, confidentiality and integrity can be achieved in one encryption step. For other cipher suites, it may be more convenient to use a COSE_Mac structure when only message integrity is required.

```
2.05 Content
Content-Format: application/cose+cbor
[ h’a10300’,                # protected { content_type: text/plain }
  { alg: AES-CCM-16-64-128,        # unprotected
    nonce: h’77cd8a8047b7af7113bb074bcc’, # nonce
  },
  h’TBD:encrypted payload w/ tag’, # ciphertext
  # recipients:
  [ [ h’’,                         # protected (absent for AE alg.)
      { alg: A128KW,               # unprotected
        kid: h’3233386473613239’   # context identifier: "238dsa29"
      },
      h’fec31142bc...’             # encrypted session key
  ] ]
]
```

Figure 3: Example for a Protected Response Containing a Resource Representation

## 3. Establishing a Security Context

Section 2.1 illustrates the use of CBOR Encoded Message Syntax for sending Authorized Requests and Responses. Before this communication can take place the security context must be established using the COSE DCAF message types as described in this section. This section describes the basic message flow as outlined in [I-D.gerdes-ace-dcaf-authorize], but using the CBOR Encoded Message Syntax to convey the DCAF information instead of DTLS.

### 3.1. Unauthorized Resource Request Message

The optional Unauthorized Resource Request message is a request for a resource hosted by S for which no proper authorization has been granted so far. S MUST treat any CoAP request as an Unauthorized Resource Request message when any of the following holds:

- The request has been received unprotected.

- The security context for the received request is unknown.

- S has no valid access ticket for the sender of the request regarding the requested action on that resource.

- S has a valid access ticket for the sender of the request, but this does not allow the requested action on the requested resource.

Note: These conditions ensure that S can handle requests autonomously once access has been granted and a security context has been established between C and S.

Unauthorized Resource Request messages MUST be denied with a client error response. In this response, the Server MUST provide proper SAM Information to enable the Client to request an access ticket from S’s SAM as described in Section 3.2. S MAY include a protected piggybacked response with the SAM Information Message in the Unauthorized Resource Request message, as discussed in Section 4.

The response code MUST be 4.01 (Unauthorized) in case the sender of the Unauthorized Resource Request message is not authenticated, or if S has no valid access ticket for C. If S has an access ticket for C but not for the resource that C has requested, S MUST reject the request with a 4.03 (Forbidden). If S has an access ticket for C but it does not cover the action C requested on the resource, S MUST reject the request with a 4.05 (Method Not Allowed).

Note: The use of the response codes 4.03 and 4.05 is intended to prevent infinite loops where a naive Client optimistically tries to access a requested resource with any access token received from the SAM. As malicious clients could pretend to be C to determine C’s privileges, these detailed response codes must be used only when a certain level of security is already available which can be achieved only when the Client is authenticated.

## 3.2. SAM Information Message

The SAM Information Message is sent by S as a response to an Unauthorized Resource Request message (see Section 3.1) to point the sender of the Unauthorized Resource Request message to S’s SAM. The SAM information is a set of attributes containing a URI that specifies the SAM in charge of S.

An optional field A lists the different content formats that are supported by S.

The message MAY also contain a timestamp generated by S.

Figure 4 shows an example for a SAM Information message payload using stylized CBOR diagnostic notation. (Refer to [I-D.gerdes-ace-dcaf-authorize] for a detailed description of the available attributes and their semantics.)

```
4.01 Unauthorized
Content-Format: application/dcaf+cbor
{SAM: "coaps://sam.example.com/authorize", TS: 168537,
A: [ ct_cose_msg ] }
```

Figure 4: SAM Information Payload Example

In this example, the attribute SAM points the receiver of this message to the URI "coaps://sam.example.com/authorize" to request access permissions. The originator of the SAM Information payload (i.e. S) uses a local clock that is loosely synchronized with a time scale common between S and SAM (e.g., wall clock time). Therefore, it has included a time stamp on its own time scale that is used as a nonce for replay attack prevention.

