

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: August 31, 2019

C. Bormann
Universitaet Bremen TZI
February 27, 2019

Additional Units for SenML
draft-bormann-senml-more-units-00

Abstract

The Sensor Measurement Lists (SenML) media type supports the indication of units for a quantity represented. This short document registers a number of additional unit names in the IANA registry for Units in SenML.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on August 31, 2019.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

| | |
|--------------------------------------|---|
| 1. Introduction | 2 |
| 2. New Units | 2 |
| 3. Rationale | 3 |
| 4. Security Considerations | 3 |
| 5. IANA Considerations | 4 |
| Acknowledgements | 4 |
| 7. Normative References | 4 |
| Author's Address | 4 |

1. Introduction

The Sensor Measurement Lists (SenML, [RFC8428]) media type supports the indication of a unit, using the SenML field "u", for the quantity given as a data value in a SenML record. For this purpose, SenML defines an IANA registry of defined Unit names and their meanings.

This short document registers a number of additional units in the IANA registry for Units in SenML that appear to be necessary for further adopting SenML in other Standards Development Organizations (SDOs).

2. New Units

IANA is requested to assign new units in the "SenML Units" subregistry of the SenML registry [IANA.senml] (as defined in [RFC8428]):

| Symbol | Description | Type | Reference |
|--------|---------------------------------------|-------|-----------|
| B | Byte (information content) | float | RFCthis |
| VA | volt-ampere (Apparent Power) | float | RFCthis |
| var | volt-ampere reactive (Reactive Power) | float | RFCthis |
| J/m | joule per meter (Energy per distance) | float | RFCthis |

Table 1: New units registered for SenML

3. Rationale

SenML [RFC8428] takes the position that unscaled SI units should always be used. However, SenML makes one exception: The degree Celsius (as Cel) is allowed as an alternative to the K (Kelvin).

This document takes the position that the same should apply to a small number of alternative units in wide use:

- o The Byte. [IEC-80000-13] defines both the bit (item 13-9.b) and the byte (item 13-9.c, also called octet) as alternative names for the coherent unit one for the purpose of giving storage capacity and related quantities. While the name octet is associated with the symbol o, this is in wide use only in French-speaking countries. Globally more wide-spread is the symbol B for byte, even though B is already taken in SI for bel. [RFC8428] therefore registers dB as the SenML unit for logarithmic relative power, leaving B free for the usage proposed here. While this is potentially confusing, the situation is widely understood in engineering circles and is unlikely to cause actual problems.
- o The Volt-Ampere. [IEC-80000-6] item 6-57.a defines the VA (volt ampere) as a unit for apparent power; items 6-59.a, 6-60.a and 6-61.a also use the unit for complex, reactive, and non-active power.
- o The Volt-Ampere-reactive. [IEC-80000-6] item 6-60.b defines the var (volt ampere reactive) as an alternative (and fully equivalent) unit to VA specifically for reactive power (with the primary unit VA). It is not presently known to this author how the upcoming revision of IEC 80000-6 will update this, but it has become clear since that there is strong interest in using this unit specifically for the imaginary content of complex power, reactive power [IEEE-1459].

The Joule per meter is not a traditional electromagnetic unit. It and its scaled derivatives (in particular Wh/km) are used to describe the energy expended for achieving motion over a given distance, e.g. as an equivalent for electrical cars of the inverse of "mileage".

4. Security Considerations

The security considerations of [RFC8428] apply. The introduction of new measurement units poses no additional security considerations except from a possible potential for additional confusion about the proper unit to use.

5. IANA Considerations

See Section 2.

Acknowledgements

Ari Keranen pointed out the need for additional units in SenML.

7. Normative References

[IANA.senml]

IANA, "Sensor Measurement Lists (SenML)",
<<http://www.iana.org/assignments/senml>>.

[IEC-80000-13]

"Quantities and units - Part 13: Information science and technology", IEC 80000-13, Edition 1.0, March 2008.

[IEC-80000-6]

"Quantities and units - Part 6: Electromagnetism",
IEC 80000-6, Edition 1.0, March 2008.

[IEEE-1459]

"IEEE Standard Definitions for the Measurement of Electric Power Quantities Under Sinusoidal, Nonsinusoidal, Balanced, or Unbalanced Conditions", IEEE Std 1459-2010, March 2010.

[RFC8428]

Jennings, C., Shelby, Z., Arkko, J., Keranen, A., and C. Bormann, "Sensor Measurement Lists (SenML)", RFC 8428, DOI 10.17487/RFC8428, August 2018, <<https://www.rfc-editor.org/info/rfc8428>>.

Author's Address

Carsten Bormann
Universitaet Bremen TZI
Postfach 330440
Bremen D-28359
Germany

Phone: +49-421-218-63921
Email: cabo@tzi.org

CoRE Working Group
Internet-Draft
Obsoletes: 7390 (if approved)
Updates: 7252, 7641, 7959 (if approved)
Intended status: Standards Track
Expires: January 9, 2020

E. Dijk
IoTconsultancy.nl
C. Wang
InterDigital
M. Tiloca
RISE AB
July 08, 2019

Group Communication for the Constrained Application Protocol (CoAP)
draft-dijk-core-groupcomm-bis-01

Abstract

This document specifies the use of the Constrained Application Protocol (CoAP) for group communication, using UDP/IP multicast as the underlying data transport. The target application area is any group communication use cases in resource-constrained networks. Both unsecured and secured CoAP group communication are specified. Security is achieved by use of the Group Object Security for Constrained RESTful Environments (Group OSCORE) protocol. Aspects of operation of using multicast CoAP in combination with CoAP block-wise transfers and CoAP observe are also specified.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on January 9, 2020.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents

(<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

| | |
|---|----|
| 1. Introduction | 3 |
| 1.1. Scope | 4 |
| 1.2. Terminology | 4 |
| 2. General Group Communication Operation | 4 |
| 2.1. Group Configuration | 5 |
| 2.1.1. Group Definition | 5 |
| 2.1.2. Group Naming | 5 |
| 2.1.3. Group Creation and Membership | 6 |
| 2.1.4. Group Maintenance | 6 |
| 2.2. CoAP Usage | 7 |
| 2.2.1. Request/Response Model | 7 |
| 2.2.2. Port and URI Path Selection | 8 |
| 2.2.3. Proxy Operation | 9 |
| 2.2.4. Congestion Control | 11 |
| 2.2.5. Observing Resources | 12 |
| 2.2.6. Block-Wise Transfer | 13 |
| 2.3. Transport | 15 |
| 2.3.1. UDP/IPv6 Multicast Transport | 15 |
| 2.3.2. UDP/IPv4 Multicast Transport | 15 |
| 2.3.3. 6LoWPAN | 15 |
| 2.4. Interworking with Other Protocols | 15 |
| 2.4.1. MLD/MLDv2/IGMP | 15 |
| 2.4.2. RPL | 15 |
| 2.4.3. MPL | 15 |
| 3. Unsecured Group Communication | 15 |
| 4. Secured Group Communication using Group OSCORE | 16 |
| 4.1. Secure Group Maintenance | 17 |
| 5. Security Considerations | 18 |
| 5.1. CoAP NoSec Mode | 18 |
| 5.2. Group OSCORE | 18 |
| 5.2.1. Group Key Management | 19 |
| 5.2.2. Source Authentication | 19 |
| 5.2.3. Counteraction of Attacks | 20 |
| 5.3. 6LoWPAN | 20 |
| 5.4. Wi-Fi | 20 |
| 5.5. Monitoring | 20 |
| 6. IANA Considerations | 20 |
| 7. References | 20 |

| | |
|--|----|
| 7.1. Normative References | 21 |
| 7.2. Informative References | 22 |
| Appendix A. Use Cases | 23 |
| A.1. Discovery | 24 |
| A.1.1. Distributed Device Discovery | 24 |
| A.1.2. Distributed Service Discovery | 24 |
| A.1.3. Directory Discovery | 25 |
| A.2. Operational Phase | 25 |
| A.2.1. Actuator Group Control | 25 |
| A.2.2. Device Group Status Request | 25 |
| A.2.3. Network-wide Query | 26 |
| A.2.4. Network-wide / Group Notification | 26 |
| A.3. Software Update | 26 |
| Acknowledgments | 27 |
| Authors' Addresses | 27 |

1. Introduction

This document specifies group communication using the Constrained Application Protocol (CoAP) [RFC7252] together with UDP/IP multicast. CoAP is a RESTful communication protocol that is used in resource-constrained nodes, and in resource-constrained networks where packet sizes should be small. This area of use is summarized as Constrained RESTful Environments (CoRE).

One-to-many group communication can be achieved in CoAP, by a client using UDP/IP multicast data transport to send multicast CoAP request messages. In response, each server in the addressed group sends a response message back to the client over UDP/IP unicast. Notable CoAP implementations supporting group communication include the framework "Eclipse Californium" 2.0.x [Californium] from the Eclipse Foundation and the "Implementation of CoAP Server & Client in Go" [Go-OCF] from the Open Connectivity Foundation (OCF).

Both unsecured and secured CoAP group communication over UDP/IP multicast are specified in this document. Security is achieved by using Group Object Security for Constrained RESTful Environments (Group OSCORE) [I-D.ietf-core-oscore-groupcomm], which in turn builds on Object Security for Constrained Restful Environments (OSCORE) [I-D.ietf-core-object-security]. This method provides end-to-end application-layer security protection of CoAP messages, by using CBOR Object Signing and Encryption (COSE) [RFC8152] [RFC7049].

All sections in the body of this document are normative, while appendices are informative. For additional background about use cases for CoAP group communication in resource-constrained devices and networks, see Appendix A.

1.1. Scope

For group communication, only solutions that use CoAP over UDP/multicast (both IPv6 and IPv4) are in scope. There are alternative methods to achieve group communication using CoAP, for example Publish-Subscribe [I-D.ietf-core-coap-pubsub] which uses a central broker server that CoAP clients access via unicast communication. The alternative methods may be usable for the same or similar use cases as are targeted in this document.

All guidelines in [RFC7390] are imported by this document which replaces [RFC7390] in this respect. This document furthermore adds the security solution using Group OSCORE as the default group communication security solution for CoAP, an updated request/response matching rule for multicast CoAP which updates [RFC7252], multicast use of CoAP Observe which updates [RFC7641] and extension of multicast use of CoAP block-wise transfers which updates [RFC7959].

Security solutions for group communication and configuration other than Group OSCORE are not in scope. General principles for secure group configuration are in scope. The experimental group configuration protocol in Section 2.6.2 of [RFC7390] is not in the scope of this document; thus, it remains an experimental protocol. Since application protocols defined on top of CoAP often define their own specific method of group configuration, the experimental protocol of [RFC7390] has not been subject to enough experimentation to warrant a change of this status.

1.2. 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 BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

This specification requires readers to be familiar with CoAP [RFC7252] terminology.

2. General Group Communication Operation

The general operation of group communication, applicable for both unsecured and secured operation, is specified in this section by going through the stack from top to bottom. First, group configuration (e.g. group creation and maintenance which are usually done by an application, user or commissioning entity) is considered in Section 2.1. Then the use of CoAP for group communication including support for protocol extensions (block-wise, Observe, PATCH

method) follows in Section 2.2. How CoAP group messages are carried over various transport layers is the subject of Section 2.3. Finally, Section 2.4 covers the interworking of CoAP with other protocols at the layers below it.

2.1. Group Configuration

2.1.1. Group Definition

A CoAP group is defined as a set of CoAP endpoints, where each endpoint is configured to receive CoAP multicast requests that are sent to the group's associated IP multicast address and UDP port. An endpoint may be a member of multiple CoAP groups. Group membership(s) of an endpoint may dynamically change over time. A device sending a CoAP request to a group is not necessarily itself a member of this group: it is only a member if it also has a CoAP server endpoint listening to requests for this CoAP group. For secure group communication, a receiver also requires the security context to decrypt and/or verify group messages in order to be a group member.

A CoAP Group URI has the scheme 'coap' and includes in the authority part either an IP multicast address or a group hostname (e.g., Group Fully Qualified Domain Name (FQDN)) that can be resolved to an IP multicast address. A Group URI also contains an optional UDP port number in the authority part. Group URIs follow the regular CoAP URI syntax (Section 6 of [RFC7252]).

Besides CoAP groups, that have relevance at the level of networked devices, there can also be application groups defined. An application group has relevance at the application level - for example an application group could denote all lights in an office room or all sensors in a hallway. There can be a one-to-one or a one-to-many relation between CoAP groups and application groups.

2.1.2. Group Naming

For clients, it is RECOMMENDED to use by default an IP multicast address literal in a configured Group URI, instead of a hostname. This is because DNS infrastructure may not be deployed in many constrained networks. In case a group hostname is used in the Group URI, it can be uniquely mapped to an IP multicast address via DNS resolution - if DNS client functionality is available in the clients and the DNS service is supported in the network. Some examples of hierarchical group FQDN naming (and scoping) for a building control application are shown in Section 2.2 of [RFC7390].

Application groups can be named in many ways, e.g. numbers, IDs, strings or URIs. An application group identifier, if used, is typically included in the path component or query component of a Group URI. Appendix A of [I-D.ietf-core-resource-directory] shows registration of application groups into a Resource Directory, along with the CoAP group it maps to.

2.1.3. Group Creation and Membership

Group membership may be (factory-)preconfigured in devices or dynamically configured in a system on-site.

To create a CoAP group, a configuring entity defines an IP multicast address (or hostname) for the group and optionally a UDP port number in case it differs from the default CoAP port 5683. Then, it configures one or more devices as listeners to that IP multicast address, with a CoAP server listening on the specific port. These devices are the group members. The configuring entity can be a local application with preconfiguration, a user, a cloud service, or a local commissioning tool for example. Also, the devices sending requests to the group in the role of CoAP clients need to be configured with the same information, even though they are not necessarily group members. One way to configure a client is to supply it with a CoAP Group URI.

For unsecure group communication, any CoAP endpoint may become a group member at any time: there is no (central) configuring entity that needs to provide the security material for the group. This means that group creation and membership cannot be tightly controlled.

The IETF does not define a mandatory, standardized protocol to accomplish these steps. For secure group communication, the part of the process that involves secure distribution of group keys MAY use standardized communication with a Group Manager as defined in Section 4. [RFC7390] defines an experimental protocol for configuration of group membership for unsecured group communication, based on JSON-formatted configuration resources. This protocol remains experimental.

2.1.4. Group Maintenance

Maintenance of a group includes necessary operations to cope with changes in a system, such as: adding group members, removing group members, reconfiguration of UDP port and/or IP multicast address, reconfiguration of the Group URI, splitting of groups, or merging of groups.

For unsecured group communication (see Section 3), addition/removal of group members is simply done by configuring these devices to start/stop listening to the group IP multicast address, and to start/stop the CoAP server listening to the group IP multicast address and port.

For secured group communication (see Section 4), the protocol Group OSCORE [I-D.ietf-core-oscore-groupcomm] is mandatory to implement. When using Group OSCORE, CoAP endpoints participating to group communication are also members of a corresponding OSCORE group, and thus share a common set of cryptographic material. Additional maintenance operations are discussed in Section 4.1.

2.2. CoAP Usage

2.2.1. Request/Response Model

All CoAP requests that are sent via IP multicast MUST be Non-confirmable (Section 8.1 of [RFC7252]). The Message ID in an IP multicast CoAP message is used for optional message deduplication as detailed in Section 4.5 of [RFC7252].

A server MAY send back a unicast response to the CoAP group communication request – whether it does this or not is selected by the server application. The unicast responses received by the CoAP client may be a mixture of success (e.g., 2.05 Content) and failure (e.g., 4.04 Not Found) codes depending on the individual server processing results.

TBD: the CoAP Option for No Server Response [RFC7967] can be used by the client to influence response suppression on the server side. Possibly we can include this information here; it specifically targets use for multicast use cases also.

The client can distinguish the origin of multiple server responses by the source IP address of the UDP message containing the CoAP response or any other available unique identifier (e.g., contained in the CoAP response payload). In case a client has sent multiple group requests with concurrent CoAP transactions ongoing, the responses are matched to a request using the Token value. The source endpoint of the response is not matched to the destination endpoint of the request, since for a multicast request these will never match. This is also specified in Section 8.2 of [RFC7252]. As an update to the [RFC7252] matching rule, a client MAY, in addition to the Token, match the source port of the request to the destination port of the response, since these will match in any correctly formatted CoAP response. This can help a client to more easily meet the below constraint on Token reuse or to more efficiently filter received responses.

For multicast CoAP requests, there are additional constraints on the reuse of Token values, compared to the unicast case. In the unicast case, if the Observe Option [RFC7641] is not used in a request, receiving a response effectively frees up its Token value for reuse since no more responses will follow. However, for multicast CoAP, the number of responses is not bounded a priori. Therefore, the reception of a response cannot be used as a trigger to "free up" a Token value for reuse. Reusing a Token value too early could lead to incorrect response/request matching in the client and would be a protocol error. Therefore, the time between reuse of Token values used in multicast requests MUST be greater than:

`NON_LIFETIME + MAX_LATENCY + MAX_SERVER_RESPONSE_DELAY`

where `NON_LIFETIME` and `MAX_LATENCY` are defined in Section 4.8 of [RFC7252]. This specification defines `MAX_SERVER_RESPONSE_DELAY` as in [RFC7390], that is: the expected maximum response delay over all servers that the client can send a multicast request to. This delay includes the maximum Leisure time period as defined in Section 8.2 of [RFC7252]. However, CoAP does not define a time limit for the server response delay. Using the default CoAP parameters, the Token reuse time MUST be greater than 250 seconds plus `MAX_SERVER_RESPONSE_DELAY`. A preferred solution to meet this requirement is to generate a new unique Token for every multicast request, such that a Token value is never reused. If a client has to reuse Token values for some reason, and also `MAX_SERVER_RESPONSE_DELAY` is unknown, then using `MAX_SERVER_RESPONSE_DELAY = 250 seconds` is a reasonable guideline. The time between Token reuses is in that case set to a value greater than 500 seconds.

2.2.2. Port and URI Path Selection

A CoAP server that is a member of a group listens for CoAP messages on the group's IP multicast address, usually on the CoAP default UDP port 5683, or another non-default UDP port if configured. Regardless of the method for selecting the port number, the same port number MUST be used across all CoAP servers that are members of a group and across all CoAP clients performing the group requests to that group. The URI Path used in the request is preferably a path that is known to be supported across all group members. However there are valid use cases where a request is known to be successful for a subset of the group and those group members for which the request is unsuccessful either ignore the multicast request or respond with an error status code.

Using different ports with the same IP multicast address is an efficient way to create multiple CoAP groups in constrained devices, in case the device's stack only supports a limited number of IP

multicast group memberships. However, it must be taken into account that this incurs additional processing overhead on each CoAP server participating in at least one of these groups: messages to groups that are not of interest to the node are only discarded at the higher transport (UDP) layer instead of directly at the network (IP) layer.

Port 5684 is reserved for DTLS-secured CoAP and MUST NOT be used for any CoAP group communication.

For a CoAP server node that supports resource discovery as defined in Section 2.4 of [RFC7252], the default port 5683 MUST be supported (see Section 7.1 of [RFC7252]) for the "All CoAP Nodes" multicast group. This implies that the receiving server when correctly operating does not send a "ICMP Destination Unreachable - Port Unreachable" in response to a resource discovery request.

2.2.3. Proxy Operation

CoAP (Section 5.7.2 of [RFC7252]) allows a client to request a forward-proxy to process its CoAP request. For this purpose, the client specifies either the request group URI as a string in the Proxy-URI option or alternatively it uses the Proxy-Scheme option with the group URI constructed from the usual Uri-* options. This approach works well for unicast requests. However, there are certain issues and limitations of processing the (unicast) responses to a CoAP group communication request made in this manner through a proxy.

A proxy may buffer all the individual (unicast) responses to a CoAP group communication request and then send back only a single (aggregated) response to the client. However, there are some issues with this aggregation approach:

- o Aggregation of (unicast) responses to a CoAP group communication request in a proxy is difficult. This is because the proxy does not know how many members there are in the group or how many group members will actually respond. Also, the proxy does not know how long to wait before deciding to send back the aggregated response to the client.
- o There is no default format defined in CoAP for aggregation of multiple responses into a single response. Such a format could be defined based on the multipart content-format [I-D.ietf-core-multipart-ct] but is not standardized yet currently.

Alternatively, if a proxy does not aggregate responses and follows the CoAP Proxy specification (Section 5.7.2 of [RFC7252]), the proxy would forward all the individual (unicast) responses to a CoAP group

communication request to the client. There are also issues with this approach:

- o The client may be confused as it may not have known that the Proxy-URI contained a group URI target. That is, the client that sent a unicast CoAP request to the proxy may be expecting only one (unicast) response. Instead, it receives multiple (unicast) responses, potentially leading to fault conditions in the application.
- o Each individual CoAP response will appear to originate (based on its IP source address) from the CoAP Proxy, and not from the server that produced the response. This makes it impossible for the client to identify the server that produced each response, unless the server identity is contained as a part of the response payload.

Due to the above issues, a CoAP Proxy SHOULD NOT support processing an IP multicast CoAP request but rather return a 501 (Not Implemented) response in such case. The exception case here (i.e., to support it) is when all the following conditions are met:

- o The CoAP Proxy MUST be explicitly configured (whitelist) to allow proxied IP multicast requests by specific client(s).
- o The proxy SHOULD return individual (unicast) CoAP responses to the client (i.e., not aggregated). If a (future) standardized aggregation format is being used, then aggregated responses may be sent.
- o It MUST be known to the person/entity doing the configuration of the proxy, or otherwise verified in some way, that the client configured in the whitelist supports receiving multiple responses to a proxied unicast CoAP request.

The operation of HTTP-to-CoAP proxies for multicast CoAP requests is specified in Section 8.4 and 10.1 of [RFC8075]. In this case, the "application/http" media type can be used to let the proxy return multiple CoAP responses - each translated to a HTTP response - back to the HTTP client. Of course the HTTP client in this case needs to be aware that it is receiving this format and needs to be able to decode from it the responses of multiple servers. The above restrictions listed for CoAP Proxies still apply to HTTP-to-CoAP proxies: specifically, the IP address of the sender of each CoAP response cannot be determined from the application/http response.

2.2.4. Congestion Control

CoAP group communication requests may result in a multitude of responses from different nodes, potentially causing congestion. Therefore, both the sending of IP multicast requests and the sending of the unicast CoAP responses to these multicast requests should be conservatively controlled.

CoAP [RFC7252] reduces IP multicast-specific congestion risks through the following measures:

- o A server may choose not to respond to an IP multicast request if there's nothing useful to respond to (e.g., error or empty response); see Section 8.2 of [RFC7252].
- o A server should limit the support for IP multicast requests to specific resources where multicast operation is required (Section 11.3 of [RFC7252]).
- o An IP multicast request MUST be Non-confirmable (Section 8.1 of [RFC7252]).
- o A response to an IP multicast request SHOULD be Non-confirmable (Section 5.2.3 of [RFC7252]).
- o A server does not respond immediately to an IP multicast request and should first wait for a time that is randomly picked within a predetermined time interval called the Leisure (Section 8.2 of [RFC7252]).

Additional guidelines to reduce congestion risks defined in this document are as follows:

- o A server in an LLN should only support group communication GET for resources that are small. For example, the payload of the response is limited to approximately 5% of the IP Maximum Transmit Unit (MTU) size, so it fits into a single link-layer frame in case IPv6 over Low-Power Wireless Personal Area Networks (6LoWPAN) (see Section 4 of [RFC4944]) is used.
- o A server SHOULD minimize the payload length in response to a multicast GET on `"/.well-known/core"` by using hierarchy in arranging link descriptions for the response. An example of this is given in Section 5 of [RFC6690].
- o A server MAY minimize the payload length of a response to a multicast GET (e.g., on `"/.well-known/core"`) using CoAP block-wise transfers [RFC7959] in case the payload is long, returning only a

first block of the CoRE Link Format description. For this reason, a CoAP client sending an IP multicast CoAP request to `"/.well-known/core"` SHOULD support block-wise transfers.

- o A client SHOULD use CoAP group communication with the smallest possible IP multicast scope that fulfills the application needs. As an example, site-local scope is always preferred over global scope IP multicast if this fulfills the application needs. Similarly, realm-local scope is always preferred over site-local scope if this fulfills the application needs.

2.2.5. Observing Resources

The CoAP Observe Option [RFC7641] is a protocol extension of CoAP, that allows a CoAP client to retrieve a representation of a resource and automatically keep this representation up-to-date over a longer period of time. The client gets notified when the representation has changed. [RFC7641] does not mention whether the Observe Option can be combined with CoAP multicast.

This section updates [RFC7641] with the use of the Observe Option in a CoAP multicast GET request. This is a useful way to start observing a particular resource on all members of a (multicast) group at the same time. Group members that do not have this resource or do not allow the GET method on it will either respond with an error status - 4.04 Not Found or 4.05 Method Not Allowed respectively - or will silently ignore the request, depending on server-specific configuration.

A client that sends a multicast GET request with the Observe Option MAY repeat this request using the same Token value and same Observe Option value, in order to ensure that enough (or all) group members have been reached with the request. This is useful in case a number of group members did not respond to the initial request. This client MAY also use the same Message ID to avoid that group members that had already received the initial request would respond again. If the client uses a different, fresh Message ID then all group members that receive this new message will respond again.

A client that sends a multicast GET request with the Observe Option MAY send a new unicast request with the same Token value and same Observe Option value, in order to ensure that the specific server receives the request. This is useful in case a specific group member, that was expected to respond to the initial group request, did not respond to the initial request. The client in this case always uses a Message ID that differs from the initial message.

In the above client behaviors, the Token value is kept identical to the initial request to avoid that the client is included in more than one entry in the list of observers (Section 4.1 of [RFC7641]). While a Token value is in use for observing a group, this Token value cannot be reused by the same client endpoint for other purposes. Another endpoint on the client i.e. using a different UDP source port MAY re-use the Token value but only if the client implements the optional updated matching rule of Section 2.2.1.

Before repeating a request as specified above, the client SHOULD wait for at least the expected round-trip time plus the Leisure time period defined in Section 8.2 of [RFC7252] to allow the server the time to respond.

For observing a group of servers through a CoAP-to-CoAP proxy or HTTP-CoAP proxy, the limitations stated in Section 2.2.3 apply.

2.2.6. Block-Wise Transfer

Section 2.8 of [RFC7959] specifies how a client can use block-wise transfer (Block2 Option) in a multicast GET request to limit the size of the initial response of each server. The client has to use unicast for any further requests to retrieve more blocks of the resource. Also, a server (group member) that needs to respond to a multicast request with a particularly large resource can use block-wise transfer (Block2 Option) at its own initiative to limit the size of the initial response. Again, a client would have to use unicast for any further requests to retrieve more blocks of the resource.

TBD: below solution for multicast block-wise Block1 is used e.g. for efficiently distributing large data/software updates using multicast. It is non-trivial to do right and needs testing. For this reason, we may decide to move this into a separate draft.

This section specifies in addition the use of CoAP block-wise transfers for multicast PUT/POST/PATCH/iPATCH requests in order to efficiently distribute a large request payload as multiple blocks to all members of a CoAP group. The Block1 Option [RFC7959] is then used by the client in each block-wise request and a server uses the Block1 Option in its response to confirm reception of a block and optionally to indicate in the first block-wise response that it prefers a smaller block size.

Prior to starting a block-wise multicast request, the client SHOULD already store a list of those members of the CoAP group that need to properly receive the request payload. These members are expected to support block-wise CoAP and are also expected to support the specific resource to which the request will be sent. Obtaining such list can

be achieved in various ways such as by group configuration, and/or CoAP discovery, and/or first sending one or more non-block-wise multicast requests to the same group and collect the responses.

The reason that the client should be aware of these group members is the following: after sending the first block (0), the client SHOULD first collect all group member responses to the first block before proceeding with further blocks. One or more of the group members MAY indicate a preference for a smaller block size in the Block1 Option in its first response. The client SHOULD use the smallest value over all collected responses as the block size to use for the remaining block-wise messages.

Since not all group member responses may be received, due to message loss, the client MAY resend the multicast request (with the same Message ID and Token) to collect the missing responses, or it MAY resend the block 0 request as a Confirmable or Non-Confirmable unicast request (with the same Message ID and Token) directly to the non-responsive group member(s), or it uses a combination of these. The reason to use the same Message ID here is to avoid that a group member server processes the request more than once.

TBD: open point - the server needs to treat a unicast message (with token T and MID M) as a duplicate of a prior multicast message (with token T and MID M). The deduplication rules allow this; however to be checked if a practical implementation also allows this?

TBD: open point - the time that the process takes to collect all "missing" responses for the first block (0), might take longer than the "operation timeout time" of the entire blockwise request per [RFC7959]. So for this case, the operation timeout time needs to be set longer than usual, or alternatively, the stateless-server mode of update needs to be mandated. In this case each block that is written produces a 2.04 not 2.31. First block with PUT may respond a 2.01.

TBD: if strict order of blocks is required by a server, the protocol must wait and collect again all responses after each block.

TBD: a protocol may be more efficient that first sends all blocks (without waiting for all responses every step) and then later checks which blocks are missing with all servers individually. These can be resent then (in unicast or multicast if many servers miss that block).

2.3. Transport

TBD: Mark [RFC8323] (TCP, TLS, WebSockets) as not applicable for this form of groupcomm, as well as CoAP-over-SMS.

2.3.1. UDP/IPv6 Multicast Transport

TBD: include the "Exceptions" cases here of RFC 7390 Section 2.10. State that IPv6 multicast is prerequisite. Also mention the All-CoAP-nodes IPv6 addresses.

2.3.2. UDP/IPv4 Multicast Transport

TBD: includes the "Exceptions" cases here of RFC 7390 2.10. State that IPv4 multicast is prerequisite. mention All-CoAP-nodes IPv4 addresses and the like

2.3.3. 6LoWPAN

TBD: 6lowpan-specific considerations to go here. Specifically, a multicast request should preferably fit in one L2 frame to avoid the strong performance drop that comes with 6LoWPAN-fragmentation and reassembly. Also reference [RFC7346] for the realm-local scope.

2.4. Interworking with Other Protocols

2.4.1. MLD/MLDv2/IGMP

TBD: see Section 4.2 of [RFC7390] and include the content here or refer to it.

2.4.2. RPL

TBD: see Section 4.3 of [RFC7390] and include the content here or refer to it.

2.4.3. MPL

TBD: see Section 4.4. [RFC7390] and include the content here or refer to it.

3. Unsecured Group Communication

CoAP group communication can operate in CoAP NoSec (No Security) mode, without using application-layer and transport-layer security mechanisms. The NoSec mode uses the "coap" scheme, and is defined in Section 9 of [RFC7252]. Before using this mode of operation, the security implications (Section 5.1) must be well understood.

4. Secured Group Communication using Group OSCORE

The application-layer protocol Object Security for Constrained RESTful Environments (OSCORE) [I-D.ietf-core-object-security] provides end-to-end encryption, integrity and replay protection of CoAP messages exchanged between two CoAP endpoints. These can act both as CoAP Client as well as CoAP Server, and share an OSCORE Security Context used to protect and verify exchanged messages. The use of OSCORE does not affect the URI scheme and OSCORE can therefore be used with any URI scheme defined for CoAP.

OSCORE uses COSE [RFC8152] to perform encryption, signing and Message Authentication Code operations, and to efficiently encode the result as a COSE object. In particular, OSCORE takes as input an unprotected CoAP message and transforms it into a protected CoAP message, by using Authenticated Encryption Algorithms with Additional Data (AEAD).

OSCORE makes it possible to selectively protect different parts of a CoAP message in different ways, so still allowing intermediaries (e.g., CoAP proxies) to perform their intended functionalities. That is, some message parts are encrypted and integrity protected; other parts only integrity protected to be accessible to, but not modifiable by, proxies; and some parts are kept as plain content to be both accessible to and modifiable by proxies. Such differences especially concern the CoAP options included in the unprotected message.

Group OSCORE [I-D.ietf-core-oscore-groupcomm] builds on OSCORE, and provides end-to-end security of CoAP messages exchanged between members of an OSCORE group, while fulfilling the same security requirements.

In particular, Group OSCORE protects CoAP requests sent over IP multicast by a CoAP client, as well as multiple corresponding CoAP responses sent over IP unicast by different CoAP servers. However, the same keying material can also be used to protect CoAP requests sent over IP unicast to a single CoAP server in the OSCORE group, as well as the corresponding responses.

Group OSCORE uses digital signatures to ensure source authentication of all messages exchanged within the OSCORE group. That is, sender devices sign their outgoing messages by means of their own private key, and embed the signature in the protected CoAP message.

A Group Manager is responsible for one or multiple OSCORE groups. In particular, the Group Manager acts as repository of public keys of

group members; manages, renews and provides keying material in the group; and drives the join process for new group members.

As recommended in [I-D.ietf-core-oscore-groupcomm], a CoAP endpoint can join an OSCORE group by using the method described in [I-D.ietf-ace-key-groupcomm-oscore] and based on the ACE framework for Authentication and Authorization in constrained environments [I-D.ietf-ace-oauth-authz].

A CoAP endpoint can discover OSCORE groups and retrieve information to join them through their Group Managers by using the method described in [I-D.tiloca-core-oscore-discovery] and based on the CoRE Resource Directory [I-D.ietf-core-resource-directory].

If security is required, CoAP group communication as described in this specification MUST use Group OSCORE. In particular, a CoAP group as defined in Section 2.1.1 and using secure group communication is associated to an OSCORE group, which includes:

- o All members of the CoAP group, i.e. the CoAP endpoints configured (also) as CoAP servers and listening to the group's multicast IP address.
- o All further CoAP endpoints configured only as CoAP clients, that send (multicast) CoAP requests to the CoAP group.

4.1. Secure Group Maintenance

Additional key management operations on the OSCORE group are required, depending also on the security requirements of the application (see Section 5.2). That is:

- o Adding new members to a CoAP group or enabling new client-only endpoints to interact with that group require also that each of such members/endpoints join the corresponding OSCORE group. By doing so, they are securely provided with the necessary cryptographic material. In case backward security is needed, this also requires to first renew such material and distribute it to the current members/endpoints, before new ones are added and join the OSCORE group.
- o In case forward security is needed, removing members from a CoAP group or stopping client-only endpoints from interacting with that group requires removing such members/endpoints from the corresponding OSCORE group. To this end, new cryptographic material is generated and securely distributed only to the remaining members/endpoints. This ensures that only the members/endpoints intended to remain are able to continue participating to

secure group communication, while the evicted ones are not able to.

The key management operations mentioned above are entrusted to the Group Manager responsible for the OSCORE group [I-D.ietf-core-oscore-groupcomm], and it is RECOMMENDED to perform them according to the approach described in [I-D.ietf-ace-key-groupcomm-oscore].

5. Security Considerations

This section provides security considerations for CoAP group communication using IP multicast.

5.1. CoAP NoSec Mode

CoAP group communication, if not protected, is vulnerable to all the attacks mentioned in Section 11 of [RFC7252] for IP multicast.

Thus, for sensitive and mission-critical applications (e.g., health monitoring systems and alarm monitoring systems), it is NOT RECOMMENDED to deploy CoAP group communication in NoSec mode.

Without application-layer security, CoAP group communication SHOULD only be deployed in applications that are non-critical, and that do not involve or may have an impact on sensitive data and personal sphere. These include, e.g., read-only temperature sensors deployed in non-sensitive environments, where the client reads out the values but does not use the data to control actuators or to base an important decision on.

Discovery of devices and resources is a typical use case where NoSec mode is applied, since the devices involved do not have yet configured any mutual security relations at the time the discovery takes place.

5.2. Group OSCORE

Group OSCORE provides end-to-end application-level security. This has many desirable properties, including maintaining security assurances while forwarding traffic through intermediaries (proxies). Application-level security also tends to more cleanly separate security from the dynamics of group membership (e.g., the problem of distributing security keys across large groups with many members that come and go).

For sensitive and mission-critical applications, CoAP group communication MUST be protected by using Group OSCORE as specified in

[I-D.ietf-core-oscore-groupcomm]. The same security considerations from Section 8 of [I-D.ietf-core-oscore-groupcomm] hold for this specification.

5.2.1. Group Key Management

A key management scheme for secure revocation and renewal of group keying material, namely group rekeying, should be adopted in OSCORE groups. In particular, the key management scheme should preserve backward and forward security in the OSCORE group, if the application requires so (see Section 2.1 of [I-D.ietf-core-oscore-groupcomm]).

Group policies should also take into account the time that the key management scheme requires to rekey the group, on one hand, and the expected frequency of group membership changes, i.e. nodes' joining and leaving, on the other hand.

In fact, it may be desirable to not rekey the group upon every single membership change, in case members' joining and leaving are frequent, and at the same time a single group rekeying instance takes a non negligible time to complete.

In such a case, the Group Manager may consider to rekey the group, e.g., after a minum number of nodes have joined or left the group within a pre-defined time interval, or according to communication patterns with predictable intervals of network inactivity. This would prevent paralyzing communications in the group, when a slow rekeying scheme is used and frequently invoked.

This comes at the cost of not continuously preserving backward and forward security, since group rekeying might not occur upon every single group membership change. That is, latest joined nodes would have access to the key material used prior to their join, and thus be able to access past group communications protected with that key material. Similarly, until the group is rekeyed, latest left nodes would preserve access to group communications protected with the retained key material.

5.2.2. Source Authentication

CoAP endpoints using Group OSCORE countersign their outgoing messages, by means of the countersignature algorithm used in the OSCORE group. This ensures source authentication of messages exchanged by CoAP endpoints through CoAP group communication. In fact, it allows to verify that a received message has actually been originated by a specific and identified member of the OSCORE group.

Appendix F of [I-D.ietf-core-oscore-groupcomm] discusses a number of cases where a recipient CoAP endpoint may skip the verification of countersignatures, possibly on a per-message basis. However, this is NOT RECOMMENDED. That is, a CoAP endpoint receiving a message secured with Group OSCORE SHOULD always verify the countersignature.

5.2.3. Counteraction of Attacks

Group OSCORE addresses security attacks mentioned in Sections 11.2–11.6 of [RFC7252], with particular reference to their execution over IP multicast. That is: it provides confidentiality and integrity of request/response data through proxies also in multicast settings; it prevents amplification attacks carried out through responses to injected requests over IP multicast; it limits the impact of attacks based on IP spoofing; it prevents cross-protocol attacks; it derives the group key material from, among other things, a Master Secret securely generated by the Group Manager and provided to CoAP endpoints upon their joining of the OSCORE group; countersignatures assure source authentication of exchanged CoAP messages, and hence prevent a group member to be used for subverting security in the whole group.

5.3. 6LoWPAN

Editor Note, TBD: identify if multi-fragment multicast requests have a negative effect on security and, if so, advice here on trying to avoid such requests. Also an attacker could use multi-fragment to occupy reassembly buffers of many routing 6LoWPAN nodes.

5.4. Wi-Fi

TBD: Wi-Fi specific security considerations; see also Section 5.3.1 of [RFC7390].

5.5. Monitoring

TBD: see Section 5.4 of [RFC7390].

6. IANA Considerations

This document has no actions for IANA.

7. References

7.1. Normative References

- [I-D.ietf-core-object-security]
Selander, G., Mattsson, J., Palombini, F., and L. Seitz,
"Object Security for Constrained RESTful Environments
(OSCORE)", draft-ietf-core-object-security-16 (work in
progress), March 2019.
- [I-D.ietf-core-oscore-groupcomm]
Tiloca, M., Selander, G., Palombini, F., and J. Park,
"Group OSCORE - Secure Group Communication for CoAP",
draft-ietf-core-oscore-groupcomm-04 (work in progress),
March 2019.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate
Requirement Levels", BCP 14, RFC 2119,
DOI 10.17487/RFC2119, March 1997,
<<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC4944] Montenegro, G., Kushalnagar, N., Hui, J., and D. Culler,
"Transmission of IPv6 Packets over IEEE 802.15.4
Networks", RFC 4944, DOI 10.17487/RFC4944, September 2007,
<<https://www.rfc-editor.org/info/rfc4944>>.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link
Format", RFC 6690, DOI 10.17487/RFC6690, August 2012,
<<https://www.rfc-editor.org/info/rfc6690>>.
- [RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object
Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049,
October 2013, <<https://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,
<<https://www.rfc-editor.org/info/rfc7252>>.
- [RFC7641] Hartke, K., "Observing Resources in the Constrained
Application Protocol (CoAP)", RFC 7641,
DOI 10.17487/RFC7641, September 2015,
<<https://www.rfc-editor.org/info/rfc7641>>.
- [RFC7959] Bormann, C. and Z. Shelby, Ed., "Block-Wise Transfers in
the Constrained Application Protocol (CoAP)", RFC 7959,
DOI 10.17487/RFC7959, August 2016,
<<https://www.rfc-editor.org/info/rfc7959>>.

- [RFC8075] Castellani, A., Loreto, S., Rahman, A., Fossati, T., and E. Dijk, "Guidelines for Mapping Implementations: HTTP to the Constrained Application Protocol (CoAP)", RFC 8075, DOI 10.17487/RFC8075, February 2017, <<https://www.rfc-editor.org/info/rfc8075>>.
- [RFC8152] Schaad, J., "CBOR Object Signing and Encryption (COSE)", RFC 8152, DOI 10.17487/RFC8152, July 2017, <<https://www.rfc-editor.org/info/rfc8152>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8323] Bormann, C., Lemay, S., Tschofenig, H., Hartke, K., Silverajan, B., and B. Raymor, Ed., "CoAP (Constrained Application Protocol) over TCP, TLS, and WebSockets", RFC 8323, DOI 10.17487/RFC8323, February 2018, <<https://www.rfc-editor.org/info/rfc8323>>.

7.2. Informative References

- [Californium]
Eclipse Foundation, "Eclipse Californium", March 2019, <<https://github.com/eclipse/californium/tree/2.0.x/californium-core/src/main/java/org/eclipse/californium/core>>.
- [Go-OCF] Open Connectivity Foundation (OCF), "Implementation of CoAP Server & Client in Go", March 2019, <<https://github.com/go-ocf/go-coap>>.
- [I-D.ietf-ace-key-groupcomm-oscore]
Tiloca, M., Park, J., and F. Palombini, "Key Management for OSCORE Groups in ACE", draft-ietf-ace-key-groupcomm-oscore-01 (work in progress), March 2019.
- [I-D.ietf-ace-oauth-authz]
Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "Authentication and Authorization for Constrained Environments (ACE) using the OAuth 2.0 Framework (ACE-OAuth)", draft-ietf-ace-oauth-authz-24 (work in progress), March 2019.

- [I-D.ietf-core-coap-pubsub]
Koster, M., Keranen, A., and J. Jimenez, "Publish-Subscribe Broker for the Constrained Application Protocol (CoAP)", draft-ietf-core-coap-pubsub-08 (work in progress), March 2019.
- [I-D.ietf-core-multipart-ct]
Fossati, T., Hartke, K., and C. Bormann, "Multipart Content-Format for CoAP", draft-ietf-core-multipart-ct-03 (work in progress), March 2019.
- [I-D.ietf-core-resource-directory]
Shelby, Z., Koster, M., Bormann, C., Stok, P., and C. Amsuess, "CoRE Resource Directory", draft-ietf-core-resource-directory-22 (work in progress), July 2019.
- [I-D.tiloca-core-oscore-discovery]
Tiloca, M., Amsuess, C., and P. Stok, "Discovery of OSCORE Groups with the CoRE Resource Directory", draft-tiloca-core-oscore-discovery-02 (work in progress), March 2019.
- [RFC7346] Droms, R., "IPv6 Multicast Address Scopes", RFC 7346, DOI 10.17487/RFC7346, August 2014, <<https://www.rfc-editor.org/info/rfc7346>>.
- [RFC7390] Rahman, A., Ed. and E. Dijk, Ed., "Group Communication for the Constrained Application Protocol (CoAP)", RFC 7390, DOI 10.17487/RFC7390, October 2014, <<https://www.rfc-editor.org/info/rfc7390>>.
- [RFC7967] Bhattacharyya, A., Bandyopadhyay, S., Pal, A., and T. Bose, "Constrained Application Protocol (CoAP) Option for No Server Response", RFC 7967, DOI 10.17487/RFC7967, August 2016, <<https://www.rfc-editor.org/info/rfc7967>>.

Appendix A. Use Cases

To illustrate where and how CoAP-based group communication can be used, this section summarizes the most common use cases. These use cases include both secured and non-secured CoAP usage. Each subsection below covers one particular category of use cases for CoRE. Within each category, a use case may cover multiple application areas such as home IoT, commercial building IoT (sensing and control), industrial IoT/control, or environmental sensing.

A.1. Discovery

Discovery of physical devices in a network, or discovery of information entities hosted on network devices, are operations that are usually required in a system during the phases of setup or (re)configuration. When a discovery use case involves devices that need to interact without having been configured previously with a common security context, unsecured CoAP communication is typically used. Discovery may involve a request to a directory server, which provides services to aid clients in the discovery process. One particular type of directory server is the CoRE Resource Directory [I-D.ietf-core-resource-directory]; and there may be other types of directories that can be used with CoAP.

A.1.1. Distributed Device Discovery

Device discovery is the discovery and identification of networked devices – optionally only devices of a particular class, type, model, or brand. Group communication is used for distributed device discovery, if a central directory server is not used. Typically in distributed device discovery, a multicast request is sent to a particular address (or address range) and multicast scope of interest, and any devices configured to be discoverable will respond back. For the alternative solution of centralized device discovery a central directory server is accessed through unicast, in which case group communication is not needed. This requires that the address of the central directory is either preconfigured in each device or configured during operation using a protocol.

In CoAP, device discovery can be implemented by CoAP resource discovery requesting (GET) a particular resource that the sought device class, type, model or brand is known to respond to. It can also be implemented using CoAP resource discovery (Section 7 of [RFC7252]) and the CoAP query interface defined in Section 4 of [RFC6690] to find these particular resources. Also, a multicast GET request to /.well-known/core can be used to discover all CoAP devices.

A.1.2. Distributed Service Discovery

Service discovery is the discovery and identification of particular services hosted on network devices. Services can be identified by one or more parameters such as ID, name, protocol, version and/or type. Distributed service discovery involves group communication to reach individual devices hosting a particular service; with a central directory server not being used.

In CoAP, services are represented as resources and service discovery is implemented using resource discovery (Section 7 of [RFC7252]) and the CoAP query interface defined in Section 4 of [RFC6690].

A.1.3. Directory Discovery

This use case is a specific sub-case of Distributed Service Discovery (Appendix A.1.2), in which a device needs to identify the location of a Directory on the network to which it can e.g. register its own offered services, or to which it can perform queries to identify and locate other devices/services it needs to access on the network. Section 3.3 of [RFC7390] shows an example of discovering a CoRE Resource Directory using CoAP group communication. As defined in [I-D.ietf-core-resource-directory], a resource directory is a web entity that stores information about web resources and implements REST interfaces for registration and lookup of those resources. For example, a device can register itself to a resource directory to let it be found by other devices and/or applications.

A.2. Operational Phase

Operational phase use cases describe those operations that occur most frequently in a networked system, during its operational lifetime and regular operation. Regular usage is when the applications on networked devices perform the tasks they were designed for and exchange of application-related data using group communication occurs. Processes like system reconfiguration, group changes, system/device setup, extra group security changes, etc. are not part of regular operation.

A.2.1. Actuator Group Control

Group communication can be beneficial to control actuators that need to act in synchrony, as a group, with strict timing (latency) requirements. Examples are office lighting, stage lighting, street lighting, or audio alert/Public Address systems. Sections 3.4 and 3.5 of [RFC7390] show examples of lighting control of a group of 6LoWPAN-connected lights.

A.2.2. Device Group Status Request

To properly monitor the status of systems, there may be a need for ad-hoc, unplanned status updates. Group communication can be used to quickly send out a request to a (potentially large) number of devices for specific information. Each device then responds back with the requested data. Those devices that did not respond to the request can optionally be polled again via reliable unicast communication to complete the dataset. The device group may be defined e.g. as "all

temperature sensors on floor 3", or "all lights in wing B". For example, it could be a status request for device temperature, most recent sensor event detected, firmware version, network load, and/or battery level.

A.2.3. Network-wide Query

In some cases a whole network or subnet of multiple IP devices needs to be queried for status or other information. This is similar to the previous use case except that the device group is not defined in terms of its function/type but in terms of its network location. Technically this is also similar to distributed service discovery (Appendix A.1.2) where a query is processed by all devices on a network – except that the query is not about services offered by the device, but rather specific operational data is requested.

A.2.4. Network-wide / Group Notification

In some cases a whole network, or subnet of multiple IP devices, or a specific target group needs to be notified of a status change or other information. This is similar to the previous two use cases except that the recipients are not expected to respond with some information. Unreliable notification can be acceptable in some use cases, in which a recipient does not respond with a confirmation of having received the notification. In such a case, the receiving CoAP server does not have to create a CoAP response. If the sender needs confirmation of reception, the CoAP servers can be configured for that resource to respond with a 2.xx success status after processing a notification request successfully.

A.3. Software Update

Multicast can be useful to efficiently distribute new software (firmware, image, application, etc.) to a group of multiple devices. In this case, the group is defined in terms of device type: all devices in the target group are known to be capable of installing and running the new software. The software is distributed as a series of smaller blocks that are collected by all devices and stored in memory. All devices in the target group are usually responsible for integrity verification of the received software; which can be done per-block or for the entire software image once all blocks have been received. Due to the inherent unreliability of CoAP multicast, there needs to be a backup mechanism (e.g. implemented using CoAP unicast) by which a device can individually request missing blocks of a whole software image/entity. Prior to multicast software update, the group of recipients can be separately notified that there is new software available and coming, using the above network-wide or group notification.

Acknowledgments

The authors sincerely thank Thomas Fossati and Jim Schaad for their comments and feedback.

The work on this document has been partly supported by VINNOVA and the Celtic-Next project CRITISEC.

Authors' Addresses

Esko Dijk
IoTconsultancy.nl

Utrecht
The Netherlands

Email: esko.dijk@iotconsultancy.nl

Chonggang Wang
InterDigital
1001 E Hector St, Suite 300
Conshohocken PA 19428
United States

Email: Chonggang.Wang@InterDigital.com

Marco Tiloca
RISE AB
Isafjordsgatan 22
Kista SE-16440 Stockholm
Sweden

Email: marco.tiloca@ri.se

ACE Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 6, 2020

F. Palombini
Ericsson AB
M. Tiloca
RISE AB
July 05, 2019

Key Provisioning for Group Communication using ACE
draft-ietf-ace-key-groupcomm-02

Abstract

This document defines message formats and procedures for requesting and distributing group keying material using the ACE framework, to protect communications between group members.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on January 6, 2020.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

| | |
|--|----|
| 1. Introduction | 3 |
| 1.1. Terminology | 3 |
| 2. Overview | 4 |
| 3. Authorization to Join a Group | 6 |
| 3.1. Authorization Request | 7 |
| 3.2. Authorization Response | 7 |
| 3.3. Token Post | 8 |
| 4. Key Distribution | 11 |
| 4.1. Key Distribution Request | 12 |
| 4.2. Key Distribution Response | 13 |
| 5. Removal of a Node from the Group | 16 |
| 5.1. Expired Authorization | 16 |
| 5.2. Request to Leave the Group | 16 |
| 6. Retrieval of New or Updated Keying Material | 17 |
| 6.1. Key Re-Distribution Request | 18 |
| 6.2. Key Re-Distribution Response | 19 |
| 7. Retrieval of Public Keys for Group Members | 19 |
| 7.1. Public Key Request | 20 |
| 7.2. Public Key Response | 20 |
| 8. ACE Groupcomm Parameters | 21 |
| 9. ACE Groupcomm Request Type | 22 |
| 10. Security Considerations | 23 |
| 10.1. Update of Keying Material | 24 |
| 10.2. Block-Wise Considerations | 24 |
| 11. IANA Considerations | 25 |
| 11.1. ACE Authorization Server Request Creation Hints Registry | 25 |
| 11.2. ACE Public Key Encoding Registry | 25 |
| 11.3. ACE Groupcomm Parameters Registry | 26 |
| 11.4. Ace Groupcomm Request Type Registry | 26 |
| 11.5. ACE Groupcomm Key Registry | 27 |
| 11.6. ACE Groupcomm Profile Registry | 28 |
| 11.7. ACE Groupcomm Policy Registry | 28 |
| 11.8. Sequence Number Synchronization Method Registry | 29 |
| 11.9. Expert Review Instructions | 29 |
| 12. References | 30 |
| 12.1. Normative References | 30 |
| 12.2. Informative References | 31 |
| Appendix A. Requirements on Application Profiles | 33 |
| Appendix B. Document Updates | 33 |
| B.1. Version -01 to -02 | 34 |
| B.2. Version -00 to -01 | 34 |
| Acknowledgments | 35 |
| Authors' Addresses | 35 |

1. Introduction

This document expands the ACE framework [I-D.ietf-ace-oauth-authz] to define the format of messages used to request, distribute and renew the keying material in a group communication scenario, e.g. based on multicast [RFC7390][I-D.dijk-core-groupcomm-bis] or on publishing-subscribing [I-D.ietf-core-coap-pubsub]. The ACE framework is based on CBOR [RFC7049], so CBOR is the format used in this specification. However, using JSON [RFC8259] instead of CBOR is possible, using the conversion method specified in Sections 4.1 and 4.2 of [RFC7049].

Profiles that use group communication can build on this document to specify the selection of the message parameters defined in this document to use and their values. Known applications that can benefit from this document would be, for example, those addressing group communication based on multicast [RFC7390][I-D.dijk-core-groupcomm-bis] or publishing/subscribing [I-D.ietf-core-coap-pubsub] in ACE.

If the application requires backward and forward security, updated keying material is generated and distributed to the group members (rekeying), when membership changes. A key management scheme performs the actual distribution of the updated keying material to the group. In particular, the key management scheme rekeys the current group members when a new node joins the group, and the remaining group members when a node leaves the group. This document provides a message format for group rekeying that allows to fulfill these requirements. Rekeying mechanisms can be based on [RFC2093], [RFC2094] and [RFC2627].

1.1. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119]. These words may also appear in this document in lowercase, absent their normative meanings.

Readers are expected to be familiar with the terms and concepts described in [I-D.ietf-ace-oauth-authz] and [RFC8152], such as Authorization Server (AS) and Resource Server (RS).

This document additionally uses the following terminology:

- o Transport profile, to indicate a profile of ACE as per Section 5.6.4.3 of [I-D.ietf-ace-oauth-authz]. That is, a transport profile specifies the communication protocol and communication security protocol between an ACE Client and Resource

Server, as well as proof-of-possession methods, if it supports proof-of-possession access tokens. Transport profiles of ACE include, for instance, [I-D.ietf-ace-oscore-profile], [I-D.ietf-ace-dtls-authorize] and [I-D.ietf-ace-mqtt-tls-profile].

- o Application profile, to indicate a profile of ACE that defines how applications enforce and use supporting security services they require. These services include, for instance, provisioning, revocation and (re-)distribution of keying material. An application profile may define specific procedures and message formats.

2. Overview

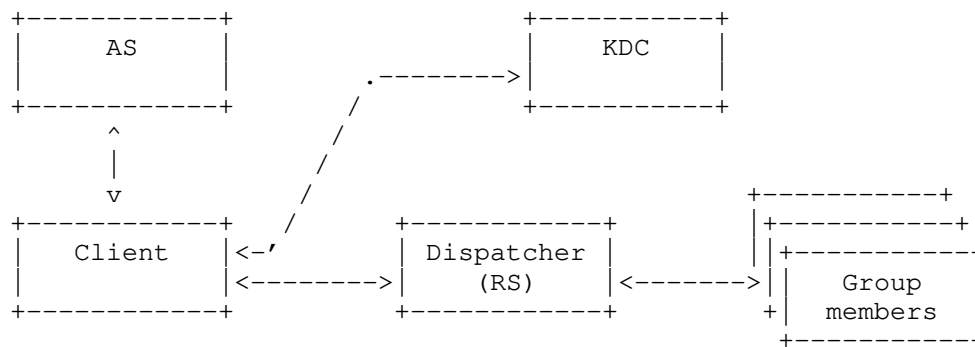


Figure 1: Key Distribution Participants

The following participants (see Figure 1) take part in the authorization and key distribution.

- o Client (C): node that wants to join the group communication. It can request write and/or read rights.
- o Authorization Server (AS): same as AS in the ACE Framework; it enforces access policies, and knows if a node is allowed to join the group with write and/or read rights.
- o Key Distribution Center (KDC): maintains the keying material to protect group communications, and provides it to Clients authorized to join the group. During the first part of the exchange (Section 3), it takes the role of the RS in the ACE Framework. During the second part (Section 4), which is not based on the ACE Framework, it distributes the keying material. In addition, it provides the latest keying material to group members when requested. If required by the application, the KDC renews

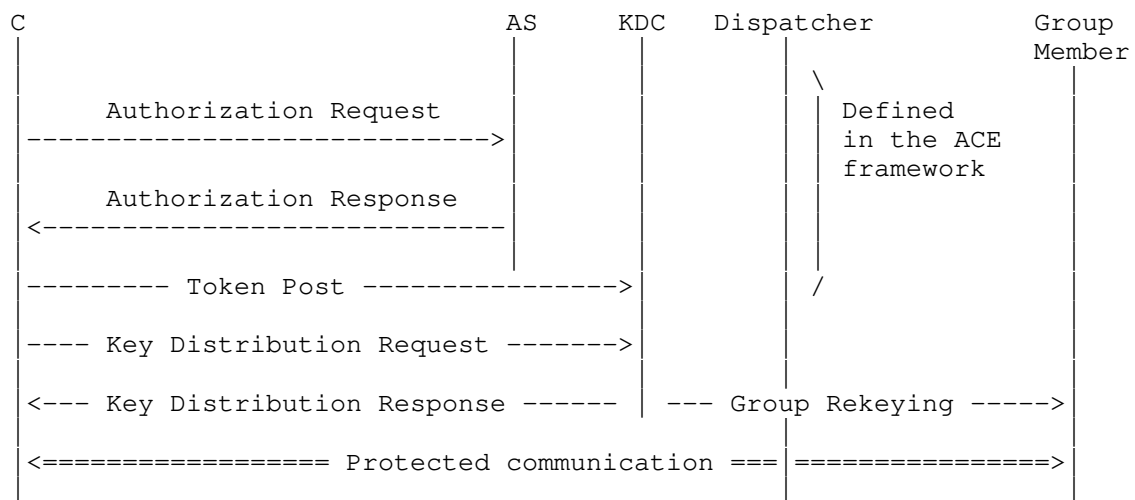
and re-distributes the keying material in the group when membership changes.

- o **Dispatcher:** entity through which the Clients communicate with the group and which distributes messages to the group members. Examples of dispatchers are: the Broker node in a pub-sub setting; a relay node for group communication that delivers group messages as multiple unicast messages to all group members; an implicit entity as in a multicast communication setting, where messages are transmitted to a multicast IP address and delivered on the transport channel.

This document specifies the message flows and formats for:

- o Authorizing a new node to join the group (Section 3), and providing it with the group keying material to communicate with the other group members (Section 4).
- o Removing of a current member from the group (Section 5).
- o Retrieving keying material as a current group member (Section 6 and Section 7).
- o Renewing and re-distributing the group keying material (rekeying) upon a membership change in the group (Section 4.2 and Section 5).

Figure 2 provides a high level overview of the message flow for a node joining a group communication setting.



The exchange of Authorization Request and Authorization Response between Client and AS MUST be secured, as specified by the transport profile of ACE used between Client and KDC.

The exchange of Key Distribution Request and Key Distribution Response between Client and KDC MUST be secured, as a result of the transport profile of ACE used between Client and KDC.

All further communications between the Client and the KDC MUST be secured, for instance with the same security mechanism used for the Key Distribution exchange.

All communications between a Client and the other group members MUST be secured using the keying material provided in Section 4.

3. Authorization to Join a Group

This section describes in detail the format of messages exchanged by the participants when a node requests access to a group. The first part of the exchange is based on ACE [I-D.ietf-ace-oauth-authz].

As defined in [I-D.ietf-ace-oauth-authz], the Client requests from the AS an authorization to join the group through the KDC (see Section 3.1). If the request is approved and authorization is granted, the AS provides the Client with a proof-of-possession access token and parameters to securely communicate with the KDC (see Section 3.2). Communications between the Client and the AS MUST be secured, according to the transport profile of ACE used. The Content-Format used in the messages is the one specified by the used transport profile of ACE (e.g. application/ace+cbor for the first two messages and application/cwt for the third message, depending on the format of the access token).

Figure 3 gives an overview of the exchange described above.

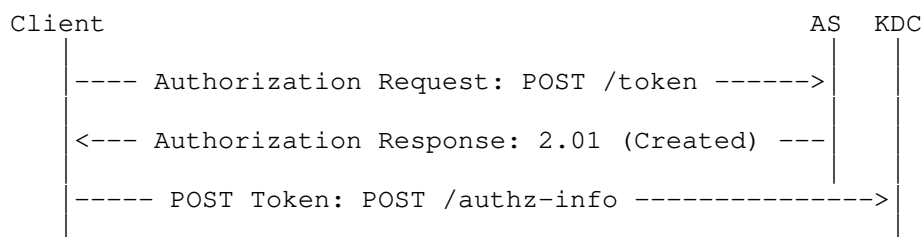


Figure 3: Message Flow of Join Authorization

3.1. Authorization Request

The Authorization Request sent from the Client to the AS is as defined in Section 5.6.1 of [I-D.ietf-ace-oauth-authz] and MUST contain the following parameters:

- o 'grant_type', with value "client_credentials".

Additionally, the Authorization Request MAY contain the following parameters, which, if included, MUST have the corresponding values:

- o 'scope', containing the identifier of the specific group (or topic in the case of pub-sub) that the Client wishes to access, and optionally the role(s) that the Client wishes to take. This value is a CBOR array encoded as a byte string, which contains:

- * As first element, the identifier of the specific group or topic.
- * Optionally, as second element, the role (or CBOR array of roles) the Client wishes to take in the group.

The encoding of the group or topic identifier and of the role identifiers is application specific.

- o 'audience', with an identifier of a KDC.
- o 'req_cnf', as defined in Section 3.1 of [I-D.ietf-ace-oauth-params], optionally containing the public key or a reference to the public key of the Client, if it wishes to communicate that to the AS.
- o Other additional parameters as defined in [I-D.ietf-ace-oauth-authz], if necessary.

3.2. Authorization Response

The Authorization Response sent from the AS to the Client is as defined in Section 5.6.2 of [I-D.ietf-ace-oauth-authz] and MUST contain the following parameters:

- o 'access_token', containing the proof-of-possession access token.
- o 'cnf' if symmetric keys are used, not present if asymmetric keys are used. This parameter is defined in Section 3.2 of [I-D.ietf-ace-oauth-params] and contains the symmetric proof-of-possession key that the Client is supposed to use with the KDC.

- o 'rs_cnf' if asymmetric keys are used, not present if symmetric keys are used. This parameter is as defined in Section 3.2 of [I-D.ietf-ace-oauth-params] and contains information about the public key of the KDC.
- o 'exp', contains the lifetime in seconds of the access token. This parameter MAY be omitted if the application defines how the expiration time is communicated to the Client via other means, or if it establishes a default value.

Additionally, the Authorization Response MAY contain the following parameters, which, if included, MUST have the corresponding values:

- o 'scope', which mirrors the 'scope' parameter in the Authorization Request (see Section 3.1). Its value is a CBOR array encoded as a byte string, containing:
 - * As first element, the identifier of the specific group or topic the Client is authorized to access.
 - * Optionally, as second element, the role (or CBOR array of roles) the Client is authorized to take in the group.

The encoding of the group or topic identifier and of the role identifiers is application specific.

- o Other additional parameters as defined in [I-D.ietf-ace-oauth-authz], if necessary.

The access token MUST contain all the parameters defined above (including the same 'scope' as in this message, if present, or the 'scope' of the Authorization Request otherwise), and additionally other optional parameters that the transport profile of ACE requires.

When receiving an Authorization Request from a Client that was previously authorized, and which still owns a valid non expired access token, the AS replies with an Authorization Response with a new access token.

3.3. Token Post

The Client sends a CoAP POST request including the access token to the KDC, as specified in Section 5.8.1 of [I-D.ietf-ace-oauth-authz]. If the specific transport profile of ACE defines it, the Client MAY use a different endpoint than /authz-info at the KDC to post the access token to.

Optionally, the Client might need to request necessary information concerning the public keys in the group, as well as concerning the algorithm and related parameters for computing signatures in the group. In such a case, the joining node MAY ask for that information to the KDC in this same request. To this end, it sends the CoAP POST request to the /authz-info endpoint using the Content-Format "application/ace+cbor" defined in Section 8.14 of [I-D.ietf-ace-oauth-authz], and includes also the following parameters:

- o 'sign_info' defined in Section 3.3.1, encoding the CBOR simple value Null, to require information and parameters on the signature algorithm and on the public keys used in the group.
- o 'pub_key_enc' defined in Section 3.3.2, encoding the CBOR simple value Null, to require information on the exact encoding of public keys used in the group.

The CDDL notation of the 'sign_info' and 'pub_key_enc' parameters formatted as in the request is given below.

```
sign_info_req = nil
```

```
pub_key_enc_req = nil
```

Alternatively, the joining node may retrieve this information by other means.

After successful verification, the Client is authorized to receive the group keying material from the KDC and join the group. In particular, the KDC replies to the Client with a 2.01 (Created) response, using Content-Format "application/ace+cbor" defined in Section 8.14 of [I-D.ietf-ace-oauth-authz].

The payload of the 2.01 response is a CBOR map, which MUST include a nonce N generated by the KDC. The Client may use this nonce for proving the possession of its own private key (see the 'client_cred_verify' parameter in Section 4).

Optionally, if they were included in the request, the AS MAY include the 'sign_info' parameter as well as the 'pub_key_enc' parameter defined in Section 3.3.1 and Section 3.3.2 of this specification, respectively.

The 'sign_info' parameter MUST be present if the POST request included the 'sign_info' parameter with value Null. If present, the 'sign_info' parameter of the 2.01 (Created) response is a CBOR array formatted as follows.

- o The first element 'sign_alg' is an integer or a text string, indicating the signature algorithm used in the group. It is required of the application profiles to define specific values for this parameter.
- o The second element 'sign_parameters' indicates the parameters of the signature algorithm. Its structure depends on the value of 'sign_alg'. It is required of the application profiles to define specific values for this parameter. If no parameters of the signature algorithm are specified, 'sign_parameters' MUST be encoding the CBOR simple value Null.
- o The third element 'sign_key_parameters' indicates the parameters of the key used with the signature algorithm. Its structure depends on the value of 'sign_alg'. It is required of the application profiles to define specific values for this parameter. If no parameters of the key used with the signature algorithm are specified, 'sign_key_parameters' MUST be encoding the CBOR simple value Null.