The content format accepted by S is ’application/cose+cbor’ defined in [I-D.ietf-cose-msg] to indicate DCAF over CBOR Encoded Message Syntax as defined in this document.

Editorial note: ct_cose_msg is to be replaced with the numeric value assigned for ’application/cose+cbor’.

The examples in this document are written in CBOR diagnostic notation to improve readability. Figure 5 illustrates the binary encoding of the message payload shown in Figure 4.

```
a2                                   # map(2)
    00                               # unsigned(0) (=SAM)
    78 21                            # text(33)
       636f6170733a2f2f73616d2e6578
       616d706c652e636f6d2f617574686f72
       697a65             # "coaps://sam.example.com/authorize"
    05                               # unsigned(5) (=TS)
    1a 00029259                      # unsigned(168537)
    0a                               # unsigned(10) (=A)
    81                               # array(2)
       19 03e7                       # unsigned(999) (=cose+cbor)
```

Figure 5: SAM Information Payload Example encoded in CBOR

## 3.3. Access Request

To retrieve an access ticket for the resource that C wants to access, C sends an Access Request to its CAM. The Access Request is constructed as follows:

1. The request method is POST.

2. The request URI is set as described below.

3. The message payload contains a COSE_encryptData or COSE_enveloped structure with content-type application/dcaf+cbor that describes the action and resource for which C requests an access ticket.

The request URI identifies a resource at CAM for handling authorization requests from C. The URI SHOULD be announced by CAM in its resource directory as described in [I-D.gerdes-ace-dcaf-authorize].

Note: Where capacity limitations of C do not allow for resource directory lookups, the request URI in Access Requests could be hard-coded during provisioning or set in a specific device configuration profile.

The message payload is constructed from the SAM information that S has returned as described in [I-D.gerdes-ace-dcaf-authorize]. An example Access Request from C to CAM is depicted in Figure 6. (Refer to [I-D.gerdes-ace-dcaf-authorize] for a detailed description of the available attributes and their semantics.)

```
POST /client-authorize
Content-Format: application/cose+cbor
[ h'a1031862',      # protected { content_type: application/dcaf+cbor }
  { alg: AES-CCM-16-64-128         # unprotected
    nonce: h'd6150b90e6f0eb5be42164062c', # nonce
  },
  h'TBD:encrypted payload w/ tag', # encrypted DCAF payload
  # recipients:
  [ [ h'',                         # protected (absent for AE algorithm)
      { alg: A128KW,               # unprotected
        kid: h'383261622e6161733432' # context identifier: "82ab.aas42"
      },
      h'52ff9ed52d...'             # encrypted session key
  ] ]
]
```

Figure 6: Access Request Message Example

The example shows an Access Request message with COSE payload that contains the encrypted and integrity protected DCAF object shown in Figure 7. To integrity-protect the CoAP message header fields the canonicalized CoAP message MUST be included in the external_aad structure. The recipient structure of this message contains a wrapped key that is encrypted with the key material for the common security context of C and CAM that is identified by the kid parameter. If the client cannot create a random session key, it could send a COSE_encryptData structure instead using the direct encryption method. The benefit of wrapping the content encryption key is that CAM can pass the encrypted content on to SAM needing to wrap the content encryption key with the key material used in the common security context with SAM.

```
   {
     SAM: "coaps://sam.example.com/authorize",
     SAI: ["coaps://temp451.example.com/s/tempC", 5],
     TS: 168537
   }
```

Figure 7: Access Request Payload Example

The example shows an Access Request message for the resource "/s/tempC" on the Server "temp451.example.com". Requested operations in attribute SAI are GET and PUT.

The attributes SAM (that denotes the Server Authorization Manager to use) and TS (a nonce generated by S) are taken from the SAM Information message from S.

The response to an Authorization Request is delivered by CAM back to C in a Ticket Transfer message.

## 3.4. Ticket Request Message

CAM processes any Access Request message received from C as defined in [I-D.gerdes-ace-dcaf-authorize]. If CAM decides to send a Ticket Request message to the SAM provided in the Access Request, it has to establish a security context with SAM. Depending on the URI scheme used in the SAM field of the Access Request message payload (the less-constrained devices CAM and SAM do not necessarily use CoAP to communicate with each other), this could be, e.g., a DTLS channel (for "coaps") or a TLS connection (for "https"), or a COSE_enveloped structure using SAM’s public key to encrypt the content encryption key.