The 'pub_key_enc' parameter MUST be present if the POST request included the 'pub_key_enc' parameter with value Null. If present, the 'pub_key_enc' parameter of the 2.01 (Created) response is a CBOR integer, indicating the encoding of public keys used in the group. The values of this field are registered in the "ACE Public Key Encoding" Registry, defined in Section 11.2. It is required of the application profiles to define specific values to use for this parameter.

The CDDL notation of the 'sign_info' and 'pub_key_enc' parameters formatted as in the response is given below.

```
sign_info_res = [  
    sign_alg : int / tstr,  
    sign_parameters : any / nil,  
    sign_key_parameters : any / nil  
]  
  
pub_key_enc_res = int
```

Note that the CBOR map specified as payload of the 2.01 (Created) response may include further parameters, e.g. according to the signalled transport profile of ACE.

Note that this step could be merged with the following message from the Client to the KDC, namely Key Distribution Request.

3.3.1. 'sign_info' Parameter

The 'sign_info' parameter is an OPTIONAL parameter of the AS Request Creation Hints message defined in Section 5.1.2. of [I-D.ietf-ace-oauth-authz]. This parameter contains information and parameters about the signature algorithm and the public keys to be used between the Client and the RS. Its exact content is application specific.

3.3.2. 'pub_key_enc' Parameter

The 'pub_key_enc' parameter is an OPTIONAL parameter of the AS Request Creation Hints message defined in Section 5.1.2. of [I-D.ietf-ace-oauth-authz]. This parameter contains information about the exact encoding of public keys to be used between the Client and the RS. Its exact content is application specific.

4. Key Distribution

This section defines how the keying material used for group communication is distributed from the KDC to the Client, when joining the group as a new member.

If not previously established, the Client and the KDC MUST first establish a pairwise secure communication channel using ACE. The exchange of Key Distribution Request-Response MUST occur over that secure channel. The Client and the KDC MAY use that same secure channel to protect further pairwise communications, that MUST be secured.

During this exchange, the Client sends a request to the AS, specifying the group it wishes to join (see Section 4.1). Then, the KDC verifies the access token and that the Client is authorized to join that group; if so, it provides the Client with the keying material to securely communicate with the member of the group (see Section 4.2). The Content-Format used in the messages is set to application/cbor.

Figure 4 gives an overview of the exchange described above.

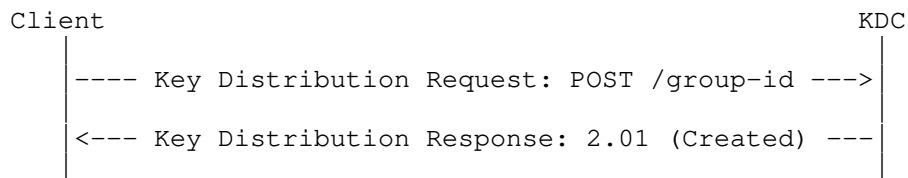


Figure 4: Message Flow of Key Distribution to a New Group Member

The same set of message can also be used for the following cases, when the Client is already a group member:

- o The Client wishes to (re-)get the current keying material, for cases such as expiration, loss or suspected mismatch, due to e.g. reboot or missed group rekeying. This is further discussed in Section 6.
- o The Client wishes to (re-)get the public keys of other group members, e.g. if it is aware of new nodes joining the group after itself. This is further discussed in Section 7.

Additionally, the format of the payload of the Key Distribution Response (Section 4.2) can be reused for messages sent by the KDC to distribute updated group keying material, in case of a new node joining the group or of a current member leaving the group. The key management scheme used to send such messages could rely on, e.g., multicast in case of a new node joining or unicast in case of a node leaving the group.

Note that proof-of-possession to bind the access token to the Client is performed by using the proof-of-possession key bound to the access token for establishing secure communication between the Client and the KDC.

If the application requires backward security, the KDC SHALL generate new group keying material and securely distribute it to all the current group members, using the message format defined in this section. Application profiles may define alternative message formats.

4.1. Key Distribution Request

The Client sends a Key Distribution Request to the KDC. This corresponds to a CoAP POST request to the endpoint in the KDC associated to the group to join. The endpoint in the KDC is associated to the 'scope' value of the Authorization Request/Response. The payload of this request is a CBOR map which MUST contain the following fields:

- o 'type', encoded as a CBOR int, with value 1 ("key distribution").

Additionally, the CBOR map in the payload MAY contain the following fields, which, if included, MUST have the corresponding values:

- o 'scope', with value the specific resource that the Client is authorized to access (i.e. group or topic identifier) and role(s), encoded as in Section 3.1.

- o 'get_pub_keys', if the Client wishes to receive the public keys of the other nodes in the group from the KDC. The value is an empty CBOR array. This parameter may be present if the KDC stores the public keys of the nodes in the group and distributes them to the Client; it is useless to have here if the set of public keys of the members of the group is known in another way, e.g. it was provided by the AS.
- o 'client_cred', with value the public key or certificate of the Client, encoded as a CBOR byte string. If the KDC is managing (collecting from/distributing to the Client) the public keys of the group members, this field contains the public key of the Client. The default encoding for public keys is COSE Keys. Alternative specific encodings of this parameter MAY be defined in applications of this specification.
- o 'client_cred_verify', encoded as a CBOR byte string. This parameter contains a signature computed by the Client over the nonce N received from the KDC in the 2.01 (Created) response to the token POST request (see Section 3.3). The Client computes the signature by using its own private key, whose corresponding public key is either directly specified in the 'client_cred' parameter or included in the certificate specified in the 'client_cred' parameter. This parameter MUST be present if the 'client_cred' parameter is present.
- o 'pub_keys_repos', can be present if a certificate is present in the 'client_cred' field, with value a list of public key repositories storing the certificate of the Client. This parameter is encoded as a CBOR array of CBOR text strings, each of which specifies the URI of a key repository.

4.2. Key Distribution Response

The KDC verifies that the 'scope' received in the Key Distribution Request, if present, is a subset of the 'scope' stored in the access token associated to this client. If verification fails, the KDC MUST respond with a 4.01 (Unauthorized) error message.

If the Key Distribution Request is not formatted correctly (e.g. no 'scope' field present while expected, or unknown fields present), the KDC MUST respond with 4.00 (Bad Request) error message.

If verification succeeds, the KDC sends a Key Distribution success Response to the Client. The Key Distribution success Response corresponds to a 2.01 Created message. The payload of this response is a CBOR map, which MUST contain:

- o 'kty', identifying the key type of the 'key' parameter. The set of values can be found in the "Key Type" column of the "ACE Groupcomm Key" Registry. Implementations MUST verify that the key type matches the application profile being used, if present, as registered in the "ACE Groupcomm Key" registry.
- o 'key', containing the keying material for the group communication, or information required to derive it.

The exact format of the 'key' value MUST be defined in applications of this specification. Additionally, documents specifying the key format MUST register it in the "ACE Groupcomm Key" registry, including its name, type and application profile to be used with, as defined in the "ACE Groupcomm Key" registry, defined in Section 11.5.

| Name | Key Type Value | Profile | Description |
|----------|----------------|---------|------------------------|
| Reserved | 0 | | This value is reserved |

Figure 5: Key Type Values

Optionally, the Key Distribution Response MAY contain the following parameters, which, if included, MUST have the corresponding values:

- o 'profile', with value a CBOR integer that MUST be used to uniquely identify the application profile for group communication. The value MUST be registered in the "ACE Groupcomm Profile" Registry.
- o 'exp', with value the expiration time of the keying material for the group communication, encoded as a CBOR unsigned integer or floating-point number. This field contains a numeric value representing the number of seconds from 1970-01-01T00:00:00Z UTC until the specified UTC date/time, ignoring leap seconds, analogous to what specified in Section 2 of [RFC7519].
- o 'pub_keys', may only be present if 'get_pub_keys' was present in the Key Distribution Request. This parameter is a CBOR byte string, which encodes the public keys of all the group members paired with the respective member identifiers. The default encoding for public keys is COSE Keys, so the default encoding for 'pub_keys' is a CBOR byte string wrapping a COSE_KeySet (see [RFC8152]), which contains the public keys of all the members of the group. In particular, each COSE Key in the COSE_KeySet includes the identifier of the corresponding group member as value of its 'kid' key parameter. Alternative specific encodings of

this parameter MAY be defined in applications of this specification.

- o 'group_policies', with value a CBOR map, whose entries specify how the group handles specific management aspects. These include, for instance, approaches to achieve synchronization of sequence numbers among group members. The elements of this field are registered in the "ACE Groupcomm Policy" Registry. This specification defines the two elements "Sequence Number Synchronization Method" and "Key Update Check Interval", which are summarized in Figure 6. Application profiles that build on this document MUST specify the exact content format of included map entries.

| Name | CBOR label | CBOR type | Description | Reference |
|--|------------|-----------|--|-------------------|
| Sequence Number Synchronization Method | TBD1 | tstr/int | Method for a recipient node to synchronize with sequence numbers of a sender node. Its value is taken from the 'Value' column of the Sequence Number Synchronization Method registry | [[this document]] |
| Key Update Check Interval | TBD2 | int | Polling interval in seconds, to check for new keying material at the KDC | [[this document]] |

Figure 6: ACE Groupcomm Policies

- o 'mgt_key_material', encoded as a CBOR byte string and containing the administrative keying material to participate in the group rekeying performed by the KDC. The exact format and content depend on the specific rekeying scheme used in the group, which may be specified in the application profile.

Specific application profiles that build on this document need to specify how exactly the keying material is used to protect the group communication.

5. Removal of a Node from the Group

This section describes at a high level how a node can be removed from the group.

If the application requires forward security, the KDC SHALL generate new group keying material and securely distribute it to all the current group members but the leaving node, using the message format defined in Section 4.2. Application profiles may define alternative message formats.

5.1. Expired Authorization

If the AS provides Token introspection (see Section 5.7 of [I-D.ietf-ace-oauth-authz]), the KDC can optionally use and check whether:

- o the node is not authorized anymore;
- o the access token is still valid, upon its expiration.

Either case, once aware that a node is not authorized anymore, the KDC has to remove the unauthorized node from the list of group members, if the KDC keeps track of that.

5.2. Request to Leave the Group

A node can actively request to leave the group. In this case, the Client can send a request formatted as follows to the KDC, to abandon the group. The client MUST use the protected channel established with ACE, mentioned in Section 4.

To request to leave a group, the client MUST send a CoAP POST request to the endpoint in the KDC associated to the group to leave (same endpoint used in Section 4.1 for Key Distribution requests). The payload of this Leave Request is a CBOR map which MUST contain:

- o 'type', encoded as a CBOR int, with value 2 ("leave").
- o 'scope', with value the specific resource that the Client is authorized to access (i.e. group or topic identifier) and wants to leave, encoded as in Section 3.1. The 'role' field is omitted.

Note that the 'role' field is omitted since such a request should only be used to leave a group altogether. If the leaving node wants to be part of a group with fewer roles, it does not need to communicate that to the KDC, and can simply stop acting according to such roles.

If the Leave Request is such that the KDC cannot extract all the necessary information to understand and process it correctly (e.g. no 'scope' field present), the KDC MUST respond with a 4.00 (Bad Request) error message. Otherwise, the KDC MUST remove the leaving node from the list of group members, if the KDC keeps track of that.

Note that, after having left the group, a node may wish to join it again. Then, as long as the node is still authorized to join the group, i.e. it has a still valid access token, it can re-request to join the group directly to the KDC without needing to retrieve a new access token from the AS. This means that the KDC needs to keep track of nodes with valid access tokens, before deleting all information about the leaving node.

6. Retrieval of New or Updated Keying Material

A node stops using the group keying material upon its expiration, according to the 'exp' parameter specified in the retained COSE Key. Then, if it wants to continue participating in the group communication, the node has to request new updated keying material to the KDC. In this case, and depending on what part of the keying material is expired, the client may need to communicate to the KDC its need for that part to be renewed: for example, if the Client uses an individual key to protect outgoing traffic and has to renew it, the node may request a new one, or new input material to derive it, without renewing the whole group keying material.

The Client may perform the same request to the KDC also upon receiving messages from other group members without being able to retrieve the material to correctly decrypt them. This may be due to a previous update of the group keying material (rekeying) triggered by the KDC, that the Client was not able to receive or decrypt.

Note that policies can be set up so that the Client sends a request to the KDC only after a given number of unsuccessfully decrypted incoming messages. It is application dependent and pertaining to the particular message exchange (e.g. [I-D.ietf-core-oscore-groupcomm]) to set up policies that instruct clients to retain unsuccessfully decrypted messages and for how long, so that they can be decrypted after getting updated keying material, rather than just considered non valid messages to discard right away.

The same request could also be sent by the client without being triggered by a failed decryption of a message, if the client wants to confirm that it has the latest group keying material. If that is the case, the client will receive from the KDC the same group keying material it has in memory.

Note that the difference between the keying material renewal request and the keying material update request is that the first one triggers the KDC to produce new keying material for that node, while the second one only triggers distribution (the renewal might have happened independently, because of expiration). Once a node receives new individual keying material, other group members may need to use the update keying material request to retrieve it.

Alternatively, the re-distribution of keying material can be initiated by the KDC, which e.g.:

- o Can maintain an Observable resource to send notifications to Clients when the keying material is updated. Such a notification would have the same payload as the Key Re-Distribution Response defined in Section 6.2.
- o Can send the payload of the Key Re-Distribution Response as one or multiple multicast requests to the members of the group, using secure rekeying schemes such as [RFC2093][RFC2094][RFC2627].
- o Can send unicast requests to each Client over a secure channel, with the Key Re-Distribution Response as payload.
- o Can act as a publisher in a pub-sub scenario, and update the keying material by publishing on a specific topic on a broker, which all the members of the group are subscribed to.

Note that these methods of KDC-initiated key re-distribution have different security properties and require different security associations.

6.1. Key Re-Distribution Request

To request a re-distribution of keying material, the Client sends a shortened Key Distribution Request to the KDC (Section 4.1), formatted as follows. The payload MUST contain the following fields:

- o 'type', encoded as a CBOR int, with value 3 ("update key") if the request is intended to retrieve updated group keying material, and 4 ("new") if the request is intended for the KDC to produce and provide new individual keying material for the Client.
- o 'scope', which contains only the identifier of the specific group or topic, encoded as in Section 3.1. That is, the role field is not present.

6.2. Key Re-Distribution Response

The KDC receiving a Key Re-Distribution Request MUST check that it is storing a valid access token from that client for that scope.

If that is not the case, i.e. it does not store the token or the token is not valid for that client for the scope requested, the KDC MUST respond with a 4.01 (Unauthorized) error message. Analogously to Section 4.2, if the Key Re-Distribution Request is not formatted correctly (e.g. no 'scope' field present, or unknown fields present), the KDC MUST respond with a 4.00 (Bad Request) error message.

Otherwise, the KDC replies to the Client with a Key Distribution Response, which MUST include the 'kty', 'key' and 'exp' parameters specified in Section 4.2. The Key Distribution Response MAY also include the 'profile', 'group_policies' and 'mgt_key_material' parameters specified in Section 4.2.

Note that this response might simply re-provide the same keying material currently owned by the Client, if it has not been renewed.

7. Retrieval of Public Keys for Group Members

In case the KDC maintains the public keys of group members, a node in the group can contact the KDC to request public keys of either all group members or a specified subset, using the messages defined below.

Figure 7 gives an overview of the exchange described above.

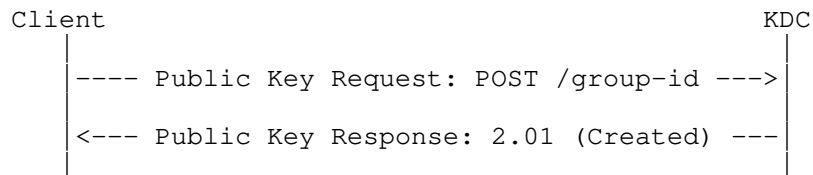


Figure 7: Message Flow of Public Key Request-Response

Note that these messages can be combined with the Key Re-Distribution messages in Section 6, to request at the same time the keying material and the public keys. In this case, either a new endpoint at the KDC may be used, or additional information needs to be sent in the request payload, to distinguish these combined messages from the Public Key messages described below, since they would be identical otherwise.

7.1. Public Key Request

To request public keys, the Client sends a shortened Key Distribution Request to the KDC (Section 4.1), formatted as follows. The payload of this request MUST contain the following fields:

- o 'type', encoded as a CBOR int, with value 5 ("pub keys").
- o 'get_pub_keys', which has as value a CBOR array including either:
 - * no elements, i.e. an empty array, in order to request the public key of all current group members; or
 - * N elements, each of which is the identifier of a group member encoded as a CBOR byte string, in order to request the public key of the specified nodes.
- o 'scope', which contains only the identifier of the specific group or topic, encoded as in Section 3.1. That is, the role field is not present.

7.2. Public Key Response

The KDC replies to the Client with a Key Distribution Response containing only the 'pub_keys' parameter, as specified in Section 4.2. The payload of this response contains the following field:

- o 'pub_keys', which contains either:
 - * the public keys of all the members of the group, if the 'get_pub_keys' parameter of the Public Key request was an empty array; or
 - * the public keys of the group members with the identifiers specified in the 'get_pub_keys' parameter of the Public Key request.

The KDC may enforce one of the following policies, in order to handle possible identifiers that are included in the 'get_pub_keys' parameter of the Public Key request but are not associated to any current group member.

- o The KDC silently ignores those identifiers.
- o The KDC retains public keys of group members for a given amount of time after their leaving, before discarding them. As long as such

public keys are retained, the KDC provides them to a requesting Client.

Either case, a node that has left the group should not expect any of its outgoing messages to be successfully processed, if received after its leaving, due to a possible group rekeying occurred before the message reception.

8. ACE Groupcomm Parameters

This specification defines a number of fields used during the message exchange. The table below summarizes them, and specifies the CBOR key to use instead of the full descriptive name.

| Name | CBOR Key | CBOR Type |
|--------------------|----------|----------------------------------|
| scope | TBD | array |
| get_pub_keys | TBD | array |
| client_cred | TBD | byte string |
| client_cred_verify | TBD | byte string |
| pub_keys_repos | TBD | array |
| kty | TBD | int / byte string |
| key | TBD | see "ACE Groupcomm Key" Registry |
| profile | TBD | int |
| exp | TBD | int / float |
| pub_keys | TBD | byte string |
| group_policies | TBD | map |
| mgt_key_material | TBD | byte string |
| type | TBD | int |

9. ACE Groupcomm Request Type

This specification defines a number of types of requests. The table below summarizes them.

| Name | Value |
|------------------|-------|
| key distribution | 1 |
| leave | 2 |
| update key | 3 |
| new | 4 |
| pub keys | 5 |

10. Security Considerations

When a Client receives a message from a sender for the first time, it needs to have a mechanism in place to avoid replay, e.g. Appendix B.2 of [I-D.ietf-core-object-security].

The KDC must renew the group keying material upon its expiration.

The KDC should renew the keying material upon group membership change, and should provide it to the current group members through the rekeying scheme used in the group.

The KDC may enforce a rekeying policy that takes into account the overall time required to rekey the group, as well as the expected rate of changes in the group membership.

That is, the KDC may not rekey the group at every membership change, for instance if members' joining and leaving occur frequently and performing a group rekeying takes too long. Instead, the KDC may rekey the group after a minum number of group members have joined or left within a given time interval, or during predictable network inactivity periods.

However, this would result in the KDC not constantly preserving backward and forward security. In fact, newly joining group members could be able to access the keying material used before their joining, and thus could access past group communications. Also, until the KDC performs a group rekeying, the newly leaving nodes would still be able to access upcoming group communications that are protected with the keying material that has not yet been updated.

10.1. Update of Keying Material

A group member can receive a message shortly after the group has been rekeyed, and new keying material has been distributed by the KDC. In the following two cases, this may result in misaligned keying material between the group members.

In the first case, the sender protects a message using the old keying material. However, the recipient receives the message after having received the new keying material, hence not being able to correctly process it. A possible way to ameliorate this issue is to preserve the old, recent, keying material for a maximum amount of time defined by the application. By doing so, the recipient can still try to process the received message using the old retained keying material as second attempt. Note that a former (compromised) group member can take advantage of this by sending messages protected with the old retained keying material. Therefore, a conservative application policy should not admit the storage of old keying material.

In the second case, the sender protects a message using the new keying material, but the recipient receives that request before having received the new keying material. Therefore, the recipient would not be able to correctly process the request and hence discards it. If the recipient receives the new keying material shortly after that and the sender endpoint uses CoAP retransmissions, the former will still be able to receive and correctly process the message. In any case, the recipient should actively ask the KDC for an updated keying material according to an application-defined policy, for instance after a given number of unsuccessfully decrypted incoming messages.

10.2. Block-Wise Considerations

If the block-wise options [RFC7959] are used, and the keying material is updated in the middle of a block-wise transfer, the sender of the blocks just changes the keying material to the updated one and continues the transfer. As long as both sides get the new keying material, updating the keying material in the middle of a transfer will not cause any issue. Otherwise, the sender will have to transmit the message again, when receiving an error message from the recipient.

Compared to a scenario where the transfer does not use block-wise, depending on how fast the keying material is changed, the nodes might consume a larger amount of the network resending the blocks again and again, which might be problematic.

11. IANA Considerations

This document has the following actions for IANA.

11.1. ACE Authorization Server Request Creation Hints Registry

IANA is asked to register the following entries in the "ACE Authorization Server Request Creation Hints" Registry defined in Section 8.1 of [I-D.ietf-ace-oauth-authz].

- o Name: sign_info
- o CBOR Key: TBD (range -256 to 255)
- o Value Type: any
- o Reference: [[This specification]]
- o Name: pub_key_enc
- o CBOR Key: TBD (range -256 to 255)
- o Value Type: integer
- o Reference: [[This specification]]

11.2. ACE Public Key Encoding Registry

This specification establishes the "ACE Public Key Encoding" IANA Registry. The Registry has been created to use the "Expert Review Required" registration procedure [RFC8126]. Expert review guidelines are provided in Section 11.9. It should be noted that, in addition to the expert review, some portions of the Registry require a specification, potentially a Standards Track RFC, be supplied as well.

The columns of this Registry are:

- o Name: This is a descriptive name that enables easier reference to the item. The name MUST be unique. It is not used in the encoding.
- o Value: The value to be used to identify this public key encoding. This value MUST be unique. The value can be a positive or a negative integer. Integer values between 0 and 255 are designated as Standards Track Document required. Integer values from 256 to 65535 are designated as Specification Required. Integer values of

greater than 65535 are designated as expert review. Integer values less than -65536 are marked as private use.

- o Description: This field contains a brief description for this public key encoding.
- o Reference: This field contains a pointer to the public specification providing the public key encoding, if one exists.

The value 0 is to be marked as "Reserved".

11.3. ACE Groupcomm Parameters Registry

This specification establishes the "ACE Groupcomm Parameters" IANA Registry. The Registry has been created to use the "Expert Review Required" registration procedure [RFC8126]. Expert review guidelines are provided in Section 11.9.

The columns of this Registry are:

- o Name: This is a descriptive name that enables easier reference to the item. The name MUST be unique. It is not used in the encoding.
- o CBOR Key: This is the value used as CBOR key of the item. These values MUST be unique. The value can be a positive integer, a negative integer, or a string.
- o CBOR Type: This contains the CBOR type of the item, or a pointer to the registry that defines its type, when that depends on another item.
- o Reference: This contains a pointer to the public specification for the format of the item, if one exists.

This Registry has been initially populated by the values in Section 8. The specification column for all of these entries will be this document.

11.4. Ace Groupcomm Request Type Registry

This specification establishes the "ACE Groupcomm Request Type" IANA Registry. The Registry has been created to use the "Expert Review Required" registration procedure [RFC8126]. Expert review guidelines are provided in Section 11.9.

The columns of this Registry are:

- o Name: This is a descriptive name that enables easier reference to the item. The name MUST be unique. It is not used in the encoding.
- o Value: This is the value used to identify the request. These values MUST be unique. The value must be a positive integer.
- o Reference: This contains a pointer to the public specification for the format of the item, if one exists.

This Registry has been initially populated by the values in Section 9. The reference column for all of these entries will be this document. The value 0 is to be marked as "Reserved".

11.5. ACE Groupcomm Key Registry

This specification establishes the "ACE Groupcomm Key" IANA Registry. The Registry has been created to use the "Expert Review Required" registration procedure [RFC8126]. Expert review guidelines are provided in Section 11.9.

The columns of this Registry are:

- o Name: This is a descriptive name that enables easier reference to the item. The name MUST be unique. It is not used in the encoding.
- o Key Type Value: This is the value used to identify the keying material. These values MUST be unique. The value can be a positive integer, a negative integer, or a string.
- o Profile: This field may contain one or more descriptive strings of application profiles to be used with this item. The values should be taken from the Name column of the "ACE Groupcomm Profile" Registry.
- o Description: This field contains a brief description of the keying material.
- o References: This contains a pointer to the public specification for the format of the keying material, if one exists.

This Registry has been initially populated by the values in Figure 5. The specification column for all of these entries will be this document.

11.6. ACE Groupcomm Profile Registry

This specification establishes the "ACE Groupcomm Profile" IANA Registry. The Registry has been created to use the "Expert Review Required" registration procedure [RFC8126]. Expert review guidelines are provided in Section 11.9. It should be noted that, in addition to the expert review, some portions of the Registry require a specification, potentially a Standards Track RFC, be supplied as well.

The columns of this Registry are:

- o Name: The name of the application profile, to be used as value of the profile attribute.
- o Description: Text giving an overview of the application profile and the context it is developed for.
- o CBOR Value: CBOR abbreviation for the name of this application profile. Different ranges of values use different registration policies [RFC8126]. Integer values from -256 to 255 are designated as Standards Action. Integer values from -65536 to -257 and from 256 to 65535 are designated as Specification Required. Integer values greater than 65535 are designated as Expert Review. Integer values less than -65536 are marked as Private Use.
- o Reference: This contains a pointer to the public specification of the abbreviation for this application profile, if one exists.

11.7. ACE Groupcomm Policy Registry

This specification establishes the "ACE Groupcomm Policy" IANA Registry. The Registry has been created to use the "Expert Review Required" registration procedure [RFC8126]. Expert review guidelines are provided in Section 11.9. It should be noted that, in addition to the expert review, some portions of the Registry require a specification, potentially a Standards Track RFC, be supplied as well.

The columns of this Registry are:

- o Name: The name of the group communication policy.
- o CBOR label: The value to be used to identify this group communication policy. Key map labels MUST be unique. The label can be a positive integer, a negative integer or a string. Integer values between 0 and 255 and strings of length 1 are

designated as Standards Track Document required. Integer values from 256 to 65535 and strings of length 2 are designated as Specification Required. Integer values of greater than 65535 and strings of length greater than 2 are designated as expert review. Integer values less than -65536 are marked as private use.

- o CBOR type: the CBOR type used to encode the value of this group communication policy.
- o Description: This field contains a brief description for this group communication policy.
- o Reference: This field contains a pointer to the public specification providing the format of the group communication policy, if one exists.

This registry will be initially populated by the values in Figure 6.

11.8. Sequence Number Synchronization Method Registry

This specification establishes the "Sequence Number Synchronization Method" IANA Registry. The Registry has been created to use the "Expert Review Required" registration procedure [RFC8126]. Expert review guidelines are provided in Section 11.9. It should be noted that, in addition to the expert review, some portions of the Registry require a specification, potentially a Standards Track RFC, be supplied as well.

The columns of this Registry are:

- o Name: The name of the sequence number synchronization method.
- o Value: The value to be used to identify this sequence number synchronization method.
- o Description: This field contains a brief description for this sequence number synchronization method.
- o Reference: This field contains a pointer to the public specification describing the sequence number synchronization method.

11.9. Expert Review Instructions

The IANA Registries established in this document are defined as expert review. This section gives some general guidelines for what the experts should be looking for, but they are being designated as experts for a reason so they should be given substantial latitude.

Expert reviewers should take into consideration the following points:

- o Point squatting should be discouraged. Reviewers are encouraged to get sufficient information for registration requests to ensure that the usage is not going to duplicate one that is already registered and that the point is likely to be used in deployments. The zones tagged as private use are intended for testing purposes and closed environments, code points in other ranges should not be assigned for testing.
- o Specifications are required for the standards track range of point assignment. Specifications should exist for specification required ranges, but early assignment before a specification is available is considered to be permissible. Specifications are needed for the first-come, first-serve range if they are expected to be used outside of closed environments in an interoperable way. When specifications are not provided, the description provided needs to have sufficient information to identify what the point is being used for.
- o Experts should take into account the expected usage of fields when approving point assignment. The fact that there is a range for standards track documents does not mean that a standards track document cannot have points assigned outside of that range. The length of the encoded value should be weighed against how many code points of that length are left, the size of device it will be used on, and the number of code points left that encode to that size.

12. References

12.1. Normative References

[I-D.ietf-ace-oauth-authz]

Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "Authentication and Authorization for Constrained Environments (ACE) using the OAuth 2.0 Framework (ACE-OAuth)", draft-ietf-ace-oauth-authz-24 (work in progress), March 2019.

[I-D.ietf-ace-oauth-params]

Seitz, L., "Additional OAuth Parameters for Authorization in Constrained Environments (ACE)", draft-ietf-ace-oauth-params-05 (work in progress), March 2019.

- [I-D.ietf-core-oscore-groupcomm]
Tiloca, M., Selander, G., Palombini, F., and J. Park,
"Group OSCORE - Secure Group Communication for CoAP",
draft-ietf-core-oscore-groupcomm-05 (work in progress),
July 2019.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate
Requirement Levels", BCP 14, RFC 2119,
DOI 10.17487/RFC2119, March 1997,
<<https://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, <<https://www.rfc-editor.org/info/rfc7049>>.
- [RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for
Writing an IANA Considerations Section in RFCs", BCP 26,
RFC 8126, DOI 10.17487/RFC8126, June 2017,
<<https://www.rfc-editor.org/info/rfc8126>>.
- [RFC8152] Schaad, J., "CBOR Object Signing and Encryption (COSE)",
RFC 8152, DOI 10.17487/RFC8152, July 2017,
<<https://www.rfc-editor.org/info/rfc8152>>.

12.2. Informative References

- [I-D.dijk-core-groupcomm-bis]
Dijk, E., Wang, C., and M. Tiloca, "Group Communication
for the Constrained Application Protocol (CoAP)", draft-
dijk-core-groupcomm-bis-00 (work in progress), March 2019.
- [I-D.ietf-ace-dtls-authorize]
Gerdes, S., Bergmann, O., Bormann, C., Selander, G., and
L. Seitz, "Datagram Transport Layer Security (DTLS)
Profile for Authentication and Authorization for
Constrained Environments (ACE)", draft-ietf-ace-dtls-
authorize-08 (work in progress), April 2019.
- [I-D.ietf-ace-mqtt-tls-profile]
Sengul, C., Kirby, A., and P. Fremantle, "MQTT-TLS profile
of ACE", draft-ietf-ace-mqtt-tls-profile-00 (work in
progress), May 2019.
- [I-D.ietf-ace-oscore-profile]
Palombini, F., Seitz, L., Selander, G., and M. Gunnarsson,
"OSCORE profile of the Authentication and Authorization
for Constrained Environments Framework", draft-ietf-ace-
oscore-profile-07 (work in progress), February 2019.

- [I-D.ietf-core-coap-pubsub]
Koster, M., Keranen, A., and J. Jimenez, "Publish-Subscribe Broker for the Constrained Application Protocol (CoAP)", draft-ietf-core-coap-pubsub-08 (work in progress), March 2019.
- [I-D.ietf-core-object-security]
Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security for Constrained RESTful Environments (OSCORE)", draft-ietf-core-object-security-16 (work in progress), March 2019.
- [RFC2093] Harney, H. and C. Muckenhirn, "Group Key Management Protocol (GKMP) Specification", RFC 2093, DOI 10.17487/RFC2093, July 1997, <<https://www.rfc-editor.org/info/rfc2093>>.
- [RFC2094] Harney, H. and C. Muckenhirn, "Group Key Management Protocol (GKMP) Architecture", RFC 2094, DOI 10.17487/RFC2094, July 1997, <<https://www.rfc-editor.org/info/rfc2094>>.
- [RFC2627] Wallner, D., Harder, E., and R. Agee, "Key Management for Multicast: Issues and Architectures", RFC 2627, DOI 10.17487/RFC2627, June 1999, <<https://www.rfc-editor.org/info/rfc2627>>.
- [RFC7390] Rahman, A., Ed. and E. Dijk, Ed., "Group Communication for the Constrained Application Protocol (CoAP)", RFC 7390, DOI 10.17487/RFC7390, October 2014, <<https://www.rfc-editor.org/info/rfc7390>>.
- [RFC7519] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", RFC 7519, DOI 10.17487/RFC7519, May 2015, <<https://www.rfc-editor.org/info/rfc7519>>.
- [RFC7959] Bormann, C. and Z. Shelby, Ed., "Block-Wise Transfers in the Constrained Application Protocol (CoAP)", RFC 7959, DOI 10.17487/RFC7959, August 2016, <<https://www.rfc-editor.org/info/rfc7959>>.
- [RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <<https://www.rfc-editor.org/info/rfc8259>>.

Appendix A. Requirements on Application Profiles

This section lists the requirements on application profiles of this specification, for the convenience of application profile designers.

- o Specify the communication protocol the members of the group must use (e.g., multicast CoAP).
- o Specify the security protocol the group members must use to protect their communication (e.g., group OSCORE). This must provide encryption, integrity and replay protection.
- o Specify the encoding and value of the identifier of group or topic and role of 'scope' (see Section 3.1).
- o Specify and register the application profile identifier (see Section 4.1).
- o Specify the acceptable values of 'kty' (see Section 4.2).
- o Specify the format and content of 'group_policies' entries (see Section 4.2).
- o Optionally, specify the format and content of 'mgt_key_material' (see Section 4.2).
- o Optionally, specify transport profile of ACE [I-D.ietf-ace-oauth-authz] to use between Client and KDC.
- o Optionally, specify the encoding of public keys, of 'client_cred', and of 'pub_keys' if COSE_Keys are not used (see Section 4.2).
- o Optionally, specify the acceptable values for parameters related to signature algorithm and signature keys: 'sign_alg', 'sign_parameters', 'sign_key_parameters', 'pub_key_enc' (see Section 3.3).
- o Optionally, specify the negotiation of parameter values for signature algorithm and signature keys, if 'sign_info' and 'pub_key_enc' are not used (see Section 3.3).

Appendix B. Document Updates

RFC EDITOR: PLEASE REMOVE THIS SECTION.

B.1. Version -01 to -02

- o Editorial fixes.
- o Distinction between transport profile and application profile (Section 1.1).
- o New parameters 'sign_info' and 'pub_key_enc' to negotiate parameter values for signature algorithm and signature keys (Section 3.3).
- o New parameter 'type' to distinguish different Key Distribution Request messages (Section 4.1).
- o New parameter 'client_cred_verify' in the Key Distribution Request to convey a Client signature (Section 4.1).
- o Encoding of 'pub_keys_repos' (Section 4.1).
- o Encoding of 'mgt_key_material' (Section 4.1).
- o Improved description on retrieval of new or updated keying material (Section 6).
- o Encoding of 'get_pub_keys' in Public Key Request (Section 7.1).
- o Extended security considerations (Sections 10.1 and 10.2).
- o New "ACE Public Key Encoding" IANA Registry (Section 11.2).
- o New "ACE Groupcomm Parameters" IANA Registry (Section 11.3), populated with the entries in Section 8.
- o New "Ace Groupcomm Request Type" IANA Registry (Section 11.4), populated with the values in Section 9.
- o New "ACE Groupcomm Policy" IANA Registry (Section 11.7) populated with two entries "Sequence Number Synchronization Method" and "Key Update Check Interval" (Section 4.2).
- o Improved list of requirements for application profiles (Appendix A).

B.2. Version -00 to -01

- o Changed name of 'req_aud' to 'audience' in the Authorization Request (Section 3.1).

- o Defined error handling on the KDC (Sections 4.2 and 6.2).
- o Updated format of the Key Distribution Response as a whole (Section 4.2).
- o Generalized format of 'pub_keys' in the Key Distribution Response (Section 4.2).
- o Defined format for the message to request leaving the group (Section 5.2).
- o Renewal of individual keying material and methods for group rekeying initiated by the KDC (Section 6).
- o CBOR type for node identifiers in 'get_pub_keys' (Section 7.1).
- o Added section on parameter identifiers and their CBOR keys (Section 8).
- o Added request types for requests to a Join Response (Section 9).
- o Extended security considerations (Section 10).
- o New IANA registries "ACE Groupcomm Key Registry", "ACE Groupcomm Profile Registry", "ACE Groupcomm Policy Registry" and "Sequence Number Synchronization Method Registry" (Section 11).
- o Added appendix about requirements for application profiles of ACE on group communication (Appendix A).

Acknowledgments

The following individuals were helpful in shaping this document: Ben Kaduk, John Mattsson, Jim Schaad, Ludwig Seitz, Goeran Selander and Peter van der Stok.

The work on this document has been partly supported by VINNOVA and the Celtic-Next project CRITISEC; and by the EIT-Digital High Impact Initiative ACTIVE.

Authors' Addresses

Francesca Palombini
Ericsson AB
Torshamnsgatan 23
Kista SE-16440 Stockholm
Sweden

Email: francesca.palombini@ericsson.com

Marco Tiloca
RISE AB
Isafjordsgatan 22
Kista SE-16440 Stockholm
Sweden

Email: marco.tiloca@ri.se

ACE Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 6, 2020

M. Tiloca
RISE AB
J. Park
Universitaet Duisburg-Essen
F. Palombini
Ericsson AB
July 05, 2019

Key Management for OSCORE Groups in ACE
draft-ietf-ace-key-groupcomm-oscore-02

Abstract

This document describes a method to request and provision keying material in group communication scenarios where the group communication is based on CoAP and secured with Object Security for Constrained RESTful Environments (OSCORE). The proposed method delegates the authentication and authorization of new client nodes that join an OSCORE group through a Group Manager server. This approach builds on the ACE framework for Authentication and Authorization, and leverages protocol-specific transport profiles of ACE to achieve communication security, proof-of-possession and server authentication.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on January 6, 2020.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

| | |
|--|----|
| 1. Introduction | 3 |
| 1.1. Terminology | 4 |
| 1.2. Relation to Other Documents | 5 |
| 2. Protocol Overview | 6 |
| 2.1. Overview of the Join Process | 7 |
| 2.2. Overview of the Group Rekeying Process | 8 |
| 3. Joining Node to Authorization Server | 9 |
| 3.1. Authorization Request | 9 |
| 3.2. Authorization Response | 10 |
| 4. Joining Node to Group Manager | 11 |
| 4.1. Token Post | 11 |
| 4.2. Join Request | 12 |
| 4.3. Join Response | 13 |
| 5. Leaving of a Group Member | 17 |
| 6. Public Keys of Joining Nodes | 18 |
| 7. Group Rekeying Process | 20 |
| 8. Security Considerations | 21 |
| 9. IANA Considerations | 22 |
| 9.1. ACE Groupcomm Key Registry | 22 |
| 9.2. OSCORE Security Context Parameters Registry | 23 |
| 9.3. ACE Groupcomm Profile Registry | 24 |
| 9.4. Sequence Number Synchronization Method Registry | 24 |
| 9.5. ACE Public Key Encoding Registry | 25 |
| 10. References | 25 |
| 10.1. Normative References | 25 |
| 10.2. Informative References | 26 |
| Appendix A. Profile Requirements | 27 |
| Appendix B. Document Updates | 28 |
| B.1. Version -01 to -02 | 28 |
| B.2. Version -00 to -01 | 29 |
| Acknowledgments | 29 |
| Authors' Addresses | 29 |

1. Introduction

Object Security for Constrained RESTful Environments (OSCORE) [I-D.ietf-core-object-security] is a method for application-layer protection of the Constrained Application Protocol (CoAP) [RFC7252], using CBOR Object Signing and Encryption (COSE) [RFC8152] and enabling end-to-end security of CoAP payload and options.

As described in [I-D.ietf-core-oscore-groupcomm], OSCORE may be used to protect CoAP group communication over IP multicast [RFC7390][I-D.dijk-core-groupcomm-bis]. This relies on a Group Manager, which is responsible for managing an OSCORE group, where members exchange CoAP messages secured with OSCORE. The Group Manager can be responsible for multiple groups, coordinates the join process of new group members, and is entrusted with the distribution and renewal of group keying material.

This specification builds on the ACE framework for Authentication and Authorization [I-D.ietf-ace-oauth-authz] and defines a method to:

- o Authorize a node to join an OSCORE group, and provide it with the group keying material to communicate with other group members.
- o Provide updated keying material to group members upon request.
- o Renew the group keying material and distribute it to the OSCORE group (rekeying) upon changes in the group membership.

A client node joins an OSCORE group through a resource server acting as Group Manager for that group. The join process relies on an Access Token, which is bound to a proof-of-possession key and authorizes the client to access a specific join resource at the Group Manager.

Messages exchanged among the participants follow the formats defined in [I-D.ietf-ace-key-groupcomm] for provisioning and renewing keying material in group communication scenarios.

In order to achieve communication security, proof-of-possession and server authentication, the client and the Group Manager leverage protocol-specific transport profiles of ACE. These include also possible forthcoming transport profiles that comply with the requirements in Appendix C of [I-D.ietf-ace-oauth-authz].

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 BCP 14 [RFC2119][RFC8174] when, and only when, they appear in all capitals, as shown here.

Readers are expected to be familiar with the terms and concepts described in the ACE framework for authentication and authorization [I-D.ietf-ace-oauth-authz]. The terminology for entities in the considered architecture is defined in OAuth 2.0 [RFC6749]. In particular, this includes Client (C), Resource Server (RS), and Authorization Server (AS).

Readers are expected to be familiar with the terms and concepts related to the CoAP protocol described in [RFC7252][RFC7390][I-D.dijk-core-groupcomm-bis]. Note that, unless otherwise indicated, the term "endpoint" is used here following its OAuth definition, aimed at denoting resources such as /token and /introspect at the AS and /authz-info at the RS. This document does not use the CoAP definition of "endpoint", which is "An entity participating in the CoAP protocol".

Readers are expected to be familiar with the terms and concepts for protection and processing of CoAP messages through OSCORE [I-D.ietf-core-object-security] also in group communication scenarios [I-D.ietf-core-oscore-groupcomm]. These include the concept of Group Manager, as the entity responsible for a set of groups where communications are secured with OSCORE. In this specification, the Group Manager acts as Resource Server.

This document refers also to the following terminology.

- o Joining node: a network node intending to join an OSCORE group, where communication is based on CoAP [RFC7390][I-D.dijk-core-groupcomm-bis] and secured with OSCORE as described in [I-D.ietf-core-oscore-groupcomm].
- o Join process: the process through which a joining node becomes a member of an OSCORE group. The join process is enforced and assisted by the Group Manager responsible for that group.
- o Join resource: a resource hosted by the Group Manager, associated to an OSCORE group under that Group Manager. A join resource is identifiable with the Group Identifier (Gid) of the respective group. A joining node accesses a join resource to start the join

process and become a member of that group. The URI of a join resource is fixed.

- o Join endpoint: an endpoint at the Group Manager associated to a join resource.
- o Requester: member of an OSCORE group that sends request messages to other members of the group.
- o Responder: member of an OSCORE group that receives request messages from other members of the group. A responder may reply back, by sending a response message to the requester which has sent the request message.
- o Monitor: member of a group that is configured as responder and never replies back to requesters after receiving request messages. This corresponds to the term "silent server" used in [I-D.ietf-core-oscore-groupcomm].
- o Group rekeying process: the process through which the Group Manager renews the security parameters and group keying material, and (re-)distributes them to the OSCORE group members.

1.2. Relation to Other Documents

Figure 1 overviews the main documents related to this specification. Arrows and asterisk-arrows denote normative references and informative references, respectively.

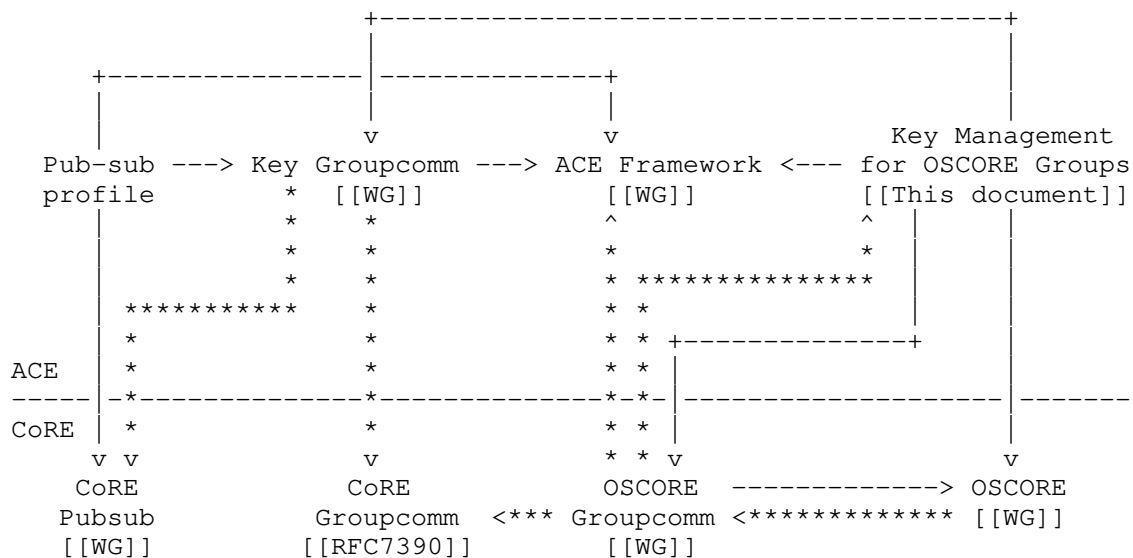


Figure 1: Related Documents

2. Protocol Overview

Group communication for CoAP over IP multicast has been enabled in [RFC7390][I-D.dijk-core-groupcomm-bis] and can be secured with Object Security for Constrained RESTful Environments (OSCORE) [I-D.ietf-core-object-security] as described in [I-D.ietf-core-oscore-groupcomm]. A network node joins an OSCORE group by interacting with the responsible Group Manager. Once registered in the group, the new node can securely exchange messages with other group members.

This specification describes how to use the ACE framework for authentication and authorization [I-D.ietf-ace-oauth-authz] to:

- o Enable a node to join an OSCORE group through the Group Manager and receive the security parameters and keying material to communicate with the other members of the group.
- o Enable members of OSCORE groups to retrieve updated group keying material from the Group Manager.
- o Enable the Group Manager to renew the security parameters and group keying material, and to (re-)distribute them to the members of the OSCORE group (rekeying).

With reference to the ACE framework and the terminology defined in OAuth 2.0 [RFC6749]:

- o The Group Manager acts as Resource Server (RS), and hosts one join resource for each OSCORE group it manages. Each join resource is exported by a distinct join endpoint. During the join process, the Group Manager provides joining nodes with the parameters and keying material for taking part to secure communications in the OSCORE group. The Group Manager also maintains the group keying material and performs the group rekeying process to distribute updated keying material to the group members.
- o The joining node acts as Client (C), and requests to join an OSCORE group by accessing the related join endpoint at the Group Manager.
- o The Authorization Server (AS) authorizes joining nodes to join OSCORE groups under their respective Group Manager. Multiple Group Managers can be associated to the same AS. The AS MAY release Access Tokens for other purposes than joining OSCORE groups under registered Group Managers. For example, the AS may also release Access Tokens for accessing resources hosted by members of OSCORE groups.

All communications between the involved entities rely on the CoAP protocol and MUST be secured.

In particular, communications between the joining node and the Group Manager leverage protocol-specific transport profiles of ACE to achieve communication security, proof-of-possession and server authentication. To this end, the AS must signal the specific transport profile to use, consistently with requirements and assumptions defined in the ACE framework [I-D.ietf-ace-oauth-authz].

With reference to the AS, communications between the joining node and the AS (/token endpoint) as well as between the Group Manager and the AS (/introspect endpoint) can be secured by different means, for instance using DTLS [RFC6347] or OSCORE [I-D.ietf-core-object-security]. Further details on how the AS secures communications (with the joining node and the Group Manager) depend on the specifically used transport profile of ACE, and are out of the scope of this specification.

2.1. Overview of the Join Process

A node performs the following steps in order to join an OSCORE group. Messages exchanged among the participants follow the formats defined in [I-D.ietf-ace-key-groupcomm], and are further specified in

Section 3 and Section 4 of this document. The Group Manager acts as the Key Distribution Center (KDC) defined in [I-D.ietf-ace-key-groupcomm].

1. The joining node requests an Access Token from the AS, in order to access a join resource on the Group Manager and hence join the associated OSCORE group (see Section 3). The joining node will start or continue using a secure communication channel with the Group Manager, according to the response from the AS.
2. The joining node transfers authentication and authorization information to the Group Manager by posting the obtained Access Token (see Section 4). After that, a joining node must have a secure communication channel established with the Group Manager, before starting to join an OSCORE group under that Group Manager (see Section 4). Possible ways to provide a secure communication channel are DTLS [RFC6347] and OSCORE [I-D.ietf-core-object-security].
3. The joining node starts the join process to become a member of the OSCORE group, by accessing the related join resource hosted by the Group Manager (see Section 4).
4. At the end of the join process, the joining node has received from the Group Manager the parameters and keying material to securely communicate with the other members of the OSCORE group.
5. The joining node and the Group Manager maintain the secure channel, to support possible future communications.

All further communications between the joining node and the Group Manager MUST be secured, for instance with the same secure channel mentioned in step 2.

2.2. Overview of the Group Rekeying Process

If the application requires backward and forward security, the Group Manager MUST generate new security parameters and group keying material, and distribute them to the group (rekeying) upon membership changes.

That is, the group is rekeyed when a node joins the group as a new member, or after a current member leaves the group. By doing so, a joining node cannot access communications in the group prior its joining, while a leaving node cannot access communications in the group after its leaving.

Parameters and keying material include a new Group Identifier (Gid) for the group and a new Master Secret for the OSCORE Common Security Context of that group (see Section 2 of [I-D.ietf-core-oscore-groupcomm]).

The Group Manager MUST support the Group Rekeying Process described in Section 7. Future application profiles may define alternative message formats and distribution schemes to perform group rekeying.

3. Joining Node to Authorization Server

This section describes how the joining node interacts with the AS in order to be authorized to join an OSCORE group under a given Group Manager. In particular, it considers a joining node that intends to contact that Group Manager for the first time.

The message exchange between the joining node and the AS consists of the messages Authorization Request and Authorization Response defined in Section 3 of [I-D.ietf-ace-key-groupcomm].

In case the specific AS associated to the Group Manager is unknown to the joining node, the latter can rely on mechanisms like the Unauthorized Resource Request message described in Section 5.1.1 of [I-D.ietf-ace-oauth-authz] to discover the correct AS to contact.

3.1. Authorization Request

The joining node contacts the AS, in order to request an Access Token for accessing the join resource hosted by the Group Manager and associated to the OSCORE group. The Access Token request sent to the /token endpoint follows the format of the Authorization Request message defined in Section 3.1 of [I-D.ietf-ace-key-groupcomm]. In particular:

- o The 'scope' parameter MUST be present and MUST include:
 - * in the first element, either the Group Identifier (Gid) of the group to join under the Group Manager, or a value from which the Group Manager can derive the Gid of the group to join. It is up to the application to define how the Group Manager possibly performs the derivation of the full Gid. Appendix C of [I-D.ietf-core-oscore-groupcomm] provides an example of structured Gid, composed of a fixed part, namely Group Prefix, and a variable part, namely Group Epoch.
 - * in the second element, the role (encoded as a text string) or CBOR array of roles that the joining node intends to have in the group it intends to join. Accepted values of roles are:

"requester", "responder", and "monitor". Possible combinations are: ["requester" , "responder"]; ["requester" , "monitor"].

- o The 'audience' parameter MUST be present and is set to the identifier of the Group Manager.

3.2. Authorization Response

The AS is responsible for authorizing the joining node to join specific OSCORE groups, according to join policies enforced on behalf of the respective Group Manager.

In case of successful authorization, the AS releases an Access Token bound to a proof-of-possession key associated to the joining node.

Then, the AS provides the joining node with the Access Token as part of an Access Token response, which follows the format of the Authorization Response message defined in Section 3.2 of [I-D.ietf-ace-key-groupcomm].

The 'exp' parameter MUST be present. Other means for the AS to specify the lifetime of Access Tokens are out of the scope of this specification.

The AS must include the 'scope' parameter in the response when the value included in the Access Token differs from the one specified by the joining node in the request. In such a case, the second element of 'scope' MUST be present and includes the role or CBOR array of roles that the joining node is actually authorized to take in the group, encoded as specified in Section 3.1 of this document.

Also, the 'profile' parameter indicates the specific transport profile of ACE to use for securing communications between the joining node and the Group Manager (see Section 5.6.4.3 of [I-D.ietf-ace-oauth-authz]).

In particular, if symmetric keys are used, the AS generates a proof-of-possession key, binds it to the Access Token, and provides it to the joining node in the 'cnf' parameter of the Access Token response. Instead, if asymmetric keys are used, the joining node provides its own public key to the AS in the 'req_cnf' parameter of the Access Token request. Then, the AS uses it as proof-of-possession key bound to the Access Token, and provides the joining node with the Group Manager's public key in the 'rs_cnf' parameter of the Access Token response.

4. Joining Node to Group Manager

The following subsections describe the interactions between the joining node and the Group Manager, i.e. the Access Token post and the Request-Response exchange to join the OSCORE group.

4.1. Token Post

The joining node posts the Access Token to the /authz-info endpoint at the Group Manager, according to the Token post defined in Section 3.3 of [I-D.ietf-ace-key-groupcomm].

At this point in time, the joining node might not have all the necessary information concerning the public keys in the OSCORE group, as well as concerning the algorithm and related parameters for computing countersignatures in the OSCORE group. In such a case, the joining node MAY use the 'sign_info' and 'pub_key_enc' parameters defined in Section 3.3 of [I-D.ietf-ace-key-groupcomm] to ask for such information.

Alternatively, the joining node may retrieve this information by other means, e.g. by using the approach described in [I-D.tiloca-core-oscore-discovery].

If the Access Token is valid, the Group Manager responds to the POST request with a 2.01 (Created) response, according to what is specified in the signalled transport profile of ACE. The Group Manager MUST use the Content-Format "application/ace+cbor" defined in Section 8.14 of [I-D.ietf-ace-oauth-authz].

The payload of the 2.01 (Created) response is a CBOR map, which MUST include the 'cnonce' parameter defined in section 5.1.2 of [I-D.ietf-ace-oauth-authz], and MAY include the 'sign_info' parameter as well as the 'pub_key_enc' parameter.

The 'cnonce' parameter includes a nonce N generated by the Group Manager. The joining node may use this nonce in order to prove the possession of its own private key, upon joining the group (see Section 4.2).

If present in the response:

- o 'sign_alg', i.e. the first element of the 'sign_info' parameter, takes value from Tables 5 and 6 of [RFC8152].
- o 'sign_parameters', i.e. the second element of the 'sign_info' parameter, takes values from the "Counter Signature Parameters" Registry (see Section 9.1 of [I-D.ietf-core-oscore-groupcomm]).

Its structure depends on the value of 'sign_alg'. If no parameters of the counter signature algorithm are specified, 'sign_parameters' MUST be encoding the CBOR simple value Null.

- o 'sign_key_parameters', i.e. the third element of the 'sign_info' parameter, takes values from the "Counter Signature Key Parameters" Registry (see Section 9.2 of [I-D.ietf-core-oscore-groupcomm]). Its structure depends on the value of 'sign_alg'. If no parameters of the key used with the counter signature algorithm are specified, 'sign_key_parameters' MUST be encoding the CBOR simple value Null.
- o 'pub_key_enc' takes value from Figure 2, as a public key encoding in the "ACE Public Key Encoding" Registry (see Section 11.2 of [I-D.ietf-ace-key-groupcomm]).

| Name | Value | Description | Reference |
|----------|-------|--------------------------------|-------------|
| COSE_Key | 1 | Public key encoded as COSE Key | {{RFC8152}} |

Figure 2: ACE Public Key Encoding Values

Note that the CBOR map specified as payload of the 2.01 (Created) response may include further parameters, e.g. according to the signalled transport profile of ACE.

Finally, the joining node establishes a secure channel with the Group Manager, according to what is specified in the Access Token response and the signalled transport profile of ACE.

4.2. Join Request

Once a secure communication channel with the Group Manager has been established, the joining node requests to join the OSCORE group, by accessing the related join resource at the Group Manager.

In particular, the joining node sends to the Group Manager a confirmable CoAP request, using the method POST and targeting the join endpoint associated to that group. This Join Request follows the format and processing of the Key Distribution Request message defined in Section 4.1 of [I-D.ietf-ace-key-groupcomm]. In particular:

- o The 'type' parameter is set to 1 ("key distribution").

- o The 'get_pub_keys' parameter is present only if the joining node wants to retrieve the public keys of the group members from the Group Manager during the join process (see Section 6). Otherwise, this parameter MUST NOT be present.
- o The 'client_cred' parameter, if present, includes the public key of the joining node. In case the joining node knows the encoding of public keys in the OSCORE group, as well as the countersignature algorithm and possible associated parameters used in the OSCORE group, the included public key MUST be in a consistent format. This parameter MAY be omitted if: i) the joining node is asking to access the group exclusively as monitor; or ii) the Group Manager already acquired this information, for instance during a past join process. In any other case, this parameter MUST be present.

Furthermore, the CBOR map specified as payload of the Join Request MAY also include the following additional parameter, which MUST be present if the 'client_cred' parameter is present.

- o The 'client_cred_verify' parameter, which is encoded as a CBOR byte string and contains a signature computed by the joining node, in order to prove possession of its own private key. The signature is computed over the nonce N received in the 2.01 (Created) response to the Token POST (see Section 4.1). In particular, the joining node MUST use the COSE_CounterSignature0 object [RFC8152], with the Sig_structure containing the nonce N as payload; and an empty external_aad. The joining node computes the signature by using the same private key and countersignature algorithm it intends to use for signing messages in the OSCORE group.

4.3. Join Response

The Group Manager processes the Join Request according to [I-D.ietf-ace-oauth-authz] and Section 4.2 of [I-D.ietf-ace-key-groupcomm]. Also, the Group Manager MUST return a 4.00 (Bad Request) response in case the Join Request includes the 'client_cred' parameter but does not include the 'client_cred_verify' parameter.

If the request processing yields a positive outcome, the Group Manager performs the further following checks.

- o In case the Join Request includes the 'client_cred' parameter, the Group Manager checks that the public key of the joining node has an accepted format. That is, the public key has to be encoded as expected in the OSCORE group, and has to be consistent with the

counter signature algorithm and possible associated parameters used in the OSCORE group. The join process fails if the public key of the joining node does not have an accepted format.

- o In case the Join Request does not include the 'client_cred' parameter, the Group Manager checks whether it is storing a public key for the joining node, which is consistent with the encoding, counter signature algorithm and possible associated parameters used in the OSCORE group. The join process fails if the Group Manager either: i) does not store a public key with an accepted format for the joining node; or ii) stores multiple public keys with an accepted format for the joining node.
- o In case the Join Request includes the 'client_cred_verify' parameter, the Group Manager verifies the signature contained in the parameter. To this end, it considers: i) as signed value, the nonce N previously provided in the 2.01 (Created) response to the Token POST (see Section 4.1); ii) the countersignature algorithm used in the OSCORE group; and iii) the public key of the joining node, either retrieved from the 'client_cred' parameter, or as stored from a past join process. The join process fails if the Group Manager does not successfully verify the signature.

If the join process has failed, the Group Manager MUST reply to the joining node with a 4.00 (Bad Request) response. The payload of this response is a CBOR map, which includes a 'sign_info' parameter and a 'pub_key_enc' parameter, formatted as in the Token POST response in Section 4.1.

Upon receiving this response, the joining node SHOULD send a new Join Request to the Group Manager, which contains:

- o The 'client_cred' parameter, including a public key in a format consistent with the encoding, countersignature algorithm and possible associated parameters indicated by the Group Manager.
- o The 'client_cred_verify' parameter, including a signature computed as described in Section 4.2, by using the public key indicated in the current 'client_cred' parameter, with the countersignature algorithm and possible associated parameters indicated by the Group Manager.

Otherwise, in case of success, the Group Manager updates the group membership by registering the joining node as a new member of the OSCORE group.

Then, the Group Manager replies to the joining node providing the updated security parameters and keying material necessary to

participate in the group communication. This Join Response follows the format and processing of the Key Distribution success Response message defined in Section 4.2 of [I-D.ietf-ace-key-groupcomm]. In particular:

- o The 'kty' parameter identifies a key of type "Group_OSCORE_Security_Context object", defined in Section 9.1 of this specification.
- o The 'key' parameter includes what the joining node needs in order to set up the OSCORE Security Context as per Section 2 of [I-D.ietf-core-oscore-groupcomm]. This parameter has as value a Group_OSCORE_Security_Context object, which is defined in this specification and extends the OSCORE_Security_Context object encoded in CBOR as defined in Section 3.2.1 of [I-D.ietf-ace-oscore-profile]. In particular, it contains the additional parameters 'cs_alg', 'cs_params', 'cs_key_params' and 'cs_key_enc' defined in Section 9.2 of this specification. More specifically, the 'key' parameter is composed as follows.
 - * The 'ms' parameter MUST be present and includes the OSCORE Master Secret value.
 - * The 'clientId' parameter, if present, has as value the OSCORE Sender ID assigned to the joining node by the Group Manager. This parameter is not present if the node joins the group exclusively as monitor, according to what specified in the Access Token (see Section 3.2). In any other case, this parameter MUST be present.
 - * The 'hkdf' parameter, if present, has as value the KDF algorithm used in the group.
 - * The 'alg' parameter, if present, has as value the AEAD algorithm used in the group.
 - * The 'salt' parameter, if present, has as value the OSCORE Master Salt.
 - * The 'contextId' parameter MUST be present and has as value the Group Identifier (Gid) associated to the OSCORE group.
 - * The 'rpl' parameter, if present, specifies the OSCORE Replay Window Size and Type value.
 - * The 'cs_alg' parameter MUST be present and specifies the algorithm used to countersign messages in the group. This parameter takes values from Tables 5 and 6 of [RFC8152].

- * The 'cs_params' parameter MAY be present and specifies the additional parameters for the counter signature algorithm. This parameter is a CBOR map whose content depends on the counter signature algorithm, as specified in Section 2 and Section 9.1 of [I-D.ietf-core-oscore-groupcomm].
- * The 'cs_key_params' parameter MAY be present and specifies the additional parameters for the key used with the counter signature algorithm. This parameter is a CBOR map whose content depends on the counter signature algorithm, as specified in Section 2 and Section 9.2 of [I-D.ietf-core-oscore-groupcomm].
- * The 'cs_key_enc' parameter MAY be present and specifies the encoding of the public keys of the group members. This parameter is a CBOR integer, whose value is taken from Figure 2, as a public key encoding in the "ACE Public Key Encoding" Registry (see Section 11.2 of [I-D.ietf-ace-key-groupcomm]). If this parameter is not present, COSE_Key (1) MUST be assumed as default value.
- o The 'profile' parameter MUST be present and has value coap_group_oscore_app (TBD), which is defined in Section 9.3 of this specification.
- o The 'exp' parameter MUST be present and specifies the expiration time in seconds after which the OSCORE Security Context derived from the 'key' parameter is not valid anymore.
- o The 'pub_keys' parameter is present only if the 'get_pub_keys' parameter was present in the Join Request. If present, this parameter includes the public keys of the group members that are relevant to the joining node. That is, it includes: i) the public keys of the responders currently in the group, in case the joining node is configured (also) as requester; and ii) the public keys of the requesters currently in the group, in case the joining node is configured (also) as responder or monitor.
- o The 'group_policies' parameter SHOULD be present and includes a list of parameters indicating particular policies enforced in the group. For instance, its field "Sequence Number Synchronization Method" can indicate the method to achieve synchronization of sequence numbers among group members (see Appendix E of [I-D.ietf-core-oscore-groupcomm]), as indicated by the corresponding value from the "Sequence Number Synchronization Method" Registry defined in Section 11.8 of [I-D.ietf-ace-key-groupcomm].

Finally, the joining node uses the information received in the Join Response to set up the OSCORE Security Context, as described in Section 2 of [I-D.ietf-core-oscore-groupcomm]. From then on, the joining node can exchange group messages secured with OSCORE as described in [I-D.ietf-core-oscore-groupcomm].

If the application requires backward security, the Group Manager SHALL generate updated security parameters and group keying material, and provide it to all the current group members (see Section 7).

When the OSCORE Security Context expires, as specified by the 'exp' parameter of the Join Response, the node considers it invalid and to be renewed. Then, the node retrieves updated security parameters and keying material, by exchanging with the Group Manager a shortened Join Request sent to the same Join Resource with the 'type' parameter set to 3 ("update key") and a shortened Join Response message, according to the approach defined in Section 6 of [I-D.ietf-ace-key-groupcomm]. Finally, the node uses the updated security parameters and keying material to set up the new OSCORE Security Context as described in Section 2 of [I-D.ietf-core-oscore-groupcomm].

Furthermore, as discussed in Section 2.2 of [I-D.ietf-core-oscore-groupcomm], the node may at some point experience a wrap-around of its own Sender Sequence Number in the group. When this happens, the node MUST send to the Group Manager a shortened Join Request message to the same Join Resource, with the 'type' parameter set to 4 ("new"). Upon receiving this request message, the Group Manager either rekeys the whole OSCORE group as discussed in Section 7, or generates a new Sender ID for that node and replies with a shortened Join Response message where:

- o Only the parameters 'type', 'kty', 'key', 'profile' and 'exp' are present.
- o The 'clientId' parameter of the 'key' parameter specifies the new Sender ID of the node.

5. Leaving of a Group Member

A node may be removed from the OSCORE group, due to expired or revoked authorization, or after its own request to the Group Manager.

If the application requires forward security, the Group Manager SHALL generate updated security parameters and group keying material, and provide it to the remaining group members (see Section 7). The leaving node must not be able to acquire the new security parameters and group keying material distributed after its leaving.