## 3.5. Ticket Grant Message

A Ticket Request Message is processed and responded to as specified in [I-D.gerdes-ace-dcaf-authorize]. SAM MUST use the same security context that has been used by CAM to transfer the Ticket Request message, i.e., if the Ticket Request message was received over DTLS, the response MUST be sent over the same DTLS session. This restriction is alleviated slightly when using COSE where the only requirement is that the CoAP response can be mapped to the respective request.

## 3.6. Ticket Transfer Message

A Ticket Transfer message is sent by CAM to deliver the authorization information from SAM in a Ticket Grant message to the requesting client C. Processing of the Ticket Grant message and construction of the Ticket Transfer message is done as specified in [I-D.gerdes-ace-dcaf-authorize]. An example for a Ticket Transfer message in response to the Ticket Access Request described in Section 3.3 is depicted in Figure 8.

```
2.05 Content
Content-Format: application/cose+cbor
[ h’a1031862’,     # protected { content_type: application/dcaf+cbor }
  { alg: AES-CCM-16-64-128         # unprotected
    nonce: h’d259f53783993e757ec9d1d957’, # nonce
    kid: h’383261622e6161733432’   # context identifier: "82ab.aas42"
  },
  h’TBD:encrypted payload w/ tag’, # encrypted DCAF payload
]
```

Figure 8: Example Ticket Transfer Message Encoded as COSE Message

In this example, a COSE_encryptData structure is used to avoid including a recipients structure. The kid parameter referring to the same security context that has been used for the Access Request message is included with the unprotected header of the COSE_encryptData structure. The encrypted DCAF payload contains the required ticket Face and Verifier as defined in [I-D.gerdes-ace-dcaf-authorize]. In this example, the ticket shown in Figure 9 is passed in the payload field of the COSE_encryptData structure shown in Figure 8.

```
{ F: {
        SAI: [ "/s/tempC", 7 ],
        TS: 0("2013-07-10T10:04:12.391"),
        L:  86400,
        G: hmac_sha256
  },
  V: h’f89947160c73601c7a65cb5e08812026
       6d0f0565160e3ff7d3907441cdf44cc9’
  CAI: [ "/s/tempC", 1 ],
  TS: 0("2013-07-10T10:04:12.855"),
  L:  86400
}
```

Figure 9: Example Ticket Transfer Message

## 3.7. Security Association between C and S

The information contained in a Ticket Transfer message (i.e. a ticket a Face and Client Information) can be used by C to establish a security context with S. While [I-D.gerdes-ace-dcaf-authorize] defines how to infer a DTLS pre-shared key, this specification uses the verifier as MAC key in a COSE_MAC structure as described below. This structure comprises the payload of a POST request to the authorization resource hosted by S as described in Section 7.

1. The CoAP request is protected as external_aad as described in Section 6.

2. The protected header contains the parameter content_type with the value ’application/dcaf+cbor’.

3. The unprotected header contains the alg parameter that denotes the MAC algorithm that is used at the content level.

4. The payload field of the COSE_MAC structure contains the ticket Face encoded as canonicalized CBOR structure, and the tag field is constructed using the verifier from the Ticket Transfer message as secret, and the recipients structure is filled with empty values.

The authorization for uploading authorization tickets is tied to a key that is associated to the particular ticket Face and MUST be generated by the authorized SAM. When receiving a POST request to the auth-info resource, S generates its own version of the verifier using the information contained in Face.

The distributed key derivation method is defined as follows:

- SAM and S both generate the verifier using the information included in Face. They use an HMAC algorithm on Face with a shared key K(SAM,S). The result serves as the content encryption key. How SAM and S exchange K(SAM,S) is not in the scope of this document. They MAY use their preshared key as K(SAM,S).

- SAM MUST include a representation of the session key in the Verifier.

- As SAM and C do not have a shared secret, the Verifier MUST be transmitted to C using protected channels.