Same considerations in Section 5 of [I-D.ietf-ace-key-groupcomm] apply here as well, considering the Group Manager acting as KDC. In particular, a node requests to leave the OSCORE group as described in Section 5.2 of [I-D.ietf-ace-key-groupcomm], i.e. by sending to the Group Manager a request to the same Join Resource with the 'type' parameter set to 2 ("leave").

6. Public Keys of Joining Nodes

Source authentication of OSCORE messages exchanged within the group is ensured by means of digital counter signatures (see Sections 2 and 3 of [I-D.ietf-core-oscore-groupcomm]). Therefore, group members must be able to retrieve each other's public key from a trusted key repository, in order to verify source authenticity of incoming group messages.

As also discussed in [I-D.ietf-core-oscore-groupcomm], the Group Manager acts as trusted repository of the public keys of the group members, and provides those public keys to group members if requested to. Upon joining an OSCORE group, a joining node is thus expected to provide its own public key to the Group Manager.

In particular, one of the following four cases can occur when a new node joins an OSCORE group.

- o The joining node is going to join the group exclusively as monitor. That is, it is not going to send messages to the group, and hence to produce signatures with its own private key. In this case, the joining node is not required to provide its own public key to the Group Manager, which thus does not have to perform any check related to the public key encoding, or to a countersignature algorithm and possible associated parameters for that joining node.
- o The Group Manager already acquired the public key of the joining node during a past join process. In this case, the joining node MAY not provide again its own public key to the Group Manager, in order to limit the size of the Join Request. The joining node MUST provide its own public key again if it has provided the Group Manager with multiple public keys during past join processes, intended for different OSCORE groups. If the joining node provides its own public key, the Group Manager performs consistency checks as in Section 4.3 and, in case of success, considers it as the public key associated to the joining node in the OSCORE group.

- o The joining node and the Group Manager use an asymmetric proof-of-possession key to establish a secure communication channel. Then, two cases can occur.
 1. The proof-of-possession key is consistent with the encoding as well as with the counter signature algorithm and possible associated parameters used in the OSCORE group. Then, the Group Manager considers the proof-of-possession key as the public key associated to the joining node in the OSCORE group. If the joining node is aware that the proof-of-possession key is also valid for the OSCORE group, it MAY not provide it again as its own public key to the Group Manager. The joining node MUST provide its own public key again if it has provided the Group Manager with multiple public keys during past join processes, intended for different OSCORE groups. If the joining node provides its own public key in the 'client_cred' parameter of the Join Request (see Section 4.2), the Group Manager performs consistency checks as in Section 4.3 and, in case of success, considers it as the public key associated to the joining node in the OSCORE group.
 2. The proof-of-possession key is not consistent with the encoding or with the counter signature algorithm and possible associated parameters used in the OSCORE group. In this case, the joining node MUST provide a different consistent public key to the Group Manager in the 'client_cred' parameter of the Join Request (see Section 4.2). Then, the Group Manager performs consistency checks on this latest provided public key as in Section 4.3 and, in case of success, considers it as the public key associated to the joining node in the OSCORE group.
- o The joining node and the Group Manager use a symmetric proof-of-possession key to establish a secure communication channel. In this case, upon performing a join process with that Group Manager for the first time, the joining node specifies its own public key in the 'client_cred' parameter of the Join Request targeting the join endpoint (see Section 4.2).

Furthermore, as described in Section 4.2, the joining node may have explicitly requested the Group Manager to retrieve the public keys of the current group members, i.e. by including the 'get_pub_keys' parameter in the Join Request. In this case, the Group Manager includes also such public keys in the 'pub_keys' parameter of the Join Response (see Section 4.3).

Later on as a group member, the node may need to retrieve the public keys of other group members. The node can do that by exchanging with the Group Manager a shortened Join Request sent to the same Join

Resource with the 'type' parameter set to 5 ("pub keys") and a shortened Join Response, according to the approach defined in Section 7 of [I-D.ietf-ace-key-groupcomm].

7. Group Rekeying Process

In order to rekey the OSCORE group, the Group Manager distributes a new Group ID of the group and a new OSCORE Master Secret for that group. When doing so, the Group Manager MAY take a best effort to preserve the same unchanged Sender IDs for all group members. This avoids affecting the retrieval of public keys from the Group Manager as well as the verification of message countersignatures.

The Group Manager MUST support at least the following group rekeying scheme. Future application profiles may define alternative message formats and distribution schemes.

The Group Manager uses the same format of the Join Response message in Section 4.3. In particular:

- o Only the parameters 'type', 'kty', 'key', 'profile' and 'exp' are present.
- o The 'ms' parameter of the 'key' parameter specifies the new OSCORE Master Secret value.
- o The 'contextId' parameter of the 'key' parameter specifies the new Group ID.

The Group Manager separately sends a group rekeying message to each group member to be rekeyed. Each rekeying message MUST be secured with the pairwise secure communication channel between the Group Manager and the group member used during the join process.

This approach requires group members to act (also) as servers, in order to correctly handle unsolicited group rekeying messages from the Group Manager. In particular, if a group member and the Group Manager use OSCORE [I-D.ietf-core-object-security] to secure their pairwise communications, the group member MUST create a Replay Window in its own Recipient Context upon establishing the OSCORE Security Context with the Group Manager, e.g. by means of the OSCORE profile of ACE [I-D.ietf-ace-oscore-profile].

Group members and the Group Manager SHOULD additionally support alternative rekeying approaches that do not require group members to act (also) as servers. A number of such approaches are defined in Section 6 of [I-D.ietf-ace-key-groupcomm], and are based on the following rationale:

- o A group member queries the Group Manager for updated group keying material, by sending a dedicated request to the same Join Resource targeted when joining the group. Like for the case discussed in Section 4.3 where the OSCORE Security Context expires, the group member exchanges with the Group Manager a shortened Join Request sent to the same Join Resource with the 'type' parameter set to 3 ("update key") and a shortened Join Response message, according to the approach defined in Section 6 of [I-D.ietf-ace-key-groupcomm].
- o A group member subscribes for updates to the join resource and its associated group keying material on the Group Manager. This can rely on CoAP Observe [RFC7641] or on a full-fledged Pub-Sub model [I-D.ietf-core-coap-pubsub] with the Group Manager acting as Broker.

Either case, the Group Manager provides the (updated) group keying material as specified above in this section.

8. Security Considerations

The method described in this document leverages the following management aspects related to OSCORE groups and discussed in the sections of [I-D.ietf-core-oscore-groupcomm] referred below.

- o Management of group keying material (see Section 2.1 of [I-D.ietf-core-oscore-groupcomm]). The Group Manager is responsible for the renewal and re-distribution of the keying material in the groups of its competence (rekeying). According to the specific application requirements, this can include rekeying the group upon changes in its membership. In particular, renewing the keying material is required upon a new node's joining or a current node's leaving, in case backward security and forward security have to be preserved, respectively.
- o Provisioning and retrieval of public keys (see Section 2 of [I-D.ietf-core-oscore-groupcomm]). The Group Manager acts as key repository of public keys of group members, and provides them upon request.
- o Synchronization of sequence numbers (see Section 5 of [I-D.ietf-core-oscore-groupcomm]). This concerns how a responder node that has just joined an OSCORE group can synchronize with the sequence number of requesters in the same group.

Before sending the Join Response, the Group Manager MUST verify that the joining node actually owns the associated private key. To this end, the Group Manager can rely on the proof-of-possession challenge-response defined in Section 4. Alternatively, the joining node can

use its own public key as asymmetric proof-of-possession key to establish a secure channel with the Group Manager, e.g. as in Section 3.2 of [I-D.ietf-ace-dtls-authorize]. However, this requires such proof-of-possession key to be consistent with the encoding as well as with the countersignature algorithm and possible associated parameters used in the OSCORE group.

A node may have joined multiple OSCORE groups under different non-synchronized Group Managers. Therefore, it can happen that those OSCORE groups have the same Group Identifier (Gid). It follows that, upon receiving a Group OSCORE message addressed to one of those groups, the node would have multiple Security Contexts matching with the Gid in the incoming message. It is up to the application to decide how to handle such collisions of Group Identifiers, e.g. by trying to process the incoming message using one Security Context at the time until the right one is found.

Further security considerations are inherited from [I-D.ietf-ace-key-groupcomm], the ACE framework for Authentication and Authorization [I-D.ietf-ace-oauth-authz], and the specific transport profile of ACE signalled by the AS, such as [I-D.ietf-ace-dtls-authorize] and [I-D.ietf-ace-oscore-profile].

9. IANA Considerations

Note to RFC Editor: Please replace all occurrences of "[[This specification]]" with the RFC number of this specification and delete this paragraph.

This document has the following actions for IANA.

9.1. ACE Groupcomm Key Registry

IANA is asked to register the following entry in the "ACE Groupcomm Key" Registry defined in Section 11.5 of [I-D.ietf-ace-key-groupcomm].

- o Name: Group_OSCORE_Security_Context object
- o Key Type Value: TBD
- o Profile: "coap_group_oscore_app", defined in Section 9.3 of this specification.
- o Description: A Group_OSCORE_Security_Context object encoded as described in Section 4.3 of this specification.
- o Reference: [[This specification]]

9.2. OSCORE Security Context Parameters Registry

IANA is asked to register the following entries in the "OSCORE Security Context Parameters" Registry defined in Section 9.2 of [I-D.ietf-ace-oscore-profile].

- o Name: cs_alg
- o CBOR Label: TBD
- o CBOR Type: tstr / int
- o Registry: COSE Algorithm Values (ECDSA, EdDSA)
- o Description: OSCORE Counter Signature Algorithm Value
- o Reference: [[This specification]]
- o Name: cs_params
- o CBOR Label: TBD
- o CBOR Type: map
- o Registry: Counter Signatures Parameters
- o Description: OSCORE Counter Signature Algorithm Additional Parameters
- o Reference: [[This specification]]
- o Name: cs_key_params
- o CBOR Label: TBD
- o CBOR Type: map
- o Registry: Counter Signatures Key Parameters
- o Description: OSCORE Counter Signature Key Additional Parameters
- o Reference: [[This specification]]
- o Name: cs_key_enc
- o CBOR Label: TBD
- o CBOR Type: integer

- o Registry: ACE Public Key Encoding
- o Description: Encoding of Public Keys to be used with the OSCORE Counter Signature Algorithm
- o Reference: [[This specification]]

9.3. ACE Groupcomm Profile Registry

IANA is asked to register the following entry in the "ACE Groupcomm Profile" Registry defined in Section 11.6 of [I-D.ietf-ace-key-groupcomm].

- o Name: coap_group_oscore_app
- o Description: Application profile to provision keying material for participating in group communication protected with Group OSCORE as per [I-D.ietf-core-oscore-groupcomm].
- o CBOR Value: TBD
- o Reference: [[This specification]]

9.4. Sequence Number Synchronization Method Registry

IANA is asked to register the following entries in the "Sequence Number Synchronization Method" Registry defined in Section 11.8 of [I-D.ietf-ace-key-groupcomm].

- o Name: Best effort
- o Value: 1
- o Description: No action is taken.
- o Reference: [I-D.ietf-core-oscore-groupcomm] (Appendix E.1).
- o Name: Baseline
- o Value: 2
- o Description: The first received request sets the baseline reference point, and is discarded with no delivery to the application.
- o Reference: [I-D.ietf-core-oscore-groupcomm] (Appendix E.2).
- o Name: Echo challenge-response

- o Value: 3
- o Description: Challenge response using the Echo Option for CoAP from [I-D.ietf-core-echo-request-tag].
- o Reference: [I-D.ietf-core-oscore-groupcomm] (Appendix E.3).

9.5. ACE Public Key Encoding Registry

This specification registers the value defined in Figure 2 in the "ACE Public Key Encoding" IANA Registry.

10. References

10.1. Normative References

- [I-D.ietf-ace-key-groupcomm]
Palombini, F. and M. Tiloca, "Key Provisioning for Group Communication using ACE", draft-ietf-ace-key-groupcomm-02 (work in progress), July 2019.
- [I-D.ietf-ace-oauth-authz]
Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "Authentication and Authorization for Constrained Environments (ACE) using the OAuth 2.0 Framework (ACE-OAuth)", draft-ietf-ace-oauth-authz-24 (work in progress), March 2019.
- [I-D.ietf-ace-oscore-profile]
Palombini, F., Seitz, L., Selander, G., and M. Gunnarsson, "OSCORE profile of the Authentication and Authorization for Constrained Environments Framework", draft-ietf-ace-oscore-profile-07 (work in progress), February 2019.
- [I-D.ietf-core-object-security]
Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security for Constrained RESTful Environments (OSCORE)", draft-ietf-core-object-security-16 (work in progress), March 2019.
- [I-D.ietf-core-oscore-groupcomm]
Tiloca, M., Selander, G., Palombini, F., and J. Park, "Group OSCORE - Secure Group Communication for CoAP", draft-ietf-core-oscore-groupcomm-05 (work in progress), July 2019.

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<https://www.rfc-editor.org/info/rfc7252>>.
- [RFC8152] Schaad, J., "CBOR Object Signing and Encryption (COSE)", RFC 8152, DOI 10.17487/RFC8152, July 2017, <<https://www.rfc-editor.org/info/rfc8152>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.

10.2. Informative References

- [I-D.dijk-core-groupcomm-bis]
Dijk, E., Wang, C., and M. Tiloca, "Group Communication for the Constrained Application Protocol (CoAP)", draft-dijk-core-groupcomm-bis-00 (work in progress), March 2019.
- [I-D.ietf-ace-dtls-authorize]
Gerdes, S., Bergmann, O., Bormann, C., Selander, G., and L. Seitz, "Datagram Transport Layer Security (DTLS) Profile for Authentication and Authorization for Constrained Environments (ACE)", draft-ietf-ace-dtls-authorize-08 (work in progress), April 2019.
- [I-D.ietf-core-coap-pubsub]
Koster, M., Keranen, A., and J. Jimenez, "Publish-Subscribe Broker for the Constrained Application Protocol (CoAP)", draft-ietf-core-coap-pubsub-08 (work in progress), March 2019.
- [I-D.ietf-core-echo-request-tag]
Amsuess, C., Mattsson, J., and G. Selander, "CoAP: Echo, Request-Tag, and Token Processing", draft-ietf-core-echo-request-tag-05 (work in progress), May 2019.
- [I-D.tiloca-core-oscore-discovery]
Tiloca, M., Amsuess, C., and P. Stok, "Discovery of OSCORE Groups with the CoRE Resource Directory", draft-tiloca-core-oscore-discovery-02 (work in progress), March 2019.

- [RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <<https://www.rfc-editor.org/info/rfc6347>>.
- [RFC6749] Hardt, D., Ed., "The OAuth 2.0 Authorization Framework", RFC 6749, DOI 10.17487/RFC6749, October 2012, <<https://www.rfc-editor.org/info/rfc6749>>.
- [RFC7390] Rahman, A., Ed. and E. Dijk, Ed., "Group Communication for the Constrained Application Protocol (CoAP)", RFC 7390, DOI 10.17487/RFC7390, October 2014, <<https://www.rfc-editor.org/info/rfc7390>>.
- [RFC7641] Hartke, K., "Observing Resources in the Constrained Application Protocol (CoAP)", RFC 7641, DOI 10.17487/RFC7641, September 2015, <<https://www.rfc-editor.org/info/rfc7641>>.

Appendix A. Profile Requirements

This appendix lists the specifications on this application profile of ACE, based on the requirements defined in Appendix A of [I-D.ietf-ace-key-groupcomm].

- o Communication protocol that the members of the group must use: CoAP, possibly over IP multicast.
- o Security protocols that the group members must use to protect their communication: Group OSCORE.
- o Specify the encoding and value of the identifier of group and role of 'scope': see Section 3.1.
- o Profile identifier: coap_group_oscore_app
- o Acceptable values of 'kty': Group_OSCORE_Security_Context object
- o Specify the format and content of 'group_policies' entries: three values are defined and registered, as content of the entry "Sequence Number Synchronization Method" (see Section 9.4).
- o (Optional) specify the format and content of 'mgt_key_material': no.
- o (Optional) specify the transport profile of ACE [I-D.ietf-ace-oauth-authz] to use between Client and Group Manager: any transport profile of ACE that complies with the requirements in Appendix C of [I-D.ietf-ace-oauth-authz].

- o (Optional) specify the encoding of public keys, of 'client_cred', and of 'pub_keys' if COSE_Keys are not used: no.
- o (Optional) specify the acceptable values for parameters related to signature algorithm and signature keys: 'sign_alg' takes value from Tables 5 and 6 of [RFC8152]; 'sign_parameters' takes values from the "Counter Signature Parameters" Registry (see Section 9.1 of [I-D.ietf-core-oscore-groupcomm]); 'sign_key_parameters' takes values from the "Counter Signature Key Parameters" Registry (see Section 9.2 of [I-D.ietf-core-oscore-groupcomm]); 'pub_key_enc' takes value from Figure 2 in Section 4.1.
- o (Optional) specify the negotiation of parameter values for signature algorithm and signature keys, if 'sign_info' and 'pub_key_enc' are not used: pre-knowledge by using the approach based on the CoRE Resource Directory described in [I-D.tiloca-core-oscore-discovery].

Appendix B. Document Updates

RFC EDITOR: PLEASE REMOVE THIS SECTION.

B.1. Version -01 to -02

- o Editorial fixes.
- o Changed: "listener" to "responder"; "pure listener" to "monitor".
- o Changed profile name to "coap_group_oscore_app", to reflect it is an application profile.
- o Added the 'type' parameter for all requests to a Join Resource.
- o Added parameters to indicate the encoding of public keys.
- o Challenge-response for proof-of-possession of signature keys (Section 4).
- o Renamed 'key_info' parameter to 'sign_info'; updated its format; extended to include also parameters of the countersignature key (Section 4.1).
- o Code 4.00 (Bad request), in responses to joining nodes providing an invalid public key (Section 4.3).
- o Clarifications on provisioning and checking of public keys (Sections 4 and 6).

- o Extended discussion on group rekeying and possible different approaches (Section 7).
- o Extended security considerations: proof-of-possession of signature keys; collision of OSCORE Group Identifiers (Section 8).
- o Registered three entries in the IANA Registry "Sequence Number Synchronization Method Registry" (Section 9).
- o Registered one public key encoding in the "ACE Public Key Encoding" IANA Registry (Section 9).

B.2. Version -00 to -01

- o Changed name of 'req_aud' to 'audience' in the Authorization Request (Section 3.1).
- o Added negotiation of countersignature algorithm/parameters between Client and Group Manager (Section 4).
- o Updated format of the Key Distribution Response as a whole (Section 4.3).
- o Added parameter 'cs_params' in the 'key' parameter of the Key Distribution Response (Section 4.3).
- o New IANA registrations in the "ACE Authorization Server Request Creation Hints" Registry, "ACE Groupcomm Key" Registry, "OSCORE Security Context Parameters" Registry and "ACE Groupcomm Profile" Registry (Section 9).

Acknowledgments

The authors sincerely thank Santiago Aragon, Stefan Beck, Martin Gunnarsson, Rikard Hoeglund, Jim Schaad, Ludwig Seitz, Goeran Selander and Peter van der Stok for their comments and feedback.

The work on this document has been partly supported by VINNOVA and the Celtic-Next project CRITISEC; and by the EIT-Digital High Impact Initiative ACTIVE.

Authors' Addresses

Marco Tiloca
RISE AB
Isafjordsgatan 22
Kista SE-164 29 Stockholm
Sweden

Email: marco.tiloca@ri.se

Jiye Park
Universitaet Duisburg-Essen
Schuetzenbahn 70
Essen 45127
Germany

Email: ji-ye.park@uni-due.de

Francesca Palombini
Ericsson AB
Torshamnsgatan 23
Kista SE-16440 Stockholm
Sweden

Email: francesca.palombini@ericsson.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 12, 2019

M. Koster
SmartThings
A. Keranen
J. Jimenez
Ericsson
March 11, 2019

Publish-Subscribe Broker for the Constrained Application Protocol (CoAP)
draft-ietf-core-coap-pubsub-08

Abstract

The Constrained Application Protocol (CoAP), and related extensions are intended to support machine-to-machine communication in systems where one or more nodes are resource constrained, in particular for low power wireless sensor networks. This document defines a publish-subscribe Broker for CoAP that extends the capabilities of CoAP for supporting nodes with long breaks in connectivity and/or up-time.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 13, 2019.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must

include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

| | |
|---|----|
| 1. Introduction | 2 |
| 2. Terminology | 3 |
| 3. Architecture | 4 |
| 3.1. CoAP Pub/sub Architecture | 4 |
| 3.2. CoAP Pub/sub Broker | 5 |
| 3.3. CoAP Pub/sub Client | 5 |
| 3.4. CoAP Pub/sub Topic | 5 |
| 3.5. Brokerless Pub/sub | 6 |
| 4. CoAP Pub/sub REST API | 6 |
| 4.1. DISCOVERY | 6 |
| 4.2. CREATE | 9 |
| 4.3. PUBLISH | 11 |
| 4.4. SUBSCRIBE | 14 |
| 4.5. UNSUBSCRIBE | 15 |
| 4.6. READ | 17 |
| 4.7. REMOVE | 18 |
| 5. CoAP Pub/sub Operation with Resource Directory | 20 |
| 6. Sleep-Wake Operation | 20 |
| 7. Simple Flow Control | 20 |
| 8. Security Considerations | 21 |
| 9. IANA Considerations | 22 |
| 9.1. Resource Type value 'core.ps' | 22 |
| 9.2. Resource Type value 'core.ps.discover' | 22 |
| 10. Acknowledgements | 22 |
| 11. References | 23 |
| 11.1. Normative References | 23 |
| 11.2. Informative References | 24 |
| Authors' Addresses | 24 |

1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] supports machine-to-machine communication across networks of constrained devices. CoAP uses a request/response model where clients make requests to servers in order to request actions on resources. Depending on the situation the same device may act either as a server, a client, or both.

One important class of constrained devices includes devices that are intended to run for years from a small battery, or by scavenging energy from their environment. These devices have limited reachability because they spend most of their time in a sleeping

state with no network connectivity. Devices may also have limited reachability due to certain middle-boxes, such as Network Address Translators (NATs) or firewalls. Such middle-boxes often prevent connecting to a device from the Internet unless the connection was initiated by the device.

For some applications the client/server and request/response communication model is not optimal but publish-subscribe communication with potentially many senders and/or receivers and communication via topics rather than directly with endpoints may fit better.

This document specifies simple extensions to CoAP for enabling publish-subscribe communication using a Broker node that enables store-and-forward messaging between two or more nodes. This model facilitates communication of nodes with limited reachability, enables simple many-to-many communication, and eases integration with other publish-subscribe systems.

2. 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 BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

This specification requires readers to be familiar with all the terms and concepts that are discussed in [RFC5988] and [RFC6690]. Readers should also be familiar with the terms and concepts discussed in [RFC7252] and [I-D.ietf-core-resource-directory]. The URI template format [RFC6570] is used to describe the REST API defined in this specification.

This specification makes use of the following additional terminology:

Publish-Subscribe (pub/sub): A messaging paradigm where messages are published to a Broker and potential receivers can subscribe to the Broker to receive messages. The publishers do not (need to) know where the message will be eventually sent: the publications and subscriptions are matched by a Broker and publications are delivered by the Broker to subscribed receivers.

CoAP pub/sub service: A group of REST resources, as defined in this document, which together implement the required features of this specification.

CoAP pub/sub Broker: A server node capable of receiving messages (publications) from and sending messages to other nodes, and able to match subscriptions and publications in order to route messages to the right destinations. The Broker can also temporarily store publications to satisfy future subscriptions and pending notifications.

CoAP pub/sub Client: A CoAP client which is capable of publish or subscribe operations as defined in this specification.

Topic: A unique identifier for a particular item being published and/or subscribed to. A Broker uses the topics to match subscriptions to publications. A reference to a Topic on a Broker is a valid CoAP URI as defined in [RFC7252]

3. Architecture

3.1. CoAP Pub/sub Architecture

Figure 1 shows the architecture of a CoAP pub/sub service. CoAP pub/sub Clients interact with a CoAP pub/sub Broker through the CoAP pub/sub REST API which is hosted by the Broker. State information is updated between the Clients and the Broker. The CoAP pub/sub Broker performs a store-and-forward of state update representations between certain CoAP pub/sub Clients. Clients Subscribe to topics upon which representations are Published by other Clients, which are forwarded by the Broker to the subscribing clients. A CoAP pub/sub Broker may be used as a REST resource proxy, retaining the last published representation to supply in response to Read requests from Clients.

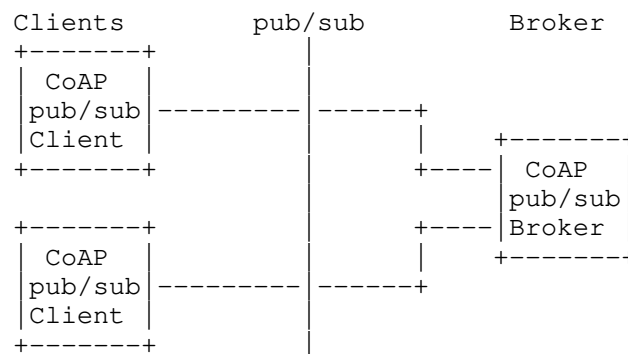


Figure 1: CoAP pub/sub Architecture

3.2. CoAP Pub/sub Broker

A CoAP pub/sub Broker is a CoAP Server that exposes a REST API for clients to use to initiate publish-subscribe interactions. Avoiding the need for direct reachability between clients, the Broker only needs to be reachable from all clients. The Broker also needs to have sufficient resources (storage, bandwidth, etc.) to host CoAP resource services, and potentially buffer messages, on behalf of the clients.

3.3. CoAP Pub/sub Client

A CoAP pub/sub Client interacts with a CoAP pub/sub Broker using the CoAP pub/sub REST API defined in this document. Clients initiate interactions with a CoAP pub/sub Broker. A data source (e.g., sensor clients) can publish state updates to the Broker and data sinks (e.g., actuator clients) can read from or subscribe to state updates from the Broker. Application clients can make use of both publish and subscribe in order to exchange state updates with data sources and data sinks.

3.4. CoAP Pub/sub Topic

The clients and Broker use topics to identify a particular resource or object in a publish-subscribe system. Topics are conventionally formed as a hierarchy, e.g. `"/sensors/weather/barometer/pressure"` or `"/EP-33543/sen/3303/0/5700"`. The topics are hosted by a Broker and all the clients using the Broker share the same namespace for topics.

Every CoAP pub/sub topic has an associated link, consisting of a reference path on the Broker using URI path [RFC3986] construction and link attributes [RFC6690]. Every topic is associated with zero or more stored representations and a content-format specified in the link. A CoAP pub/sub topic value may alternatively consist of a collection of one or more sub-topics, consisting of links to the sub-topic URIs and indicated by a link-format content-format. Sub-topics are also topics and may have their own sub-topics, forming a tree structure of unique paths that is implemented using URIs. The full URI of a topic includes a scheme and authority for the Broker, for example `"coaps://192.0.2.13:5684/EP-33543/sen/3303/0/5700"`.

A Topic may have a lifetime defined by using the CoAP Max-Age option on topic creation, or on publish operations to the topic. The lifetime is refreshed each time a representation is published to the topic. If the lifetime expires, the topic is removed from discovery responses, returns errors on subscription, and any outstanding subscriptions are cancelled.

3.5. Brokerless Pub/sub

In some use cases, it is desirable to use pub/sub semantics for peer-to-peer communication, but it is not feasible or desirable to include a separate node on the network to serve as a Broker. In other use cases, it is desirable to enable one-way-only communication, such as sensors pushing updates to a service.

Figure 2 shows an arrangement for using CoAP pub/sub in a "Brokerless" configuration between peer nodes. Nodes in a Brokerless system may act as both Broker and client. A node that supports Broker functionality may be pre-configured with topics that expose services and resources. Brokerless peer nodes can be mixed with client and Broker nodes in a system with full interoperability.

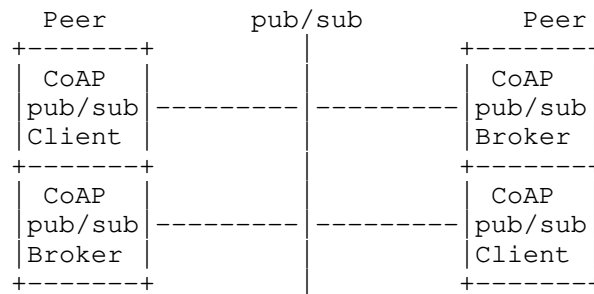


Figure 2: Brokerless pub/sub

4. CoAP Pub/sub REST API

This section defines the REST API exposed by a CoAP pub/sub Broker to pub/sub Clients. The examples throughout this section assume the use of CoAP [RFC7252]. A CoAP pub/sub Broker implementing this specification SHOULD support the DISCOVERY, CREATE, PUBLISH, SUBSCRIBE, UNSUBSCRIBE, READ, and REMOVE operations defined in this section. Optimized implementations MAY support a subset of the operations as required by particular constrained use cases.

4.1. DISCOVERY

CoAP pub/sub Clients discover CoAP pub/sub Brokers by using CoAP Simple Discovery or through a Resource Directory (RD) [I-D.ietf-core-resource-directory]. A CoAP pub/sub Broker SHOULD indicate its presence and availability on a network by exposing a link to the entry point of its pub/sub API at its .well-known/core location [RFC6690]. A CoAP pub/sub Broker MAY register its pub/sub REST API entry point with a Resource Directory. Figure 3 shows an

example of a client discovering a local pub/sub API using CoAP Simple Discovery. A Broker wishing to advertise the CoAP pub/sub API for Simple Discovery or through a Resource Directory MUST use the link relation `rt=core.ps`. A Broker MAY advertise its supported content formats and other attributes in the link to its pub/sub API.

A CoAP pub/sub Broker MAY offer a topic discovery entry point to enable Clients to find topics of interest, either by topic name or by link attributes which may be registered when the topic is created. Figure 4 shows an example of a client looking for a topic with a resource type (rt) of "temperature" using Discover. The client then receives the URI of the resource and its content-format. A pub/sub Broker wishing to advertise topic discovery MUST use the relation `rt=core.ps.discover` in the link.

A CoAP pub/sub Broker MAY provide topic discovery functionality through the `.well-known/core` resource. Links to topics may be exposed at `.well-known/core` in addition to links to the pub/sub API. Figure 5 shows an example of topic discovery through `.well-known/core`.

Topics in the broker may be created in hierarchies (see Section 4.2) with parent topics having sub-topics. For a discovery the broker may choose to not expose the sub-topics in order to limit amount of topic links sent in a discovery response. The client can then perform discovery for the parent topics it wants to discover the sub-topics.

The DISCOVER interface is specified as follows:

Interaction: Client -> Broker

Method: GET

URI Template: `{+ps}/{+topic}{?q*}`

URI Template Variables: `ps` := Pub/sub REST API entry point (optional). The entry point of the pub/sub REST API, as obtained from discovery, used to discover topics.

`topic` := The desired topic to return links for (optional).

`q` := Query Filter (optional). MAY contain a query filter list as per [RFC6690] Section 4.1.

Content-Format: `application/link-format`

The following response codes are defined for the DISCOVER operation:

Success: 2.05 "Content" with an application/link-format payload containing one or more matching entries for the Broker resource. A pub/sub Broker SHOULD use the value "/ps/" for the base URI of the pub/sub API wherever possible.

Failure: 4.04 "Not Found" is returned in case no matching entry is found for a unicast request.

Failure: 4.00 "Bad Request" is returned in case of a malformed request for a unicast request.

Failure: No error response to a multicast request.

| Client | Broker |
|---|---|
| <pre> ----- GET /.well-known/core?rt=core.ps ---->> -- Content-Format: application/link-format --- </pre> | <pre> <<--- 2.05 Content </ps/>;rt=core.ps;rt=core.ps.discover;ct=40 --- </pre> |

Figure 3: Example of DISCOVER pub/sub function

| Client | Broker |
|--|--|
| <pre> ----- GET /ps/?rt="temperature" ----->> Content-Format: application/link-format </pre> | <pre> <<--- 2.05 Content </ps/currentTemp>;rt="temperature";ct=50 --- </pre> |

Figure 4: Example of DISCOVER topic

| Client | Broker |
|--|--|
| <pre> ----- GET /.well-known/core?ct=50 ----->> Content-Format: application/link-format </pre> | <pre> <<--- 2.05 Content </ps/currentTemp>;rt="temperature";ct=50 --- </pre> |

Figure 5: Example of DISCOVER topic

4.2. CREATE

A CoAP pub/sub broker SHOULD allow Clients to create new topics on the broker using CREATE. Some exceptions are for fixed brokerless devices and pre-configured brokers in dedicated installations. A client wishing to create a topic MUST use a CoAP POST to the pub/sub API with a payload indicating the desired topic. The topic specification sent in the payload MUST use a supported serialization of the CoRE link format [RFC6690]. The target of the link MUST be a URI formatted string. The client MUST indicate the desired content format for publishes to the topic by using the ct (Content Format) link attribute in the link-format payload. Additional link target attributes and relation values MAY be included in the topic specification link when a topic is created.

The client MAY indicate the lifetime of the topic by including the Max-Age option in the CREATE request.

Topic hierarchies can be created by creating parent topics. A parent topic is created with a POST using ct (Content Format) link attribute value which describes a supported serialization of the CoRE link format [RFC6690], such as application/link-format (ct=40) or its JSON or CBOR serializations. If a topic is created which describes a link serialization, that topic may then have sub-topics created under it as shown in Figure 7.