- SAM MUST NOT include a representation of the Verifier in Face.

- SAM MUST NOT encrypt Face.

Once S has validated the contents of the POST request using the locally generated verifier, it creates a new resource that represents this authorization and returns the Location-Path of this new resource. This path then can be used by C to update the authorization information and MUST be used by C in the kid parameter to identify this security context as described in Section 2.1.

An example for the POST request and corresponding 2.01 response is given in Figure 10. The Location-Path returned by S is subsequently used by C as identifier for the security context tied to this authorization.

```
C --> S
POST /authorize
Content-Format: application/cose+cbor
[ h'a1031862',     # protected { content_type: application/dcaf+cbor }
  { alg: HMAC 256/256 },        # unprotected
  h'{ SAI: [ "/s/tempC" ... }', # DCAF payload wrapped in CBOR binary
  h'....',              # tag: HMAC(options+protected+payload, secret)
  [ [ h'', {}, h'' ] ]          # recipients
]

S --> C
2.01 Created
Content-Format: application/cose+cbor
Location-Path: 238dsa29
Authorization: [ h'a1031862',   # protected
  { alg: HMAC 256/256 },        # unprotected
  h'',                          # empty payload
  h'....',                      # tag: HMAC(options+protected, secret)
  [ [ h'', {}, h'' ] ]          # recipients
]
```

Figure 10: Example POST to S's auth-info Resource and Response

## 4. Piggybacked Protected Content

Piggybacked protected content was introduced in [I-D.gerdes-ace-dcaf-authorize] as a possibility to deliver an encrypted resource representation without having to maintain authorization information for the respective resource. Once a requesting client has received the piggybacked content, it needs to request authorization for accessing the protected data. To do so, it constructs an Access Request as defined in Section 3.3. If access to the protected data is granted, the requesting client will be provided with cryptographic material to verify the integrity and authenticity of the piggybacked content and decrypt the protected data in case it is encrypted.

## 5. CoAP Options Authorization and Authorization-Format

The options Authorization and Authorization-Format have the properties shown in Table 1.

| No. | C | U | N | R | Name | Format | Length | Default |
|---|---|---|---|---|---|---|---|---|
| 64 | | | | | Authorization | opaque | 1-1034 | (none) |
| 65 | x | | | | Authorization-Format | uint | 0-2 | application/cose+cbor |

Table 1: The Options Authorization and Authorization-Format

## 6. Canonicalization of the CoAP Message Header

This section describes the canonicalization of parts from the CoAP message for integrity protection. As intermediaries such as caching proxies may change certain fields in a CoAP message, only those fields are considered that must not be changed by intermediaries. The canonicalized CoAP message then serves as external_aad to the COSE MAC_structure and Enc_structure as used in this specification. The canonicalized CoAP message is constructed as follows:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Ver| 0 |   0   |      Code     | Options to protect (if any) ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 11: Canonicalized CoAP Message Header

As shown in Figure 11, only the version bits and the message code from the CoAP base header are relevant for integrity protection. [_3] From the list of options that a message might have, only the following options are to be included with the canonicalized message. [_4]

o  If-Match

o  Uri-Host

o  ETag

- If-None-Match
- Observe
- Uri-Port
- Location-Path
- Uri-Path
- Uri-Query
- Accept
- Location-Query
- Proxy-Uri
- Proxy-Scheme
- Size1

Note: The Content-Format must be contained in the protected header of the MAC_structure or Enc_structure and hence is not required here.

An application that requires integrity protection of new options that are not listed here must add a critical-options header field to the MAC_structure or Enc_structure containing a CBOR array with the additional options to protect in ascending numerical order.