Only one level in the topic hierarchy may be created as a result of a CREATE operation, unless create on PUBLISH is supported (see Section 4.3). The topic string used in the link target MUST NOT contain the "/" character.

A topic creator MUST include exactly one content format link attribute value (ct) in the create payload. If the content format option is not included or if the option is repeated, the Broker MUST reject the operation with an error code of "4.00 Bad Request".

Only one topic may be created per request. If there is more than one link included in a CREATE request, the Broker MUST reject the operation with an error code of "4.00 Bad Request".

A Broker MUST return a response code of "2.01 Created" if the topic is created and MUST return the URI path of the created topic via Location-Path options. If a new topic can not be created, the Broker MUST return the appropriate 4.xx response code indicating the reason for failure.

A Broker SHOULD remove topics if the Max-Age of the topic is exceeded without any publishes to the topic. A Broker SHOULD retain a topic

indefinitely if the Max-Age option is elided or is set to zero upon topic creation. The lifetime of a topic MUST be refreshed upon create operations with a target of an existing topic.

A topic creator SHOULD PUBLISH an initial value to a newly-created Topic in order to enable responses to READ and SUBSCRIBE requests that may be submitted after the topic is discoverable.

The CREATE interface is specified as follows:

Interaction: Client -> Broker

Method: POST

URI Template: {+ps}/{+topic}

URI Template Variables: ps := Pub/sub REST API entry point (optional). The entry point of the pub/sub REST API, as obtained from discovery, used to discover topics.

topic := The desired topic to return links for (optional).

Content-Format: application/link-format

Payload: The desired topic to CREATE

The following response codes are defined for the CREATE operation:

Success: 2.01 "Created". Successful Creation of the topic

Failure: 4.00 "Bad Request". Malformed request.

Failure: 4.01 "Unauthorized". Authorization failure.

Figure 6 shows an example of a topic called "topic1" being successfully created.

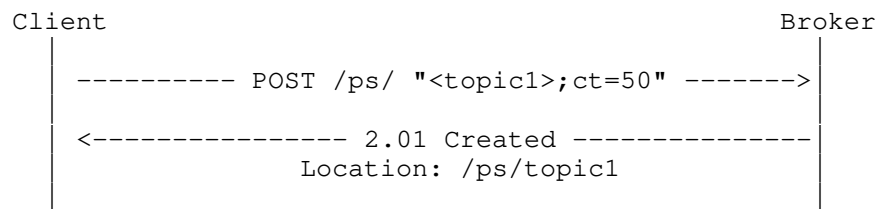


Figure 6: Example of CREATE topic

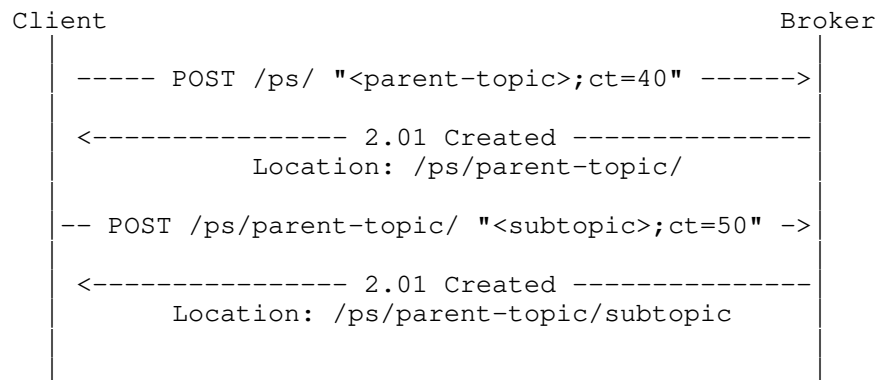


Figure 7: Example of CREATE of topic hierarchy

4.3. PUBLISH

A CoAP pub/sub Broker MAY allow clients to PUBLISH to topics on the Broker. A client MAY use the PUT or the POST method to publish state updates to the CoAP pub/sub Broker. A client MUST use the content format specified upon creation of a given topic to publish updates to that topic. The Broker MUST reject publish operations which do not use the specified content format. A CoAP client publishing on a topic MAY indicate the maximum lifetime of the value by including the Max-Age option in the publish request. The Broker MUST return a response code of "2.04 Changed" if the publish is accepted. A Broker MAY return a "4.04 Not Found" if the topic does not exist. A Broker MAY return "4.29 Too Many Requests" if simple flow control as described in Section 7 is implemented.

A Broker MUST accept PUBLISH operations using the PUT method. PUBLISH operations using the PUT method replace any stored representation associated with the topic, with the supplied representation. A Broker MAY reject, or delay responses to, PUT requests to a topic while pending resolution of notifications to subscribers from previous PUT requests.

Create on PUBLISH: A Broker MAY accept PUBLISH operations to new topics using the PUT method. If a Broker accepts a PUBLISH using PUT to a topic that does not exist, the Broker MUST create the topic using the information in the PUT operation. The Broker MUST create a topic with the URI-Path of the request, including all of the sub-topics necessary, and create a topic link with the ct attribute set to the content-format value from the header of the PUT request. If topic is created, the Broker MUST return the response "2.01 Created"

with the URI of the created topic, including all of the created path segments, returned via the Location-Path option.

Figure 9 shows an example of a topic being created on first PUBLISH.

A Broker MAY accept PUBLISH operations using the POST method. If a Broker accepts PUBLISH using POST it shall respond with the 2.04 Changed status code. If an attempt is made to PUBLISH using POST to a topic that does not exist, the Broker SHALL return a response status indicating resource not found, such as HTTP 404 or CoAP 4.04.

A Broker MAY perform garbage collection of stored representations which have been delivered to all subscribers or which have timed out. A Broker MAY retain at least one most recently published representation to return in response to SUBSCRIBE and READ requests.

A Broker MUST make a best-effort attempt to notify all clients subscribed on a particular topic each time it receives a publish on that topic. An example is shown in Figure 10.

If a client publishes to a Broker without the Max-Age option, the Broker MUST refresh the topic lifetime with the most recently set Max-Age value, and the Broker MUST include the most recently set Max-Age value in the Max-Age option of all notifications.

If a client publishes to a Broker with the Max-Age option, the Broker MUST include the same value for the Max-Age option in all notifications.

A Broker MUST use CoAP Notification as described in [RFC7641] to notify subscribed clients.

The PUBLISH operation is specified as follows:

Interaction: Client -> Broker

Method: PUT, POST

URI Template: {+ps}/{+topic}

URI Template Variables: ps := Pub/sub REST API entry point (optional). The entry point of the pub/sub REST API, as obtained from discovery, used to discover topics.

topic := The desired topic to return links for (optional).

Content-Format: Any valid CoAP content format

Payload: Representation of the topic value (CoAP resource state representation) in the indicated content format

The following response codes are defined for the PUBLISH operation:

Success: 2.01 "Created". Successful publish, topic is created

Success: 2.04 "Changed". Successful publish, topic is updated

Failure: 4.00 "Bad Request". Malformed request.

Failure: 4.01 "Unauthorized". Authorization failure.

Failure: 4.04 "Not Found". Topic does not exist.

Failure: 4.29 "Too Many Requests". The client should slow down the rate of publish messages for this topic (see Section 7).

Figure 8 shows an example of a new value being successfully published to the topic "topic1". See Figure 10 for an example of a Broker forwarding a message from a publishing client to a subscribed client.

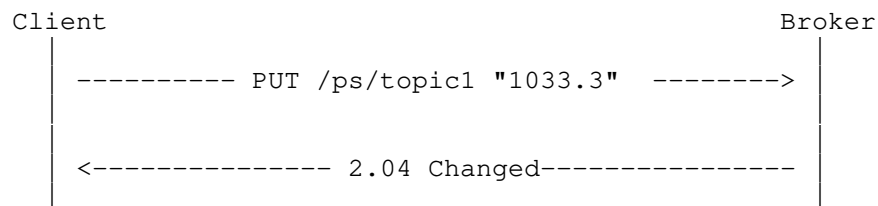


Figure 8: Example of PUBLISH

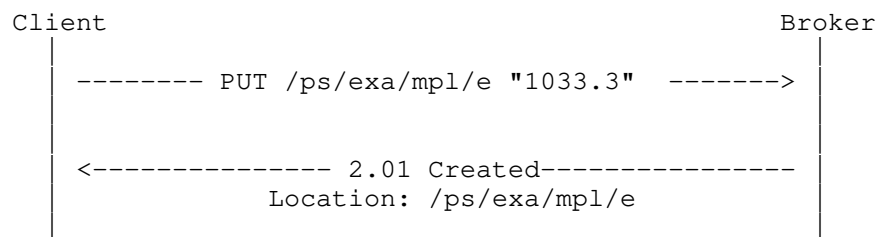


Figure 9: Example of CREATE on PUBLISH

4.4. SUBSCRIBE

A CoAP pub/sub Broker MAY allow Clients to subscribe to topics on the Broker using CoAP Observe as described in [RFC7641]. A CoAP pub/sub Client wishing to Subscribe to a topic on a Broker MUST use a CoAP GET with the Observe option set to 0 (zero). The Broker MAY add the client to a list of observers. The Broker MUST return a response code of "2.05 Content" along with the most recently published value if the topic contains a valid value and the Broker can supply the requested content format. The Broker MUST reject Subscribe requests on a topic if the content format of the request is not the content format the topic was created with.

If the topic was published with the Max-Age option, the Broker MUST set the Max-Age option in the valid response to the amount of time remaining for the value to be valid since the last publish operation on that topic.

The Broker MUST return a response code "4.04 Not Found" if the topic does not exist or has been removed, or if Max-Age of a previously published representation has expired.

If a Topic has been created but not yet published to when a SUBSCRIBE to the topic is received, the Broker MAY acknowledge and queue the pending SUBSCRIBE and defer the response until an initial PUBLISH occurs.

The Broker MUST return a response code "4.15 Unsupported Content Format" if it can not return the requested content format. If a Broker is unable to accept a new Subscription on a topic, it SHOULD return the appropriate response code without the Observe option as per [RFC7641] Section 4.1.

There is no explicit maximum lifetime of a Subscription, thus a Broker may remove subscribers at any time. The Broker, upon removing a Subscriber, will transmit the appropriate response code without the Observe option, as per [RFC7641] Section 4.2, to the removed Subscriber.

The SUBSCRIBE operation is specified as follows:

Interaction: Client -> Broker

Method: GET

Options: Observe:0

URI Template: {+ps}/{+topic}

URI Template Variables: `ps` := Pub/sub REST API entry point (optional). The entry point of the pub/sub REST API, as obtained from discovery, used to discover topics.

`topic` := The desired topic to return links for (optional).

The following response codes are defined for the SUBSCRIBE operation:

Success: 2.05 "Content". Successful subscribe, current value included

Failure: 4.00 "Bad Request". Malformed request.

Failure: 4.01 "Unauthorized". Authorization failure.

Failure: 4.04 "Not Found". Topic does not exist.

Failure: 4.15 "Unsupported Content Format". Unsupported content format.

Figure 10 shows an example of Client2 subscribing to "topic1" and receiving a response from the Broker, with a subsequent notification. The subscribe response from the Broker uses the last stored value associated with the topic1. The notification from the Broker is sent in response to the publish received from Client1.

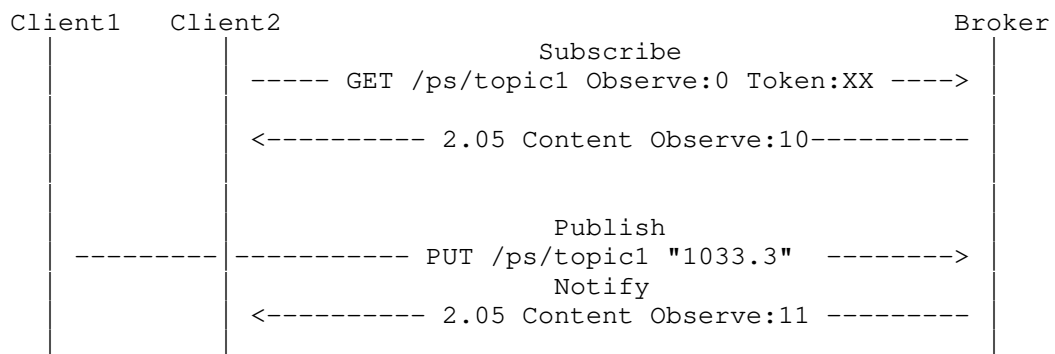


Figure 10: Example of SUBSCRIBE

4.5. UNSUBSCRIBE

If a CoAP pub/sub Broker allows clients to SUBSCRIBE to topics on the Broker, it MUST allow Clients to unsubscribe from topics on the Broker using the CoAP Cancel Observation operation. A CoAP pub/sub Client wishing to unsubscribe to a topic on a Broker MUST either use

CoAP GET with Observe using an Observe parameter of 1 or send a CoAP Reset message in response to a publish, as per [RFC7641].

The UNSUBSCRIBE operation is specified as follows:

Interaction: Client -> Broker

Method: GET

Options: Observe:1

URI Template: {+ps}/{+topic}{?q*}

URI Template Variables: ps := Pub/sub REST API entry point (optional). The entry point of the pub/sub REST API, as obtained from discovery, used to discover topics.

topic := The desired topic to return links for (optional).

q := Query Filter (optional). MAY contain a query filter list as per [RFC6690] Section 4.1.

The following response codes are defined for the UNSUBSCRIBE operation:

Success: 2.05 "Content". Successful unsubscribe, current value included

Success: 2.07 "No Content". Successful unsubscribe, value not included

Failure: 4.00 "Bad Request". Malformed request.

Failure: 4.01 "Unauthorized". Authorization failure.

Failure: 4.04 "Not Found". Topic does not exist.

Figure 11 shows an example of a client unsubscribe using the Observe=1 cancellation method.

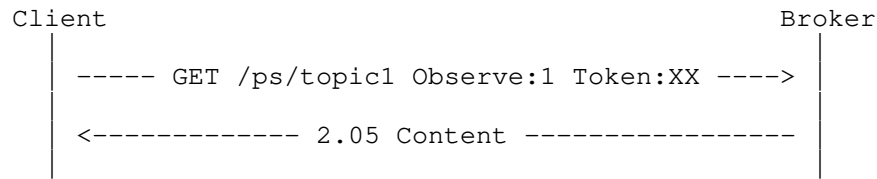


Figure 11: Example of UNSUBSCRIBE

4.6. READ

A CoAP pub/sub Broker MAY accept Read requests on a topic using the the CoAP GET method if the content format of the request matches the content format the topic was created with. The Broker MUST return a response code of "2.05 Content" along with the most recently published value if the topic contains a valid value and the Broker can supply the requested content format.

If the topic was published with the Max-Age option, the Broker MUST set the Max-Age option in the valid response to the amount of time remaining for the value to be valid since the last publish operation on that topic.

The Broker MUST return a response code "4.04 Not Found" if the topic does not exist or has been removed, or if Max-Age of a previously published representation has expired.

If a Topic has been created but not yet published to when a READ to the topic is received, the Broker MAY acknowledge and queue the pending READ, and defer the response until an initial PUBLISH occurs.

The Broker MUST return a response code "4.15 Unsupported Content Format" if the Broker can not return the requested content format.

The READ operation is specified as follows:

Interaction: Client -> Broker

Method: GET

URI Template: {+ps}/{+topic}

URI Template Variables: ps := Pub/sub REST API entry point (optional). The entry point of the pub/sub REST API, as obtained from discovery, used to discover topics.

topic := The desired topic to return links for (optional).

The following response codes are defined for the READ operation:

Success: 2.05 "Content". Successful READ, current value included

Failure: 4.00 "Bad Request". Malformed request.

Failure: 4.01 "Unauthorized". Authorization failure.

Failure: 4.04 "Not Found". Topic does not exist.

Failure: 4.15 "Unsupported Content Format". Unsupported content-format.

Figure 12 shows an example of a successful READ from topic1, followed by a Publish on the topic, followed at some time later by a read of the updated value from the recent Publish.

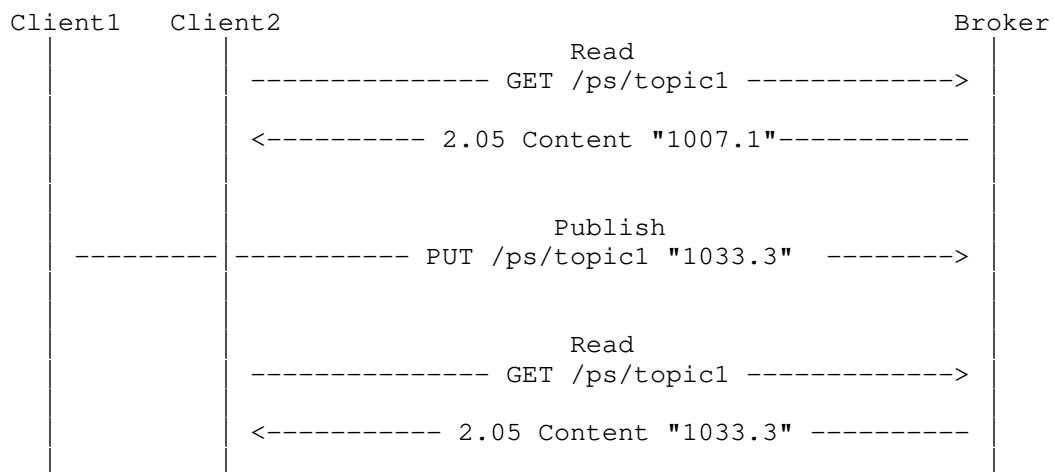


Figure 12: Example of READ

4.7. REMOVE

A CoAP pub/sub Broker MAY allow clients to remove topics from the Broker using the CoAP Delete method on the URI of the topic. The CoAP pub/sub Broker MUST return "2.02 Deleted" if the removal is successful. The Broker MUST return the appropriate 4.xx response code indicating the reason for failure if the topic can not be removed.

When a topic is removed for any reason, the Broker SHOULD remove all of the observers from the list of observers and return a final 4.04

"Not Found" response as per [RFC7641] Section 3.2. If a topic which has sub-topics is removed, then all of its sub-topics MUST be recursively removed.

The REMOVE operation is specified as follows:

Interaction: Client -> Broker

Method: DELETE

URI Template: {+ps}/{+topic}

URI Template Variables: ps := Pub/sub REST API entry point (optional). The entry point of the pub/sub REST API, as obtained from discovery, used to discover topics.

topic := The desired topic to return links for (optional).

Content-Format: None

Response Payload: None

The following response codes are defined for the REMOVE operation:

Success: 2.02 "Deleted". Successful remove

Failure: 4.00 "Bad Request". Malformed request.

Failure: 4.01 "Unauthorized". Authorization failure.

Failure: 4.04 "Not Found". Topic does not exist.

Figure 13 shows a successful remove of topic1.

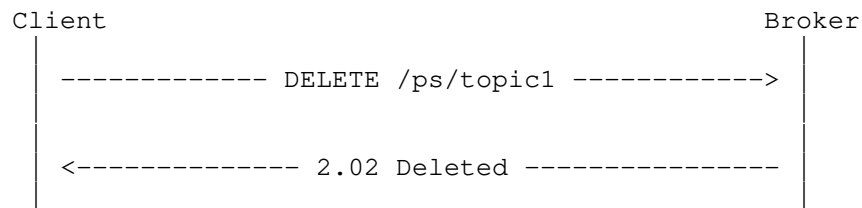


Figure 13: Example of REMOVE

5. CoAP Pub/sub Operation with Resource Directory

A CoAP pub/sub Broker may register the base URI, which is the REST API entry point for a pub/sub service, with a Resource Directory. A pub/sub Client may use an RD to discover a pub/sub Broker.

A CoAP pub/sub Client may register links [RFC6690] with a Resource Directory to enable discovery of created pub/sub topics. A pub/sub Client may use an RD to discover pub/sub Topics. A client which registers pub/sub Topics with an RD MUST use the context relation (con) [I-D.ietf-core-resource-directory] to indicate that the context of the registered links is the pub/sub Broker.

A CoAP pub/sub Broker may alternatively register links to its topics to a Resource Directory by triggering the RD to retrieve it's links from .well-known/core. In order to use this method, the links must first be exposed in the .well-known/core of the pub/sub Broker. See Section 4.1 in this document.

The pub/sub Broker triggers the RD to retrieve its links by sending a POST with an empty payload to the .well-known/core of the Resource Directory. The RD server will then retrieve the links from the .well-known/core of the pub/sub Broker and incorporate them into the Resource Directory. See [I-D.ietf-core-resource-directory] for further details.

6. Sleep-Wake Operation

CoAP pub/sub provides a way for client nodes to sleep between operations, conserving energy during idle periods. This is made possible by shifting the server role to the Broker, allowing the Broker to be always-on and respond to requests from other clients while a particular client is sleeping.

For example, the Broker will retain the last state update received from a sleeping client, in order to supply the most recent state update to other clients in response to read and subscribe operations.

Likewise, the Broker will retain the last state update received on the topic such that a sleeping client, upon waking, can perform a read operation to the Broker to update its own state from the most recent system state update.

7. Simple Flow Control

Since the Broker node has to potentially send a large amount of notification messages for each publish message and it may be serving a large amount of subscribers and publishers simultaneously, the

Broker may become overwhelmed if it receives many publish messages to popular topics in a short period of time.

If the Broker is unable to serve a certain client that is sending publish messages too fast, the Broker SHOULD respond with Response Code 4.29, "Too Many Requests" [RFC8516] and set the Max-Age Option to indicate the number of seconds after which the client can retry. The Broker MAY stop creating notifications from the publish messages from this client and to this topic for the indicated time.

If a client receives the 4.29 Response Code from the Broker for a publish message to a topic, it MUST NOT send new publish messages to the Broker on the same topic before the time indicated in Max-Age has passed.

8. Security Considerations

CoAP pub/sub re-uses CoAP [RFC7252], CoRE Resource Directory [I-D.ietf-core-resource-directory], and Web Linking [RFC5988] and therefore the security considerations of those documents also apply to this specification. Additionally, a CoAP pub/sub Broker and the clients SHOULD authenticate each other and enforce access control policies. A malicious client could subscribe to data it is not authorized to or mount a denial of service attack against the Broker by publishing a large number of resources. The authentication can be performed using the already standardized DTLS offered mechanisms, such as certificates. DTLS also allows communication security to be established to ensure integrity and confidentiality protection of the data exchanged between these relevant parties. Provisioning the necessary credentials, trust anchors and authorization policies is non-trivial and subject of ongoing work.

The use of a CoAP pub/sub Broker introduces challenges for the use of end-to-end security between for example a client device on a sensor network and a client application running in a cloud-based server infrastructure since Brokers terminate the exchange. While running separate DTLS sessions from the client device to the Broker and from Broker to client application protects confidentiality on those paths, the client device does not know whether the commands coming from the Broker are actually coming from the client application. Similarly, a client application requesting data does not know whether the data originated on the client device. For scenarios where end-to-end security is desirable the use of application layer security is unavoidable. Application layer security would then provide a guarantee to the client device that any request originated at the client application. Similarly, integrity protected sensor data from a client device will also provide guarantee to the client application that the data originated on the client device itself. The protected

data can also be verified by the intermediate Broker ensuring that it stores/caches correct request/response and no malicious messages/requests are accepted. The Broker would still be able to perform aggregation of data/requests collected.

Depending on the level of trust users and system designers place in the CoAP pub/sub Broker, the use of end-to-end object security is RECOMMENDED as described in [I-D.palombini-ace-coap-pubsub-profile]. An example application that uses the CoAP pub/sub Broker and relies on end-to-end object security is described in [RFC8387]. When only end-to-end encryption is necessary and the CoAP Broker is trusted, Payload Only Protection (Mode:PAYL) could be used. The Publisher would wrap only the payload before sending it to the Broker and set the option Content-Format to application/smpayl. Upon receipt, the Broker can read the unencrypted CoAP header to forward it to the subscribers.

9. IANA Considerations

This document registers one attribute value in the Resource Type (rt=) registry established with [RFC6690] and appends to the definition of one CoAP Response Code in the CoRE Parameters Registry.

9.1. Resource Type value 'core.ps'

- o Attribute Value: core.ps
- o Description: Section 4 of [[This document]]
- o Reference: [[This document]]
- o Notes: None

9.2. Resource Type value 'core.ps.discover'

- o Attribute Value: core.ps.discover
- o Description: Section 4 of [[This document]]
- o Reference: [[This document]]
- o Notes: None

10. Acknowledgements

The authors would like to thank Hannes Tschofenig, Zach Shelby, Mohit Sethi, Peter van der Stok, Tim Kellogg, Anders Eriksson, Goran

Selander, Mikko Majanen, and Olaf Bergmann for their contributions and reviews.

11. References

11.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <<https://www.rfc-editor.org/info/rfc3986>>.
- [RFC6570] Gregorio, J., Fielding, R., Hadley, M., Nottingham, M., and D. Orchard, "URI Template", RFC 6570, DOI 10.17487/RFC6570, March 2012, <<https://www.rfc-editor.org/info/rfc6570>>.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <<https://www.rfc-editor.org/info/rfc6690>>.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<https://www.rfc-editor.org/info/rfc7252>>.
- [RFC7641] Hartke, K., "Observing Resources in the Constrained Application Protocol (CoAP)", RFC 7641, DOI 10.17487/RFC7641, September 2015, <<https://www.rfc-editor.org/info/rfc7641>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8516] Keranen, A., "'Too Many Requests' Response Code for the Constrained Application Protocol", RFC 8516, DOI 10.17487/RFC8516, January 2019, <<https://www.rfc-editor.org/info/rfc8516>>.

11.2. Informative References

- [I-D.ietf-core-object-security]
Selander, G., Mattsson, J., Palombini, F., and L. Seitz,
"Object Security for Constrained RESTful Environments
(OSCORE)", draft-ietf-core-object-security-16 (work in
progress), March 2019.
- [I-D.ietf-core-resource-directory]
Shelby, Z., Koster, M., Bormann, C., Stok, P., and C.
Amsuess, "CoRE Resource Directory", draft-ietf-core-
resource-directory-20 (work in progress), March 2019.
- [I-D.palombini-ace-coap-pubsub-profile]
Palombini, F., "CoAP Pub-Sub Profile for Authentication
and Authorization for Constrained Environments (ACE)",
draft-palombini-ace-coap-pubsub-profile-03 (work in
progress), June 2018.
- [RFC5988] Nottingham, M., "Web Linking", RFC 5988,
DOI 10.17487/RFC5988, October 2010,
<<https://www.rfc-editor.org/info/rfc5988>>.
- [RFC8387] Sethi, M., Arkko, J., Keranen, A., and H. Back, "Practical
Considerations and Implementation Experiences in Securing
Smart Object Networks", RFC 8387, DOI 10.17487/RFC8387,
May 2018, <<https://www.rfc-editor.org/info/rfc8387>>.

Authors' Addresses

Michael Koster
SmartThings

Email: Michael.Koster@smarththings.com

Ari Keranen
Ericsson

Email: ari.keranen@ericsson.com

Jaime Jimenez
Ericsson

Email: jaime.jimenez@ericsson.com

CoRE
Internet-Draft
Intended status: Standards Track
Expires: January 10, 2020

M. Veillette, Ed.
Trilliant Networks Inc.
P. van der Stok, Ed.
consultant
A. Pelov
Acklio
A. Bierman
YumaWorks
I. Petrov, Ed.
Acklio
July 09, 2019

CoAP Management Interface
draft-ietf-core-comi-06

Abstract

This document describes a network management interface for constrained devices and networks, called CoAP Management Interface (CoMI). The Constrained Application Protocol (CoAP) is used to access datastore and data node resources specified in YANG, or SMIV2 converted to YANG. CoMI uses the YANG to CBOR mapping and converts YANG identifier strings to numeric identifiers for payload size reduction. The complete solution composed of CoMI, [I-D.ietf-core-yang-cbor] and [I-D.ietf-core-sid] is called CORECONF. CORECONF extends the set of YANG based protocols, NETCONF and RESTCONF, with the capability to manage constrained devices and networks.

Note

Discussion and suggestions for improvement are requested, and should be sent to yot@ietf.org.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any

time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on January 10, 2020.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

| | |
|--|----|
| 1. Introduction | 3 |
| 1.1. Terminology | 4 |
| 2. CoMI Architecture | 5 |
| 2.1. Major differences between RESTCONF and CoMI | 6 |
| 2.1.1. Differences due to CoAP and its efficient usage | 6 |
| 2.1.2. Differences due to the use of CBOR | 7 |
| 2.2. Compression of YANG identifiers | 7 |
| 2.3. Instance identifier | 8 |
| 2.4. Content-Formats | 8 |
| 2.5. Unified datastore | 10 |
| 3. Example syntax | 11 |
| 4. CoAP Interface | 11 |
| 4.1. Using the 'k' Uri-Query option | 13 |
| 4.2. Data Retrieval | 14 |
| 4.2.1. Using the 'c' Uri-Query option | 14 |
| 4.2.2. Using the 'd' Uri-Query option | 15 |
| 4.2.3. GET | 16 |
| 4.2.4. FETCH | 18 |
| 4.3. Data Editing | 19 |
| 4.3.1. Data Ordering | 19 |
| 4.3.2. POST | 19 |
| 4.3.3. PUT | 20 |
| 4.3.4. iPATCH | 21 |
| 4.3.5. DELETE | 22 |
| 4.4. Full datastore access | 23 |
| 4.4.1. Full datastore examples | 23 |

| | | |
|-------------|---|----|
| 4.5. | Event stream | 24 |
| 4.5.1. | Notify Examples | 25 |
| 4.5.2. | The 'f' Uri-Query option | 26 |
| 4.6. | RPC statements | 27 |
| 4.6.1. | RPC Example | 27 |
| 5. | Use of Block-wise Transfers | 29 |
| 6. | Application Discovery | 29 |
| 6.1. | YANG library | 29 |
| 6.2. | Resource Discovery | 30 |
| 6.2.1. | Datastore Resource Discovery | 30 |
| 6.2.2. | Data node Resource Discovery | 30 |
| 6.2.3. | Event stream Resource Discovery | 31 |
| 7. | Error Handling | 31 |
| 8. | Security Considerations | 35 |
| 9. | IANA Considerations | 35 |
| 9.1. | Resource Type (rt=) Link Target Attribute Values Registry | 35 |
| 9.2. | CoAP Content-Formats Registry | 36 |
| 9.3. | Media Types Registry | 36 |
| 10. | Acknowledgements | 38 |
| 11. | References | 38 |
| 11.1. | Normative References | 38 |
| 11.2. | Informative References | 40 |
| Appendix A. | ietf-comi YANG module | 40 |
| Appendix B. | ietf-comi .sid file | 46 |
| Authors' | Addresses | 49 |

1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] is designed for Machine to Machine (M2M) applications such as smart energy, smart city and building control. Constrained devices need to be managed in an automatic fashion to handle the large quantities of devices that are expected in future installations. Messages between devices need to be as small and infrequent as possible. The implementation complexity and runtime resources need to be as small as possible.

This draft describes the CoAP Management Interface which uses CoAP methods to access structured data defined in YANG [RFC7950]. This draft is complementary to [RFC8040] which describes a REST-like interface called RESTCONF, which uses HTTP methods to access structured data defined in YANG.

The use of standardized data models specified in a standardized language, such as YANG, promotes interoperability between devices and applications from different manufacturers.

CoMI and RESTCONF are intended to work in a stateless client-server fashion. They use a single round-trip to complete a single editing transaction, where NETCONF needs multiple round trips.

To promote small messages, CORECONF uses a YANG to CBOR mapping [I-D.ietf-core-yang-cbor] and numeric identifiers [I-D.ietf-core-sid] to minimize CBOR payloads and URI length.

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 BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

The following terms are defined in the YANG data modelling language [RFC7950]: action, anydata, anyxml, client, container, data model, data node, identity, instance identifier, leaf, leaf-list, list, module, RPC, schema node, server, submodule.

The following terms are defined in [RFC6241]: configuration data, datastore, state data

The following term is defined in [I-D.ietf-core-sid]: YANG schema item identifier (SID).

The following terms are defined in the CoAP protocol [RFC7252]: Confirmable Message, Content-Format, Endpoint.

The following terms are defined in this document:

data node resource: a CoAP resource that models a YANG data node.

datastore resource: a CoAP resource that models a YANG datastore.

event stream resource: a CoAP resource used by clients to observe YANG notifications.

notification instance: An instance of a schema node of type notification, specified in a YANG module implemented by the server. The instance is generated in the server at the occurrence of the corresponding event and reported by an event stream.

list instance identifier: Handle used to identify a YANG data node that is an instance of a YANG "list" specified with the values of the key leaves of the list.

single instance identifier: Handle used to identify a specific data node which can be instantiated only once. This includes data nodes defined at the root of a YANG module and data nodes defined within a container. This excludes data nodes defined within a list or any children of these data nodes.

instance-identifier: List instance identifier or single instance identifier.

instance-value: The value assigned to a schema node instance. Schema node values are serialized into the payload according to the rules defined in section 4 of [I-D.ietf-core-yang-cbor].

2. CoMI Architecture

This section describes the CoMI architecture to use CoAP for reading and modifying the content of datastore(s) used for the management of the instrumented node.

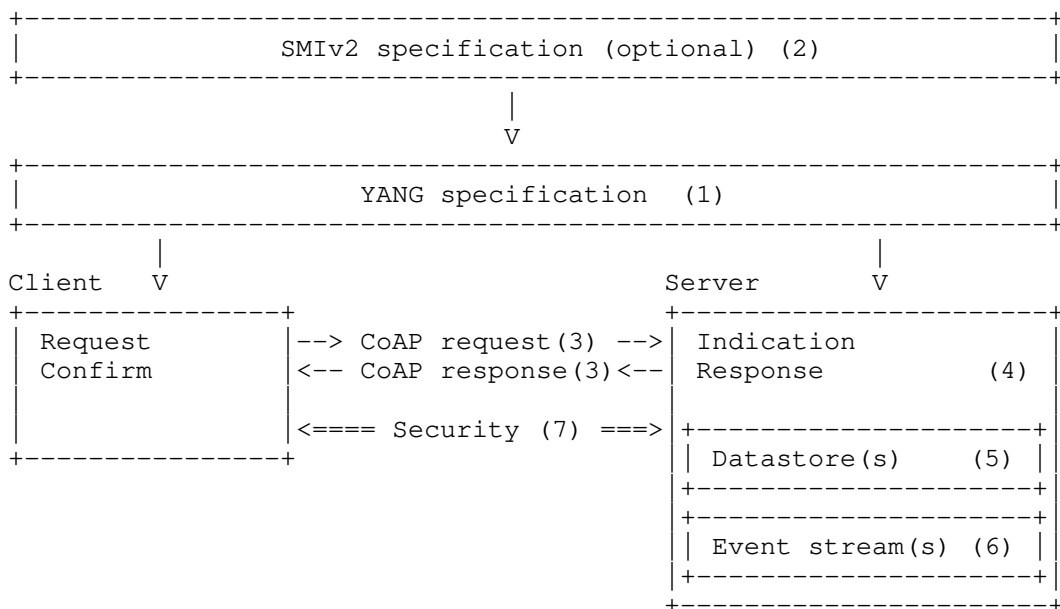


Figure 1: Abstract CoMI architecture

Figure 1 is a high-level representation of the main elements of the CoMI management architecture. The different numbered components of Figure 1 are discussed according to component number.

- (1) YANG specification: contains a set of named and versioned modules.
- (2) SMIV2 specification: Optional part that consists of a named module which, specifies a set of variables and "conceptual tables". There is an algorithm to translate SMIV2 specifications to YANG specifications.
- (3) CoAP request/response messages: The CoMI client sends request messages to and receives response messages from the CoMI server.
- (4) Request, Indication, Response, Confirm: Processes performed by the CoMI clients and servers.
- (5) Datastore: A resource used to access configuration data, state data, RPCs and actions. A CoMI server may support a single unified datastore or multiple datastores as those defined by Network Management Datastore Architecture (NMDA) [RFC8342].
- (6) Event stream: A resource used to get real time notifications. A CoMI server may support multiple Event streams serving different purposes such as normal monitoring, diagnostic, syslog, security monitoring.
- (7) Security: The server MUST prevent unauthorized users from reading or writing any CoMI resources. CoMI relies on security protocols such as DTLS [RFC6347] to secure CoAP communications.

2.1. Major differences between RESTCONF and CoMI

CoMI is a RESTful protocol for small devices where saving bytes to transport counts. Contrary to RESTCONF, many design decisions are motivated by the saving of bytes. Consequently, CoMI is not a RESTCONF over CoAP protocol, but differs more significantly from RESTCONF.

2.1.1. Differences due to CoAP and its efficient usage

- o CoMI uses CoAP/UDP as transport protocol and CBOR as payload format [I-D.ietf-core-yang-cbor]. RESTCONF uses HTTP/TCP as transport protocol and JSON or XML as payload formats.
- o CoMI uses the methods FETCH and iPATCH to access multiple data nodes. RESTCONF uses instead the HTTP method PATCH and the HTTP method GET with the "fields" Query parameter.
- o RESTCONF uses the HTTP methods HEAD, and OPTIONS, which are not supported by CoAP.

- o CoMI does not support "insert" query parameter (first, last, before, after) and the "point" query parameter which are supported by RESTCONF.
- o CoMI does not support the "start-time" and "stop-time" query parameters to retrieve past notifications.
- o CoMI does not support the "filter" query parameters to observe a specific set of notifications.

2.1.2. Differences due to the use of CBOR

- o CoMI encodes YANG identifier strings as numbers, where RESTCONF does not.
- o CoMI also differ in the handling of default values, only 'report-all' and 'trip' options are supported.

2.2. Compression of YANG identifiers

In the YANG specification, items are identified with a name string. In order to significantly reduce the size of identifiers used in CoMI, numeric identifiers are used instead of these strings. YANG Schema Item iDentifier (SID) is defined in [I-D.ietf-core-yang-cbor] section 2.1.

When used in a URI, SIDs are encoded in base64 using the URL and Filename safe alphabet as defined by [RFC4648] section 5, without padding. The last 6 bits encoded is always aligned with the least significant 6 bits of the SID represented using an unsigned integer. 'A' characters (value 0) at the start of the resulting string are removed.

```
SID in base64 = URLsafeChar[SID >> 60 & 0x3F] |
                URLsafeChar[SID >> 54 & 0x3F] |
                URLsafeChar[SID >> 48 & 0x3F] |
                URLsafeChar[SID >> 42 & 0x3F] |
                URLsafeChar[SID >> 36 & 0x3F] |
                URLsafeChar[SID >> 30 & 0x3F] |
                URLsafeChar[SID >> 24 & 0x3F] |
                URLsafeChar[SID >> 18 & 0x3F] |
                URLsafeChar[SID >> 12 & 0x3F] |
                URLsafeChar[SID >> 6 & 0x3F] |
                URLsafeChar[SID & 0x3F]
```

For example, SID 1721 is encoded as follow.


```
URLsafeChar[1721 >> 60 & 0x3F] = URLsafeChar[0] = 'A'
URLsafeChar[1721 >> 54 & 0x3F] = URLsafeChar[0] = 'A'
URLsafeChar[1721 >> 48 & 0x3F] = URLsafeChar[0] = 'A'
URLsafeChar[1721 >> 42 & 0x3F] = URLsafeChar[0] = 'A'
URLsafeChar[1721 >> 36 & 0x3F] = URLsafeChar[0] = 'A'
URLsafeChar[1721 >> 30 & 0x3F] = URLsafeChar[0] = 'A'
URLsafeChar[1721 >> 24 & 0x3F] = URLsafeChar[0] = 'A'
URLsafeChar[1721 >> 18 & 0x3F] = URLsafeChar[0] = 'A'
URLsafeChar[1721 >> 12 & 0x3F] = URLsafeChar[0] = 'A'
URLsafeChar[1721 >> 6 & 0x3F]   = URLsafeChar[26] = 'a'
URLsafeChar[1721 & 0x3F]       = URLsafeChar[57] = '5'
```

The resulting base64 representation of SID 1721 is "a5"

2.3. Instance identifier

Instance identifiers are used to uniquely identify data node instances within a datastore. This YANG built-in type is defined in [RFC7950] section 9.13. An instance identifier is composed of the data node identifier (i.e. a SID) and for data nodes within list(s) the keys used to index within these list(s).

When part of a payload, instance identifiers are encoded in CBOR based on the rules defined in [I-D.ietf-core-yang-cbor] section 6.13.1. When part of a URI, the SID is appended to the URI of the targeted datastore, the keys are specified using the 'k' URI-Query as defined in Section 4.1.

2.4. Content-Formats

CoMI uses Content-Formats based on the YANG to CBOR mapping specified in [I-D.ietf-core-yang-cbor].

The following Content-formats are defined:

application/yang-data+cbor: This Content-Format represents a CBOR YANG document containing one or multiple data node values. Each data node is identified by its associated SID.

FORMAT: CBOR map of SID, instance-value

The message payload of Content-Format 'application/yang-data+cbor' is encoded using a CBOR map. Each entry of this CBOR map is composed of a key and a value. CBOR map keys are set to the SID or SID deltas associated with the data nodes as defined in [I-D.ietf-core-yang-cbor] section 4, CBOR map values are set to the instance value as defined in [I-D.ietf-core-yang-cbor] section 4.

`application/yang-identifiers+cbor`: This Content-Format represents a CBOR YANG document containing a list of instance identifier used to target specific data node instances within a datastore.

FORMAT: CBOR array of instance-identifier

The message payload of Content-Format '`application/yang-identifiers+cbor`' is encoded using a CBOR array. Each entry of this CBOR array contain an instance identifier encoded as defined in [I-D.ietf-core-yang-cbor] section 6.13.1.

`application/yang-instances+cbor`: This Content-Format represents a CBOR YANG document containing a list of data node instances. Each data node instance is identified by its associated instance identifier.

FORMAT: CBOR array of CBOR map of instance-identifier, instance-value

The message payload of Content-Format '`application/yang-instances+cbor`' is encoded using a CBOR array. Each entry within this CBOR array contains a CBOR map carrying an instance identifier and associated instance value. Instance identifiers are encoded using the rules defined in [I-D.ietf-core-yang-cbor] section 6.13.1, values are encoded using the rules defined in [I-D.ietf-core-yang-cbor] section 4.

When present in an iPATCH request payload, this Content-Format carry a list of data node instances to be replaced, created, or deleted. For each data node instance D, for which the instance identifier is the same as a data node instance I, in the targeted datastore resource: the value of D replaces the value of I. When the value of D is null, the data node instance I is removed. When the targeted datastore resource does not contain a data node instance with the same instance identifier as D, a new instance is created with the same instance identifier and value as D.

The different Content-format usages are summarized in the table below:

| Method | Resource | Content-Format |
|----------------|--------------|------------------------------------|
| GET response | data node | /application/yang-data+cbor |
| PUT request | data node | /application/yang-data+cbor |
| POST request | data node | /application/yang-data+cbor |
| DELETE | data node | n/a |
| GET response | datastore | /application/yang-data+cbor |
| PUT request | datastore | /application/yang-data+cbor |
| POST request | datastore | /application/yang-data+cbor |
| FETCH request | datastore | /application/yang-identifiers+cbor |
| FETCH response | datastore | /application/yang-instances+cbor |
| iPATCH request | datastore | /application/yang-instances+cbor |
| GET response | event stream | /application/yang-instances+cbor |
| POST request | rpc, action | /application/yang-data+cbor |
| POST response | rpc, action | /application/yang-data+cbor |

2.5. Unified datastore

CoMI supports a simple datastore model consisting of a single unified datastore. This datastore provides access to both configuration and operational data. Configuration updates performed on this datastore are reflected immediately or with a minimal delay as operational data.

Alternatively, CoMI servers MAY implement a more complex datastore model such as the Network Management Datastore Architecture (NMDA) as defined by [RFC8342]. Each datastore supported is implemented as a datastore resource.

Characteristics of the unified datastore are summarized in the table below:

| Name | Value |
|--------------|--|
| Name | unified |
| YANG modules | all modules |
| YANG nodes | all data nodes ("config true" and "config false") |
| Access | read-write |
| How applied | changes applied in place immediately or with a minimal delay |
| Protocols | CORECONF |
| Defined in | "ietf-comi" |

3. Example syntax

CBOR is used to encode CoMI request and response payloads. The CBOR syntax of the YANG payloads is specified in [RFC7049]. The payload examples are notated in Diagnostic notation (defined in section 6 of [RFC7049]) that can be automatically converted to CBOR.

SIDs in URIs are represented as a base64 number, SIDs in the payload are represented as decimal numbers.

4. CoAP Interface

This note specifies a Management Interface. CoAP endpoints that implement the CoMI management protocol, support at least one discoverable management resource of resource type (rt): core.c.ds, with example path: /c, where c is short-hand for CoMI. The path /c is recommended, but not compulsory (see Section 6).

The mapping of YANG data node instances to CoMI resources is as follows. Every data node of the YANG modules loaded in the CoMI server represents a sub-resource of the datastore resource (e.g. /c/sid). When multiple instances of a list exist, instance selection is possible as described in Section 4.1, Section 4.2.3.1, and Section 4.2.4.

CoMI also supports event stream resources used to observe notification instances. Event stream resources can be discovered using resource type (rt): core.c.ev.

The description of the CoMI management interface is shown in the table below:

| Function | Recommended path | rt |
|--------------|------------------|-----------|
| Datastore | /c | core.c.ds |
| Data node | /c/SID | core.c.dn |
| Event stream | /s | core.c.ev |

The path values in the table are the recommended ones. On discovery, the server makes the actual path values known for these resources.

The methods used by CoMI are:

| Operation | Description |
|-----------|---|
| GET | Retrieve the datastore resource or a data node resource |
| FETCH | Retrieve specific data nodes within a datastore resource |
| POST | Create a datastore resource or a data node resource, invoke an RPC or action |
| PUT | Create or replace a datastore resource or a data node resource |
| iPATCH | Idem-potently create, replace, and delete data node resource(s) within a datastore resource |
| DELETE | Delete a datastore resource or a data node resource |

There is one Uri-Query option for the GET, PUT, POST, and DELETE methods.

| Uri-Query option | Description |
|------------------|--|
| k | Select an instance within YANG list(s) |

This parameter is not used for FETCH and iPATCH, because their request payloads support list instance selection.

4.1. Using the 'k' Uri-Query option

The "k" (key) parameter specifies a specific instance of a data node. The SID in the URI is followed by the (?k=key1,key2,...). Where SID identifies a data node, and key1, key2 are the values of the key leaves that specify an instance. Lists can have multiple keys, and lists can be part of lists. The order of key value generation is given recursively by:

- o For a given list, if a parent data node is a list, generate the keys for the parent list first.
- o For a given list, generate key values in the order specified in the YANG module.

Key values are encoded using the rules defined in the following table.

| YANG datatype | Uri-Query text content |
|-----------------------------|--------------------------------|
| uint8,uint16,uint32, uint64 | int2str(key) |
| int8, int16,int32, int64 | urlSafeBase64(CBORencode(key)) |
| decimal64 | urlSafeBase64(CBOR key) |
| string | key |
| boolean | "0" or "1" |
| enumeration | int2str(key) |
| bits | urlSafeBase64(CBORencode(key)) |
| binary | urlSafeBase64(key) |
| identityref | int2str(key) |
| union | urlSafeBase64(CBORencode(key)) |
| instance-identifier | urlSafeBase64(CBORencode(key)) |

In this table:

- o The method `int2str()` is used to convert an integer value to a decimal string. For example, `int2str(0x0123)` return the string "291".
- o The method `urlSafeBase64()` is used to convert a binary string to base64 using the URL and Filename safe alphabet as defined by [RFC4648] section 5, without padding. For example, `urlSafeBase64(\xF9\x56\xA1\x3C)` return the string "-VahPA".
- o The method `CBORencode()` is used to convert a YANG value to CBOR as specified in [I-D.ietf-core-yang-cbor] section 6.

The resulting key string is encoded in a Uri-Query as specified in [RFC7252] section 6.5.

4.2. Data Retrieval

One or more data nodes can be retrieved by the client. The operation is mapped to the GET method defined in section 5.8.1 of [RFC7252] and to the FETCH method defined in section 2 of [RFC8132].

There are two additional Uri-Query options for the GET and FETCH methods.

| Uri-Query option | Description |
|------------------|---|
| c | Control selection of configuration and non-configuration data nodes (GET and FETCH) |
| d | Control retrieval of default values. |

4.2.1. Using the 'c' Uri-Query option

The 'c' (content) option controls how descendant nodes of the requested data nodes will be processed in the reply.

The allowed values are:

| Value | Description |
|-------|---|
| c | Return only configuration descendant data nodes |
| n | Return only non-configuration descendant data nodes |
| a | Return all descendant data nodes |

This option is only allowed for GET and FETCH methods on datastore and data node resources. A 4.02 (Bad Option) error is returned if used for other methods or resource types.

If this Uri-Query option is not present, the default value is "a".

4.2.2. Using the 'd' Uri-Query option

The "d" (with-defaults) option controls how the default values of the descendant nodes of the requested data nodes will be processed.

The allowed values are:

| Value | Description |
|-------|---|
| a | All data nodes are reported. Defined as 'report-all' in section 3.1 of [RFC6243]. |
| t | Data nodes set to the YANG default are not reported. Defined as 'trim' in section 3.2 of [RFC6243]. |

If the target of a GET or FETCH method is a data node that represents a leaf that has a default value, and the leaf has not been given a value by any client yet, the server MUST return the default value of the leaf.

If the target of a GET method is a data node that represents a container or list that has child resources with default values, and these have not been given value yet,

The server MUST NOT return the child resource if d= 't'

The server MUST return the child resource if d= 'a'.

If this Uri-Query option is not present, the default value is 't'.

4.2.3. GET

A request to read the values of a data node instance is sent with a CoAP GET message. A base64-encoded instance-identifier in SID-form is specified in the URI path prefixed with the example path /c.

FORMAT:

GET /c/instance-identifier

2.05 Content (Content-Format: application/yang-data+cbor)
CBOR map of SID, instance-value

The returned payload contains the CBOR encoding of the specified data node instance value.

4.2.3.1. GET Examples

Using for example the current-datetime leaf from module ietf-system [RFC7317], a request is sent to retrieve the value of 'system-state/clock/current-datetime' specified in container system-state. The SID of 'system-state/clock/current-datetime' is 1723, encoded in base64 according to Section 2.2, yields a7. The response to the request returns the CBOR map with the key set to the SID of the requested data node (i.e. 1723) and the value encoded using a 'text string' as defined in [I-D.ietf-core-yang-cbor] section 6.4.

REQ: GET example.com/c/a7

RES: 2.05 Content (Content-Format: application/yang-data+cbor)
{
 1723 : "2014-10-26T12:16:31Z"
}

The next example represents the retrieval of a YANG container. In this case, the CoMI client performs a GET request on the clock container (SID = 1721; base64: a5). The container returned is encoded using a CBOR map as specified by [I-D.ietf-core-yang-cbor] section 4.2.

REQ: GET example.com/c/a5

RES: 2.05 Content (Content-Format: application/yang-data+cbor)
{
 1721 : {
 +2 : "2014-10-26T12:16:51Z", / current-datetime SID 1723 /
 +1 : "2014-10-21T03:00:00Z" / boot-datetime SID 1722 /
 }
}

This example shows the retrieval of the /interfaces/interface YANG list accessed using SID 1533 (base64: X9). The return payload is encoded using a CBOR array as specified by [I-D.ietf-core-yang-cbor] section 4.4.1 containing 2 instances.

REQ: GET example.com/c/X9

RES: 2.05 Content (Content-Format: application/yang-data+cbor)

```
{
  1533 : [
    {
      +4 : "eth0",           / name (SID 1537) /
      +1 : "Ethernet adaptor", / description (SID 1534) /
      +5 : 1880,             / type, (SID 1538) identity /
                           / ethernetCsmacd (SID 1880) /
      +2 : true              / enabled ( SID 1535) /
    },
    {
      +4 : "eth1",           / name (SID 1537) /
      +1 : "Ethernet adaptor", / description (SID 1534) /
      +5 : 1880,             / type, (SID 1538) identity /
                           / ethernetCsmacd (SID 1880) /
      +2 : false             / enabled ( SID 1535) /
    }
  ]
}
```

To retrieve a specific instance within the /interfaces/interface YANG list, the CoMI client adds the key of the targeted instance in its CoAP request using the 'k' URI-Query. The return payload containing the instance requested is encoded using a CBOR array as specified by [I-D.ietf-core-yang-cbor] section 4.4.1.

REQ: GET example.com/c/X9?k="eth0"

RES: 2.05 Content (Content-Format: application/yang-data+cbor)

```
{
  1533 : [
    {
      +4 : "eth0",           / name (SID 1537) /
      +1 : "Ethernet adaptor", / description (SID 1534) /
      +5 : 1880,             / type, (SID 1538) identity /
                           / ethernetCsmacd (SID 1880) /
      +2 : true              / enabled ( SID 1535) /
    }
  ]
}
```

It is equally possible to select a leaf of a specific instance of a list. The example below requests the description leaf (SID=1534, base64: X-) within the interface list corresponding to the interface name "eth0". The returned value is encoded in CBOR based on the rules specified by [I-D.ietf-core-yang-cbor] section 6.4.

REQ: GET example.com/c/X-?k="eth0"

RES: 2.05 Content (Content-Format: application/yang-data+cbor)
{
 1534 : "Ethernet adaptor"
}

4.2.4. FETCH

The FETCH is used to retrieve multiple data node instance values. The FETCH request payload contains the list of instance identifier of the data node instances requested.

The return response payload contains a list of data node instance values in the same order as requested. A CBOR null is returned for each data node requested by the client, not supported by the server or not currently instantiated.

For compactness, indexes of the list instance identifiers returned by the FETCH response SHOULD be elided, only the SID is provided. In this case, the format of each entry within the CBOR array of the FETCH response is identical to the format as a GET response.

FORMAT:

FETCH /c (Content-Format: application/yang-identifiers+cbor)
CBOR array of instance-identifier

2.05 Content (Content-Format: application/yang-instances+cbor)
CBOR array of CBOR map of instance-identifier, instance-value

4.2.4.1. FETCH examples

This example uses the current-datetime leaf from module ietf-system [RFC7317] and the interface list from module ietf-interfaces [RFC7223]. In this example the value of current-datetime (SID 1723) and the interface list (SID 1533) instance identified with name="eth0" are queried.

```
REQ:  FETCH /c (Content-Format: application/yang-identifiers+cbor)
[
  1723,                / current-datetime (SID 1723) /
  [1533, "eth0"]        / interface (SID 1533) with name = "eth0" /
]

RES:  2.05 Content (Content-Format: application/yang-instances+cbor)
[
  {
    1723 : "2014-10-26T12:16:31Z" / current-datetime (SID 1723) /
  },
  {
    1533 : {
      +4 : "eth0",          / name (SID 1537) /
      +1 : "Ethernet adaptor", / description (SID 1534) /
      +5 : 1880,            / type (SID 1538), identity /
                           / ethernetCsmacd (SID 1880) /
      +2 : true             / enabled (SID 1535) /
    }
  }
]
```

4.3. Data Editing

CoMI allows datastore contents to be created, modified and deleted using CoAP methods.

4.3.1. Data Ordering

A CoMI server SHOULD preserve the relative order of all user-ordered list and leaf-list entries that are received in a single edit request. These YANG data node types are encoded as CBOR arrays so messages will preserve their order.

4.3.2. POST

The CoAP POST operation is used in CoMI for creation of data node resources and the invocation of "ACTION" and "RPC" resources. Refer to Section 4.6 for details on "ACTION" and "RPC" resources.

A request to create a data node resource is sent with a CoAP POST message. The URI specifies the data node to be instantiated at the exception of list instances. In this case, for compactness, the URI specifies the list for which an instance is created.

FORMAT:

```
POST /c/<instance identifier>
(Content-Format: application/yang-data+cbor)
CBOR map of SID, instance-value
```

2.01 Created

If the data node resource already exists, then the POST request MUST fail and a "4.09 Conflict" response code MUST be returned

4.3.2.1. Post example

The example uses the interface list from module ietf-interfaces [RFC7223]. This example creates a new list instance within the interface list (SID = 1533):

```
REQ: POST /c/X9 (Content-Format: application/yang-data+cbor)
{
  1533 : [
    {
      +4 : "eth5",           / name (SID 1537) /
      +1 : "Ethernet adaptor", / description (SID 1534) /
      +5 : 1880,             / type (SID 1538), identity /
                               / ethernetCsmacd (SID 1880) /
      +2 : true               / enabled (SID 1535) /
    }
  ]
}
```

RES: 2.01 Created

4.3.3. PUT

A data node resource instance is created or replaced with the PUT method. A request to set the value of a data node instance is sent with a CoAP PUT message.

FORMAT:

```
PUT /c/<instance identifier>
(Content-Format: application/yang-data+cbor)
CBOR map of SID, instance-value
```

2.01 Created

4.3.3.1. PUT example

The example uses the interface list from module ietf-interfaces [RFC7223]. Example updates the instance of the list interface (SID = 1533) with key name="eth0":

```
REQ: PUT /c/X9?k="eth0" (Content-Format: application/yang-data+cbor)
{
  1533 : [
    {
      +4 : "eth0",           / name (SID 1537) /
      +1 : "Ethernet adaptor", / description (SID 1534) /
      +5 : 1880,             / type (SID 1538), identity /
                           / ethernetCsmacd (SID 1880) /
      +2 : true              / enabled (SID 1535) /
    }
  ]
}
```

RES: 2.04 Changed

4.3.4. iPATCH

One or multiple data node instances are replaced with the idempotent CoAP iPATCH method [RFC8132].

There are no Uri-Query options for the iPATCH method.

The processing of the iPATCH command is specified by Content-Format 'application/yang-instances+cbor'. In summary, if the CBOR patch payload contains a data node instance that is not present in the target, this instance is added. If the target contains the specified instance, the content of this instance is replaced with the value of the payload. A null value indicates the removal of an existing data node instance.

FORMAT:

```
iPATCH /c (Content-Format: application/yang-instances+cbor)
CBOR array of CBOR map of instance-identifier, instance-value
```

2.04 Changed

4.3.4.1. iPATCH example

In this example, a CoMI client requests the following operations:

- o Set "/system/ntp/enabled" (SID 1755) to true.

- o Remove the server "tac.nrc.ca" from the "/system/ntp/server" (SID 1756) list.
- o Add/set the server "NTP Pool server 2" to the list "/system/ntp/server" (SID 1756).

REQ: iPATCH /c (Content-Format: application/yang-instances+cbor)

```
[
  {
    1755 : true                / enabled (SID 1755) /
  },
  {
    [1756, "tac.nrc.ca"] : null / server (SID 1756) /
  },
  {
    1756 : {
      +3 : "tic.nrc.ca",      / name (SID 1759) /
      +4 : true,              / prefer (SID 1760) /
      +5 : {
        +1 : "132.246.11.231" / address (SID 1762) /
      }
    }
  }
]
```

RES: 2.04 Changed

4.3.5. DELETE

A data node resource is deleted with the DELETE method.

FORMAT:

Delete /c/<instance identifier>

2.02 Deleted

4.3.5.1. DELETE example

This example uses the interface list from module ietf-interfaces [RFC7223]. This example deletes an instance of the interface list (SID = 1533):

REQ: DELETE /c/X9?k="eth0"

RES: 2.02 Deleted

4.4. Full datastore access

The methods GET, PUT, POST, and DELETE can be used to request, replace, create, and delete a whole datastore respectively.

FORMAT:

GET /c

2.05 Content (Content-Format: application/yang-data+cbor)
CBOR map of SID, instance-value

FORMAT:

PUT /c (Content-Format: application/yang-data+cbor)
CBOR map of SID, instance-value

2.04 Changed

FORMAT:

POST /c (Content-Format: application/yang-data+cbor)
CBOR map of SID, instance-value

2.01 Created

FORMAT:

DELETE /c

2.02 Deleted

The content of the CBOR map represents the complete datastore of the server at the GET indication of after a successful processing of a PUT or POST request.

4.4.1. Full datastore examples

The example uses the interface list from module ietf-interfaces [RFC7223] and the clock container from module ietf-system [RFC7317]. We assume that the datastore contains two modules ietf-system (SID 1700) and ietf-interfaces (SID 1500); they contain the 'interface' list (SID 1533) with one instance and the 'clock' container (SID 1721). After invocation of GET, a CBOR map with data nodes from these two modules is returned:

REQ: GET /c

```
RES: 2.05 Content (Content-Format: application/yang-data+cbor)
{
  1721 : {
    / Clock (SID 1721) /
    +2: "2016-10-26T12:16:31Z", / current-datetime (SID 1723) /
    +1: "2014-10-05T09:00:00Z" / boot-datetime (SID 1722) /
  },
  1533 : [
    {
      / interface (SID 1533) /
      +4 : "eth0", / name (SID 1537) /
      +1 : "Ethernet adaptor", / description (SID 1534) /
      +5 : 1880, / type (SID 1538), identity: /
      / ethernetCsmacd (SID 1880) /
      +2 : true / enabled (SID 1535) /
    }
  ]
}
```

4.5. Event stream

Event notification is an essential function for the management of servers. CoMI allows notifications specified in YANG [RFC5277] to be reported to a list of clients. The recommended path of the default event stream is /s. The server MAY support additional event stream resources to address different notification needs.

Reception of notification instances is enabled with the CoAP Observe [RFC7641] function. Clients subscribe to the notifications by sending a GET request with an "Observe" option, specifying the /s resource when the default stream is selected.

Each response payload carries one or multiple notifications. The number of notifications reported, and the conditions used to remove notifications from the reported list is left to implementers. When multiple notifications are reported, they MUST be ordered starting from the newest notification at index zero.

The format of notification contents is defined in [I-D.ietf-core-yang-cbor] section 4.2.1. For notification without any content, a null value is returned.

An example implementation is:

Every time an event is generated, the generated notification instance is appended to the chosen stream(s). After an aggregation period, which may be adjusted using an exclusion delay and limited by the maximum number of notifications supported, the

content of the instance is sent to all clients observing the modified stream.

FORMAT:

GET /<stream-resource> Observe(0)

2.05 Content (Content-Format: application/yang-instances+cbor)
CBOR array of CBOR map of instance-identifier, instance-value

The array of data node instances may contain identical entries which have been generated at different times.

4.5.1. Notify Examples

Let suppose the server generates the example-port-fault event as defined below.

```
module example-port {  
  ...  
  notification example-port-fault { // SID 60010  
    description  
      "Event generated if a hardware fault is detected";  
    leaf port-name { // SID 60011  
      type string;  
    }  
    leaf port-fault { // SID 60012  
      type string;  
    }  
  }  
}
```

By executing a GET on the /s resource the client receives the following response:

REQ: GET /s Observe(0)

```
RES: 2.05 Content (Content-Format: application/yang-tree+cbor)
      Observe(12)
[
  {
    60010 : {
      +1 : "0/4/21",      / port-name (SID 60011) /
      +2 : "Open pin 2"   / port-fault (SID 60012) /
    }
  },
  {
    60010 : {
      +1 : "1/4/21",      / port-name (SID 60011) /
      +2 : "Open pin 5"   / port-fault (SID 60012) /
    }
  }
]
```

In the example, the request returns a success response with the contents of the last two generated events. Consecutively the server will regularly notify the client when a new event is generated.

To check that the client is still alive, the server MUST send Confirmable Message periodically. When the client does not confirm the notification from the server, the server will remove the client from the list of observers [RFC7641].

4.5.2. The 'f' Uri-Query option

The 'f' (filter) option is used to indicate which subset of all possible notifications is of interest. If not present, all events notifications supported by the event stream are reported.

When not supported by a CoMI server, this option shall be ignored, all events notifications are reported independently of the presence and content of the 'f' (filter) option.

When present, this option contains a comma separated list of notification SIDs. For example, the following request returns notifications 60010 and 60020.

REQ: GET /s?f=60010,60020 Observe(0)

4.6. RPC statements

The YANG "action" and "RPC" statements specify the execution of a Remote procedure Call (RPC) in the server. It is invoked using a POST method to an "Action" or "RPC" resource instance.

The request payload contains the values assigned to the input container when specified. The response payload contains the values of the output container when specified. Both the input and output containers are encoded in CBOR using the rules defined in [I-D.ietf-core-yang-cbor] section 4.2.1.

The returned success response code is 2.05 Content.

FORMAT:

```
POST /c/<instance identifier>
      (Content-Format: application/yang-data+cbor)
CBOR map of SID, instance-value

2.05 Content (Content-Format: application/yang-data+cbor)
CBOR map of SID, instance-value
```

4.6.1. RPC Example

The example is based on the YANG action "reset" as defined in [RFC7950] section 7.15.3 and annotated below with SIDs.

```

module example-server-farm {
  yang-version 1.1;
  namespace "urn:example:server-farm";
  prefix "sfarm";

  import ietf-yang-types {
    prefix "yang";
  }

  list server {                                // SID 60000
    key name;
    leaf name {                                // SID 60001
      type string;
    }
    action reset {                             // SID 60002
      input {
        leaf reset-at {                       // SID 60003
          type yang:date-and-time;
          mandatory true;
        }
      }
      output {
        leaf reset-finished-at {             // SID 60004
          type yang:date-and-time;
          mandatory true;
        }
      }
    }
  }
}

```

This example invokes the 'reset' action (SID 60002, base64: Opq), of the server instance with name equal to "myserver".

REQ: POST /c/Opq?k="myserver"

(Content-Format: application/yang-data+cbor)

```

{
  60002 : {
    +1 : "2016-02-08T14:10:08Z09:00" / reset-at (SID 60003) /
  }
}

```

RES: 2.05 Content (Content-Format: application/yang-data+cbor)

```

{
  60002 : {
    +2 : "2016-02-08T14:10:08Z09:18" / reset-finished-at (SID 60004)/
  }
}

```

5. Use of Block-wise Transfers

The CoAP protocol provides reliability by acknowledging the UDP datagrams. However, when large pieces of data need to be transported, datagrams get fragmented, thus creating constraints on the resources in the client, server and intermediate routers. The block option [RFC7959] allows the transport of the total payload in individual blocks of which the size can be adapted to the underlying transport sizes such as: (UDP datagram size ~64KiB, IPv6 MTU of 1280, IEEE 802.15.4 payload of 60-80 bytes). Each block is individually acknowledged to guarantee reliability.

Notice that the Block mechanism splits the data at fixed positions, such that individual data fields may become fragmented. Therefore, assembly of multiple blocks may be required to process the complete data field.