Figure 12 shows an example for a POST request to upload SenML [I-D.jennings-core-senml] sensor readings to a remote server. The protected header in the COSE_Mac structure contains a ’required options’ entry that lists the custom option X-Something, hence the external_aad would contain a canonicalized message header that consists of the CoAP version number, the method POST, Uri-Path ’measurements’, Uri-Path ’current’, and X-Something 1234 as delta-encoded options in ascending order as specified in Section 3.1 of [RFC7252].

```
   POST /measurements/current
   Content-Format: application/senml+cbor
   X-Something: 1234
   Authorization: [
     # protected { content_type: application/senml+cbor,
     #             "required options": [ X-Something ] }
     h’a203766170706c69636174696f6e2f73656e6d6...’,
     { alg: HMAC 256/256,          # unprotected
       kid: h’3233386473613239’    # context identifier: "238dsa29"
     },
     h’a2231a4eaecc5d....’,        # payload: "{ -4: 1320078..."
     h’....’,            # tag: HMAC(options+protected+payload, secret)
     [ [ nil, {}, h’’ ] ]          # recipients
   ]

   { -4: 1320078429,
     -2: [{0: "temperature", 2: 272, 1: "Cel"},
          {0: "humidity", 2: 80, 1: "%RH"}]
   }
```

Figure 12: Example Message with Protected Custom Option

## 7.  The "auth-info" Link Relation

This section defines a resource type "auth-info" that can be used by clients to establish a new security context with S using the authorization information retrieved from SAM.  When used with the parameter rt in a web link, "auth-info" indicates that the corresponding target URI can be used in a POST message to upload the authorization information contained in the request payload.

The following example shows the web link used by S in this document to accept authorization information created by SAM.

```
   <authorize>;rt="auth-info";ct=TBD1,ct_cose_msg
              ;title="Upload Authorization Information"
```

The resource directory that hosts the resource descriptions of S could list the following description.  In this example, the URI "ep/node138/a/switch2941" is relative to the resource context "coaps://sam.example.com/", i.e. the Server Authorization Manager SAM.

```
   <ep/node138/a/switch2941>;rt="auth-info";ct=TBD1,ct_cose_msg
                            ;ep="node138"
                            ;title="Upload Authorization Information"
                            ;anchor="coaps://s.example.com/"
```

## 8. Security Considerations

The SAM Information message cannot be protected as no security context between S and C is present at the time the message is sent. An attacker thus can inject a SAM Information message listing a different SAM URI to trick C into disclosing the intended action. Where this is an issue, C could retrieve the SAM URI from a resource directory as described in [I-D.gerdes-ace-dcaf-authorize].

## 9. IANA Considerations

The following registrations are done following the procedure specified in [RFC6838].

Note to RFC Editor: Please replace all occurrences of "[RFC-XXXX]" with the RFC number of this specification.

### 9.1. CoAP Option Registration

IANA is requested to add the following entries to the CoAP Option Numbers registry:

| Number | Name | Reference |
|---|---|---|
| 64 | Authorization | [RFC-XXXX] |
| 65 | Authorization-Format | [RFC-XXXX] |

## 10. Acknowledgements

The authors would like to thank Carsten Bormann for his valuable input and feedback.

## 11. References

### 11.1. Normative References

[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-03 (work in progress), September 2015.

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

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

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <http://www.rfc-editor.org/info/rfc7252>.

## 11.2. Informative References

[I-D.ietf-ace-actors] Gerdes, S., Seitz, L., Selander, G., and C. Bormann, "An architecture for authorization in constrained environments", draft-ietf-ace-actors-02 (work in progress), October 2015.

[I-D.ietf-cose-msg] Schaad, J., "CBOR Encoded Message Syntax", draft-ietf-cose-msg-06 (work in progress), October 2015.

[I-D.jennings-core-senml] Jennings, C., Shelby, Z., Arkko, J., and A. Keranen, "Media Types for Sensor Markup Language (SENML)", draft-jennings-core-senml-02 (work in progress), October 2015.

[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <http://www.rfc-editor.org/info/rfc6347>.

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

## Editorial Comments

[_1] Editor’s note: As a consequence, if no such security context is found, the request will be rejected as Unauthorized Request.

[_3] Editor’s note: message type, token and id can change on the way.

[_4] TBD

## Authors' Addresses

Olaf Bergmann
Universitaet Bremen TZI
Postfach 330440
Bremen  D-28359
Germany

Phone: +49-421-218-63904
Email: bergmann@tzi.org

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

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