Beware of race conditions. Blocks are filled one at a time and care should be taken that the whole data representation is sent in multiple blocks sequentially without interruption. On the server, values are changed, lists are re-ordered, extended or reduced. When these actions happen during the serialization of the contents of the resource, the transported results do not correspond with a state having occurred in the server; or worse the returned values are inconsistent. For example: array length does not correspond with the actual number of items. It may be advisable to use Indefinite-length CBOR arrays and maps, which are foreseen for data streaming purposes.

6. Application Discovery

Two application discovery mechanisms are supported by CoMI, the YANG library data model as defined by [I-D.veillette-core-yang-library] and the CORE resource discovery [RFC6690]. Implementers may choose to implement one or the other or both.

6.1. YANG library

The YANG library data model [I-D.veillette-core-yang-library] provides a high level description of the resources available. The YANG library contains the list of modules, features and deviations supported by the CoMI server. From this information, CoMI clients can infer the list of data nodes supported and the interaction model to be used to access them. This module also contains the list of datastores implemented.

The location of the YANG library can be found by sending a GET request to `"/.well-known/core"` including a resource type (RT)

parameter with the value "core.c.yml". Upon success, the return payload will contain the root resource of the YANG library module.

```
REQ: GET /.well-known/core?rt=core.c.yml
```

```
RES: 2.05 Content (Content-Format: application/link-format)
</c/kv>;rt="core.c.yml"
```

6.2. Resource Discovery

Even if the YANG library provides all the information needed for application discovery once it is itself discovered, other types of resources could be discovered using the implementation of Resource discovery as defined by [RFC6690]. This can be desirable for a seamless integration with other CoAP interfaces and services.

6.2.1. Datastore Resource Discovery

The presence and location of (path to) each datastore implemented by the CoMI server can be discovered by sending a GET request to `"/.well-known/core"` including a resource type (RT) parameter with the value `"core.c.ds"`.

Upon success, the return payload contains the list of datastore resources.

Each datastore returned is further qualified using the `"ds"` Link-Format attribute. This attribute is set to the SID assigned to the datastore identity. When a unified datastore is implemented, the `ds` attribute is set to 1029. For other examples of datastores, see the Network Management Datastore Architecture (NMDA) [RFC7950].

```
link-extension    = ( "ds" "=" sid ) )
                  ; SID assigned to the datastore identity
sid                = 1*DIGIT
```

For example:

```
REQ: GET /.well-known/core?rt=core.c.ds
```

```
RES: 2.05 Content (Content-Format: application/link-format)
</c>; rt="core.c.ds";ds= 1029
```

6.2.2. Data node Resource Discovery

The presence and location of (path to) each data node implemented by the CoMI server are discovered by sending a GET request to `"/.well-`

known/core" including a resource type (RT) parameter with the value "core.c.dn".

Upon success, the return payload contains the SID assigned to each data node and their location.

The example below shows the discovery of the presence and location of data nodes. Data nodes '/ietf-system:system-state/clock/boot-datetime' (SID 1722) and '/ietf-system:system-state/clock/current-datetime' (SID 1723) are returned.

```
REQ: GET /.well-known/core?rt=core.c.dn
```

```
RES: 2.05 Content (Content-Format: application/link-format)
</c/a6>;rt="core.c.dn",
</c/a7>;rt="core.c.dn"
```

Without additional filtering, the list of data nodes may become prohibitively long. Implementations MAY return a subset of this list or can rely solely on the YANG library.

6.2.3. Event stream Resource Discovery

The presence and location of (path to) each event stream implemented by the CoMI server are discovered by sending a GET request to "/.well-known/core" including a resource type (RT) parameter with the value "core.c.es".

Upon success, the return payload contains the list of event stream resources.

For example:

```
REQ: GET /.well-known/core?rt=core.c.es
```

```
RES: 2.05 Content (Content-Format: application/link-format)
</s>;rt="core.c.es"
```

7. Error Handling

In case a request is received which cannot be processed properly, the CoMI server MUST return an error response. This error response MUST contain a CoAP 4.xx or 5.xx response code.

Errors returned by a CoMI server can be broken into two categories, those associated to the CoAP protocol itself and those generated during the validation of the YANG data model constraints as described in [RFC7950] section 8.

The following list of common CoAP errors should be implemented by CoMI servers. This list is not exhaustive, other errors defined by CoAP and associated RFCs may be applicable.

- o Error 4.01 (Unauthorized) is returned by the CoMI server when the CoMI client is not authorized to perform the requested action on the targeted resource (i.e. data node, datastore, rpc, action or event stream).
- o Error 4.02 (Bad Option) is returned by the CoMI server when one or more CoAP options are unknown or malformed.
- o Error 4.04 (Not Found) is returned by the CoMI server when the CoMI client is requesting a non-instantiated resource (i.e. data node, datastore, rpc, action or event stream).
- o Error 4.05 (Method Not Allowed) is returned by the CoMI server when the CoMI client is requesting a method not supported on the targeted resource. (e.g. GET on an rpc, PUT or POST on a data node with "config" set to false).
- o Error 4.08 (Request Entity Incomplete) is returned by the CoMI server if one or multiple blocks of a block transfer request is missing, see [RFC7959] for more details.
- o Error 4.13 (Request Entity Too Large) may be returned by the CoMI server during a block transfer request, see [RFC7959] for more details.
- o Error 4.15 (Unsupported Content-Format) is returned by the CoMI server when the Content-Format used in the request does not match those specified in section Section 2.4.

CoMI server MUST also enforce the different constraints associated to the YANG data models implemented. These constraints are described in [RFC7950] section 8. These errors are reported using the CoAP error code 4.00 (Bad Request) and may have the following error container as payload. The YANG definition and associated .sid file are available in Appendix A and Appendix B. The error container is encoded using the encoding rules of a YANG data template as defined in [I-D.ietf-core-yang-cbor] section 5.

```
+--rw error!
  +--rw error-tag          identityref
  +--rw error-app-tag?     identityref
  +--rw error-data-node?   instance-identifier
  +--rw error-message?     string
```

The following 'error-tag' and 'error-app-tag' are defined by the ietf-comi YANG module, these tags are implemented as YANG identity and can be extended as needed.

- o error-tag 'operation-failed' is returned by the CoMI server when the operation request cannot be processed successfully.
 - * error-app-tag 'malformed-message' is returned by the CoMI server when the payload received from the CoMI client does not contain a well-formed CBOR content as defined in [RFC7049] section 3.3 or does not comply with the CBOR structure defined within this document.
 - * error-app-tag 'data-not-unique' is returned by the CoMI server when the validation of the 'unique' constraint of a list or leaf-list fails.
 - * error-app-tag 'too-many-elements' is returned by the CoMI server when the validation of the 'max-elements' constraint of a list or leaf-list fails.
 - * error-app-tag 'too-few-elements' is returned by the CoMI server when the validation of the 'min-elements' constraint of a list or leaf-list fails.
 - * error-app-tag 'must-violation' is returned by the CoMI server when the restrictions imposed by a 'must' statement are violated.
 - * error-app-tag 'duplicate' is returned by the CoMI server when a client tries to create a duplicate list or leaf-list entry.
- o error-tag 'invalid-value' is returned by the CoMI server when the CoMI client tries to update or create a leaf with a value encoded using an invalid CBOR datatype or if the 'range', 'length', 'pattern' or 'require-instance' constrain is not fulfilled.
 - * error-app-tag 'invalid-datatype' is returned by the CoMI server when CBOR encoding does not follow the rules set by or when the value is incompatible with the YANG Built-In type (e.g. a value greater than 127 for an int8, undefined enumeration)
 - * error-app-tag 'not-in-range' is returned by the CoMI server when the validation of the 'range' property fails.
 - * error-app-tag 'invalid-length' is returned by the CoMI server when the validation of the 'length' property fails.

- * error-app-tag 'pattern-test-failed' is returned by the CoMI server when the validation of the 'pattern' property fails.
- o error-tag 'missing-element' is returned by the CoMI server when the operation requested by a CoMI client fails to comply with the 'mandatory' constraint defined. The 'mandatory' constraint is enforced for leafs and choices, unless the node or any of its ancestors have a 'when' condition or 'if-feature' expression that evaluates to 'false'.
- * error-app-tag 'missing-key' is returned by the CoMI server to further qualify a missing-element error. This error is returned when the CoMI client tries to create or list instance, without all the 'key' specified or when the CoMI client tries to delete a leaf listed as a 'key'.
- * error-app-tag 'missing-input-parameter' is returned by the CoMI server when the input parameters of an RPC or action are incomplete.
- o error-tag 'unknown-element' is returned by the CoMI server when the CoMI client tries to access a data node of a YANG module not supported, of a data node associated to an 'if-feature' expression evaluated to 'false' or to a 'when' condition evaluated to 'false'.
- o error-tag 'bad-element' is returned by the CoMI server when the CoMI client tries to create data nodes for more than one case in a choice.
- o error-tag 'data-missing' is returned by the CoMI server when a data node required to accept the request is not present.
- * error-app-tag 'instance-required' is returned by the CoMI server when a leaf of type 'instance-identifier' or 'leafref' marked with require-instance set to 'true' refers to an instance that does not exist.
- * error-app-tag 'missing-choice' is returned by the CoMI server when no nodes exist in a mandatory choice.
- o error-tag 'error' is returned by the CoMI server when an unspecified error has occurred.

For example, the CoMI server might return the following error.

```
RES: 4.00 Bad Request (Content-Format: application/yang-data+cbor)
{
  1024 : {
    +4 : 1011,          / error-tag (SID 1028) /
                        /   = invalid-value (SID 1011) /
    +1 : 1018,          / error-app-tag (SID 1025) /
                        /   = not-in-range (SID 1018) /
    +2 : 1740,          / error-data-node (SID 1026) /
                        /   = timezone-utc-offset (SID 1740) /
    +3 : "maximum value exceeded" / error-message (SID 1027) /
  }
}
```

8. Security Considerations

For secure network management, it is important to restrict access to configuration variables only to authorized parties. CoMI re-uses the security mechanisms already available to CoAP, this includes DTLS [RFC6347] for protected access to resources, as well suitable authentication and authorization mechanisms.

Among the security decisions that need to be made are selecting security modes and encryption mechanisms (see [RFC7252]). This requires a trade-off, as the NoKey mode gives no protection at all, but is easy to implement, whereas the Certificate mode is quite secure, but may be too complex for constrained devices.

In addition, mechanisms for authentication and authorization may need to be selected in case NoKey is used.

9. IANA Considerations

9.1. Resource Type (rt=) Link Target Attribute Values Registry

This document adds the following resource type to the "Resource Type (rt=) Link Target Attribute Values", within the "Constrained RESTful Environments (CoRE) Parameters" registry.

| Value | Description | Reference |
|-----------|---------------------|-----------|
| core.c.ds | YANG datastore | RFC XXXX |
| core.c.dn | YANG data node | RFC XXXX |
| core.c.yl | YANG module library | RFC XXXX |
| core.c.es | YANG event stream | RFC XXXX |

// RFC Ed.: replace RFC XXXX with this RFC number and remove this note.

9.2. CoAP Content-Formats Registry

This document adds the following Content-Format to the "CoAP Content-Formats", within the "Constrained RESTful Environments (CoRE) Parameters" registry.

| Media Type | Content Coding | ID | Reference |
|-----------------------------------|----------------|------|-----------|
| application/yang-data+cbor | | TBD1 | RFC XXXX |
| application/yang-identifiers+cbor | | TBD2 | RFC XXXX |
| application/yang-instances+cbor | | TBD3 | RFC XXXX |

// RFC Ed.: replace TBD1, TBD2 and TBD3 with assigned IDs and remove this note. // RFC Ed.: replace RFC XXXX with this RFC number and remove this note.

9.3. Media Types Registry

This document adds the following media types to the "Media Types" registry.

| Name | Template | Reference |
|-----------------------|---------------------------------------|-----------|
| yang-data+cbor | application/yang-data+cbor | RFC XXXX |
| yang-identifiers+cbor | application/ yang-identifiers+cbor | RFC XXXX |
| yang-instances+cbor | application/ yang-instances+cbor | RFC XXXX |

Each of these media types share the following information:

- o Subtype name: <as listed in table>
- o Required parameters: N/A
- o Optional parameters: N/A
- o Encoding considerations: binary
- o Security considerations: See the Security Considerations section of RFC XXXX
- o Interoperability considerations: N/A
- o Published specification: RFC XXXX
- o Applications that use this media type: CoMI
- o Fragment identifier considerations: N/A
- o Additional information:
 - * Deprecated alias names for this type: N/A
 - * Magic number(s): N/A
 - * File extension(s): N/A
 - * Macintosh file type code(s): N/A
- o Person & email address to contact for further information:
iesg&ietf.org

- o Intended usage: COMMON
- o Restrictions on usage: N/A
- o Author: Michel Veillette, ietf@augustcellars.com
- o Change Controller: IESG
- o Provisional registration? No

// RFC Ed.: replace RFC XXXX with this RFC number and remove this note.

10. Acknowledgements

We are very grateful to Bert Greevenbosch who was one of the original authors of the CoMI specification.

Mehmet Ersue and Bert Wijnen explained the encoding aspects of PDUs transported under SNMP. Carsten Bormann has given feedback on the use of CBOR.

The draft has benefited from comments (alphabetical order) by Rodney Cummings, Dee Denteneer, Esko Dijk, Michael van Hartskamp, Tanguy Ropitault, Juergen Schoenwaelder, Anuj Sehgal, Zach Shelby, Hannes Tschofenig, Michael Verschoor, and Thomas Watteyne.

11. References

11.1. Normative References

- [I-D.ietf-core-sid]
Veillette, M., Pelov, A., and I. Petrov, "YANG Schema Item Identifier (SID)", draft-ietf-core-sid-07 (work in progress), July 2019.
- [I-D.ietf-core-yang-cbor]
Veillette, M., Petrov, I., and A. Pelov, "CBOR Encoding of Data Modeled with YANG", draft-ietf-core-yang-cbor-10 (work in progress), April 2019.
- [I-D.veillette-core-yang-library]
Veillette, M. and I. Petrov, "Constrained YANG Module Library", draft-veillette-core-yang-library-04 (work in progress), March 2019.

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <<https://www.rfc-editor.org/info/rfc4648>>.
- [RFC5277] Chisholm, S. and H. Trevino, "NETCONF Event Notifications", RFC 5277, DOI 10.17487/RFC5277, July 2008, <<https://www.rfc-editor.org/info/rfc5277>>.
- [RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <<https://www.rfc-editor.org/info/rfc6241>>.
- [RFC6243] Bierman, A. and B. Lengyel, "With-defaults Capability for NETCONF", RFC 6243, DOI 10.17487/RFC6243, June 2011, <<https://www.rfc-editor.org/info/rfc6243>>.
- [RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <<https://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, <<https://www.rfc-editor.org/info/rfc7252>>.
- [RFC7641] Hartke, K., "Observing Resources in the Constrained Application Protocol (CoAP)", RFC 7641, DOI 10.17487/RFC7641, September 2015, <<https://www.rfc-editor.org/info/rfc7641>>.
- [RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <<https://www.rfc-editor.org/info/rfc7950>>.
- [RFC7959] Bormann, C. and Z. Shelby, Ed., "Block-Wise Transfers in the Constrained Application Protocol (CoAP)", RFC 7959, DOI 10.17487/RFC7959, August 2016, <<https://www.rfc-editor.org/info/rfc7959>>.
- [RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <<https://www.rfc-editor.org/info/rfc8040>>.

- [RFC8132] van der Stok, P., Bormann, C., and A. Sehgal, "PATCH and FETCH Methods for the Constrained Application Protocol (CoAP)", RFC 8132, DOI 10.17487/RFC8132, April 2017, <<https://www.rfc-editor.org/info/rfc8132>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.

11.2. Informative References

- [RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <<https://www.rfc-editor.org/info/rfc6347>>.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <<https://www.rfc-editor.org/info/rfc6690>>.
- [RFC7223] Bjorklund, M., "A YANG Data Model for Interface Management", RFC 7223, DOI 10.17487/RFC7223, May 2014, <<https://www.rfc-editor.org/info/rfc7223>>.
- [RFC7317] Bierman, A. and M. Bjorklund, "A YANG Data Model for System Management", RFC 7317, DOI 10.17487/RFC7317, August 2014, <<https://www.rfc-editor.org/info/rfc7317>>.
- [RFC8342] Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K., and R. Wilton, "Network Management Datastore Architecture (NMDA)", RFC 8342, DOI 10.17487/RFC8342, March 2018, <<https://www.rfc-editor.org/info/rfc8342>>.

Appendix A. ietf-comi YANG module

```
<CODE BEGINS> file "ietf-comi@2019-03-28.yang"
module ietf-comi {
  yang-version 1.1;

  namespace "urn:ietf:params:xml:ns:yang:ietf-comi";
  prefix comi;

  import ietf-datastores {
    prefix ds;
  }

  import ietf-restconf {
    prefix rc;
    description
```

```
    "This import statement is required to access
      the yang-data extension defined in RFC 8040.";
  reference "RFC 8040: RESTCONF Protocol";
}

organization
  "IETF Core Working Group";

contact
  "Michel Veillette
    <mailto:michel.veillette@trilliantinc.com>

    Alexander Pelov
    <mailto:alexander@ackl.io>

    Peter van der Stok
    <mailto:consultancy@vanderstok.org>

    Andy Bierman
    <mailto:andy@yumaworks.com>";

description
  "This module contains the different definitions required
    by the CoMI protocol.";

revision 2019-03-28 {
  description
    "Initial revision.";
  reference
    "[I-D.ietf-core-comi] CoAP Management Interface";
}

identity unified {
  base ds:datastore;
  description
    "Identifier of the unified configuration and operational
      state datastore.";
}

identity error-tag {
  description
    "Base identity for error-tag.";
}

identity operation-failed {
  base error-tag;
  description
    "Returned by the CoMI server when the operation request
```

```
        can't be processed successfully.";
    }

    identity invalid-value {
        base error-tag;
        description
            "Returned by the CoMI server when the CoMI client tries to
            update or create a leaf with a value encoded using an
            invalid CBOR datatype or if the 'range', 'length',
            'pattern' or 'require-instance' constrain is not
            fulfilled.";
    }

    identity missing-element {
        base error-tag;
        description
            "Returned by the CoMI server when the operation requested
            by a CoMI client fails to comply with the 'mandatory'
            constraint defined. The 'mandatory' constraint is
            enforced for leafs and choices, unless the node or any of
            its ancestors have a 'when' condition or 'if-feature'
            expression that evaluates to 'false'.";
    }

    identity unknown-element {
        base error-tag;
        description
            "Returned by the CoMI server when the CoMI client tries to
            access a data node of a YANG module not supported, of a
            data node associated with an 'if-feature' expression
            evaluated to 'false' or to a 'when' condition evaluated
            to 'false'.";
    }

    identity bad-element {
        base error-tag;
        description
            "Returned by the CoMI server when the CoMI client tries to
            create data nodes for more than one case in a choice.";
    }

    identity data-missing {
        base error-tag;
        description
            "Returned by the CoMI server when a data node required to
            accept the request is not present.";
    }
}
```

```
identity error {
  base error-tag;
  description
    "Returned by the CoMI server when an unspecified error has
    occurred.";
}

identity error-app-tag {
  description
    "Base identity for error-app-tag.";
}

identity malformed-message {
  base error-app-tag;
  description
    "Returned by the CoMI server when the payload received
    from the CoMI client don't contain a well-formed CBOR
    content as defined in [RFC7049] section 3.3 or don't
    comply with the CBOR structure defined within this
    document.";
}

identity data-not-unique {
  base error-app-tag;
  description
    "Returned by the CoMI server when the validation of the
    'unique' constraint of a list or leaf-list fails.";
}

identity too-many-elements {
  base error-app-tag;
  description
    "Returned by the CoMI server when the validation of the
    'max-elements' constraint of a list or leaf-list fails.";
}

identity too-few-elements {
  base error-app-tag;
  description
    "Returned by the CoMI server when the validation of the
    'min-elements' constraint of a list or leaf-list fails.";
}

identity must-violation {
  base error-app-tag;
  description
    "Returned by the CoMI server when the restrictions
    imposed by a 'must' statement are violated.";
```

```
}

identity duplicate {
  base error-app-tag;
  description
    "Returned by the CoMI server when a client tries to create
    a duplicate list or leaf-list entry.";
}

identity invalid-datatype {
  base error-app-tag;
  description
    "Returned by the CoMI server when CBOR encoding is
    incorrect or when the value encoded is incompatible with
    the YANG Built-In type. (e.g. value greater than 127
    for an int8, undefined enumeration).";
}

identity not-in-range {
  base error-app-tag;
  description
    "Returned by the CoMI server when the validation of the
    'range' property fails.";
}

identity invalid-length {
  base error-app-tag;
  description
    "Returned by the CoMI server when the validation of the
    'length' property fails.";
}

identity pattern-test-failed {
  base error-app-tag;
  description
    "Returned by the CoMI server when the validation of the
    'pattern' property fails.";
}

identity missing-key {
  base error-app-tag;
  description
    "Returned by the CoMI server to further qualify a
    missing-element error. This error is returned when the
    CoMI client tries to create or list instance, without all
    the 'key' specified or when the CoMI client tries to
    delete a leaf listed as a 'key'.";
}
```

```
identity missing-input-parameter {
  base error-app-tag;
  description
    "Returned by the CoMI server when the input parameters
    of a RPC or action are incomplete.";
}

identity instance-required {
  base error-app-tag;
  description
    "Returned by the CoMI server when a leaf of type
    'instance-identifier' or 'leafref' marked with
    require-instance set to 'true' refers to an instance
    that does not exist.";
}

identity missing-choice {
  base error-app-tag;
  description
    "Returned by the CoMI server when no nodes exist in a
    mandatory choice.";
}

rc:yang-data comi-error {
  container error {
    description
      "Optional payload of a 4.00 Bad Request CoAP error.";

    leaf error-tag {
      type identityref {
        base error-tag;
      }
      mandatory true;
      description
        "The enumerated error-tag.";
    }

    leaf error-app-tag {
      type identityref {
        base error-app-tag;
      }
      description
        "The application-specific error-tag.";
    }

    leaf error-data-node {
      type instance-identifier;
      description
```

```
        "When the error reported is caused by a specific data node,
        this leaf identifies the data node in error.";
    }

    leaf error-message {
        type string;
        description
            "A message describing the error.";
    }
}
}
}
<CODE ENDS>
```

Appendix B. ietf-comi .sid file

```
{
  "assignment-ranges": [
    {
      "entry-point": 1000,
      "size": 100
    }
  ],
  "module-name": "ietf-comi",
  "module-revision": "2019-03-28",
  "items": [
    {
      "namespace": "module",
      "identifier": "ietf-comi",
      "sid": 1000
    },
    {
      "namespace": "identity",
      "identifier": "bad-element",
      "sid": 1001
    },
    {
      "namespace": "identity",
      "identifier": "data-missing",
      "sid": 1002
    },
    {
      "namespace": "identity",
      "identifier": "data-not-unique",
      "sid": 1003
    },
    {
      "namespace": "identity",
```

```
    "identifier": "duplicate",
    "sid": 1004
  },
  {
    "namespace": "identity",
    "identifier": "error",
    "sid": 1005
  },
  {
    "namespace": "identity",
    "identifier": "error-app-tag",
    "sid": 1006
  },
  {
    "namespace": "identity",
    "identifier": "error-tag",
    "sid": 1007
  },
  {
    "namespace": "identity",
    "identifier": "instance-required",
    "sid": 1008
  },
  {
    "namespace": "identity",
    "identifier": "invalid-datatype",
    "sid": 1009
  },
  {
    "namespace": "identity",
    "identifier": "invalid-length",
    "sid": 1010
  },
  {
    "namespace": "identity",
    "identifier": "invalid-value",
    "sid": 1011
  },
  {
    "namespace": "identity",
    "identifier": "malformed-message",
    "sid": 1012
  },
  {
    "namespace": "identity",
    "identifier": "missing-choice",
    "sid": 1013
  },
}
```



```
{
  "namespace": "identity",
  "identifier": "missing-element",
  "sid": 1014
},
{
  "namespace": "identity",
  "identifier": "missing-input-parameter",
  "sid": 1015
},
{
  "namespace": "identity",
  "identifier": "missing-key",
  "sid": 1016
},
{
  "namespace": "identity",
  "identifier": "must-violation",
  "sid": 1017
},
{
  "namespace": "identity",
  "identifier": "not-in-range",
  "sid": 1018
},
{
  "namespace": "identity",
  "identifier": "operation-failed",
  "sid": 1019
},
{
  "namespace": "identity",
  "identifier": "pattern-test-failed",
  "sid": 1020
},
{
  "namespace": "identity",
  "identifier": "too-few-elements",
  "sid": 1021
},
{
  "namespace": "identity",
  "identifier": "too-many-elements",
  "sid": 1022
},
{
  "namespace": "identity",
  "identifier": "unified",
```

```
    "sid": 1029
  },
  {
    "namespace": "identity",
    "identifier": "unknown-element",
    "sid": 1023
  },
  {
    "namespace": "data",
    "identifier": "/ietf-comi:error",
    "sid": 1024
  },
  {
    "namespace": "data",
    "identifier": "/ietf-comi:error/error-app-tag",
    "sid": 1025
  },
  {
    "namespace": "data",
    "identifier": "/ietf-comi:error/error-data-node",
    "sid": 1026
  },
  {
    "namespace": "data",
    "identifier": "/ietf-comi:error/error-message",
    "sid": 1027
  },
  {
    "namespace": "data",
    "identifier": "/ietf-comi:error/error-tag",
    "sid": 1028
  }
]
}
```

Authors' Addresses

Michel Veillette (editor)
Trilliant Networks Inc.
610 Rue du Luxembourg
Granby, Quebec J2J 2V2
Canada

Email: michel.veillette@trilliant.com

Peter van der Stok (editor)
consultant

Phone: +31-492474673 (Netherlands), +33-966015248 (France)
Email: consultancy@vanderstok.org
URI: www.vanderstok.org

Alexander Pelov
Acklio
2bis rue de la Chataigneraie
Cesson-Sevigne, Bretagne 35510
France

Email: a@ackl.io

Andy Bierman
YumaWorks
685 Cochran St.
Suite #160
Simi Valley, CA 93065
USA

Email: andy@yumaworks.com

Ivaylo Petrov (editor)
Acklio
1137A avenue des Champs Blancs
Cesson-Sevigne, Bretagne 35510
France

Email: ivaylo@ackl.io

CoRE Working Group
Internet-Draft
Intended status: Informational
Expires: January 9, 2020

Z. Shelby
ARM
M. Koster
SmartThings
C. Groves

J. Zhu
Huawei
B. Silverajan, Ed.
Tampere University
July 08, 2019

Dynamic Resource Linking for Constrained RESTful Environments
draft-ietf-core-dynlink-09

Abstract

This specification defines Link Bindings, which provide dynamic linking of state updates between resources, either on an endpoint or between endpoints, for systems using CoAP (RFC7252). This specification also defines Conditional Notification Attributes that work with Link Bindings or with CoAP Observe (RFC7641).

Editor note

The git repository for the draft is found at <https://github.com/core-wg/dynlink>

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on January 9, 2020.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

| | |
|---|----|
| 1. Introduction | 3 |
| 2. Terminology | 3 |
| 3. Conditional Notification Attributes | 4 |
| 3.1. Attribute Definitions | 4 |
| 3.1.1. Minimum Period (pmin) | 5 |
| 3.1.2. Maximum Period (pmax) | 6 |
| 3.1.3. Change Step (st) | 6 |
| 3.1.4. Greater Than (gt) | 6 |
| 3.1.5. Less Than (lt) | 7 |
| 3.1.6. Notification Band (band) | 7 |
| 3.2. Server processing of Conditional Notification Attributes | 8 |
| 4. Link Bindings | 9 |
| 4.1. The "bind" attribute and Binding Methods | 10 |
| 4.1.1. Polling | 11 |
| 4.1.2. Observe | 11 |
| 4.1.3. Push | 12 |
| 4.2. Link Relation | 12 |
| 5. Binding Table | 12 |
| 6. Implementation Considerations | 13 |
| 7. Security Considerations | 14 |
| 8. IANA Considerations | 14 |
| 8.1. Resource Type value 'core.bnd' | 14 |
| 8.2. Link Relation Type | 14 |
| 9. Acknowledgements | 15 |
| 10. Contributors | 15 |
| 11. Changelog | 15 |
| 12. References | 17 |
| 12.1. Normative References | 17 |
| 12.2. Informative References | 18 |
| Appendix A. Examples | 18 |
| A.1. Minimum Period (pmin) example | 18 |

| | |
|--|----|
| A.2. Maximum Period (pmax) example | 19 |
| A.3. Greater Than (gt) example | 20 |
| A.4. Greater Than (gt) and Period Max (pmax) example | 21 |
| Authors' Addresses | 22 |

1. Introduction

IETF Standards for machine to machine communication in constrained environments describe a REST protocol [RFC7252] and a set of related information standards that may be used to represent machine data and machine metadata in REST interfaces. CoRE Link-format [RFC6690] is a standard for doing Web Linking [RFC8288] in constrained environments.

This specification introduces the concept of a Link Binding, which defines a new link relation type to create a dynamic link between resources over which state updates are conveyed. Specifically, a Link Binding is a unidirectional link for binding the states of source and destination resources together such that updates to one are sent over the link to the other. CoRE Link Format representations are used to configure, inspect, and maintain Link Bindings. This specification additionally defines Conditional Notification Attributes for use with Link Bindings and with CoRE Observe [RFC7641].

2. 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 BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

This specification requires readers to be familiar with all the terms and concepts that are discussed in [RFC8288] and [RFC6690]. This specification makes use of the following additional terminology:

Link Binding: A unidirectional logical link between a source resource and a destination resource, over which state information is synchronized.

State Synchronization: Depending on the binding method (Polling, Observe, Push) different REST methods may be used to synchronize the resource values between a source and a destination. The process of using a REST method to achieve this is defined as "State Synchronization". The endpoint triggering the state synchronization is the synchronization initiator.

Notification Band: A resource value range that results in state synchronization. The value range may be bounded by a minimum and maximum value or may be unbounded having either a minimum or maximum value.

3. Conditional Notification Attributes

3.1. Attribute Definitions

This specification defines Conditional Notification Attributes, which provide for fine-grained control of notification and state synchronization when using CoRE Observe [RFC7641] or Link Bindings (see Section 4). Conditional Notification Attributes define the conditions that trigger a notification.

When resource interfaces following this specification are made available over CoAP, the CoAP Observation mechanism [RFC7641] MAY also be used to observe any changes in a resource, and receive asynchronous notifications as a result. A resource marked as Observable in its link description SHOULD support these Conditional Notification Attributes.

The set of parameters defined here allow a client to control how often a client is interested in receiving notifications and how much a resource value should change for the new representation to be interesting.

One or more Notification Attributes MAY be included as query parameters in an Observe request.

These attributes are defined below:

| Attribute | Parameter | Value |
|--------------------|-----------|-----------------|
| Minimum Period (s) | pmin | xs:decimal (>0) |
| Maximum Period (s) | pmax | xs:decimal (>0) |
| Change Step | st | xs:decimal (>0) |
| Greater Than | gt | xs:decimal |
| Less Than | lt | xs:decimal |
| Notification Band | band | xs:boolean |

Table 1: Conditional Notification Attributes

Conditional Notification Attributes SHOULD be evaluated on all potential notifications from a resource, whether resulting from an internal server-driven sampling process or from external update requests to the server.

Note: In this draft, we assume that there are finite quantization effects in the internal or external updates to the value of a resource; specifically, that a resource may be updated at any time with any valid value. We therefore avoid any continuous-time assumptions in the description of the Conditional Notification Attributes and instead use the phrase "sampled value" to refer to a member of a sequence of values that may be internally observed from the resource state over time.

3.1.1. Minimum Period (pmin)

When present, the minimum period indicates the minimum time, in seconds, between two consecutive notifications (whether or not the resource value has changed). In the absence of this parameter, the minimum period is up to the server. The minimum period MUST be greater than zero otherwise the receiver MUST return a CoAP error code 4.00 "Bad Request" (or equivalent).

A server MAY report the last sampled value that occurred during the pmin interval, after the pmin interval expires.

Note: Due to finite quantization effects, the time between notifications may be greater than pmin even when the sampled value changes within the pmin interval. Pmin may or may not be used to drive the internal sampling process.

3.1.2. Maximum Period (pmax)

When present, the maximum period indicates the maximum time, in seconds, between two consecutive notifications (whether or not the resource value has changed). In the absence of this parameter, the maximum period is up to the server. The maximum period MUST be greater than zero and MUST be greater than the minimum period parameter (if present) otherwise the receiver MUST return a CoAP error code 4.00 "Bad Request" (or equivalent).

3.1.3. Change Step (st)

When present, the change step indicates how much the value of a resource SHOULD change before triggering a notification, compared to the value of the previous notification. Upon reception of a query including the st attribute, the most recently sampled value of the resource is reported, and then set as the last reported value (last_rep_v). When a subsequent sample or update of the resource value differs from the last reported value by an amount, positive or negative, greater than or equal to st, and the time for pmin has elapsed since the last notification, a notification is sent and the last reported value is updated to the value sent in the notification. The change step MUST be greater than zero otherwise the receiver MUST return a CoAP error code 4.00 "Bad Request" (or equivalent).

The Change Step parameter can only be supported on resources with a scalar numeric value.

Note: Due to sampling and other constraints, e.g. pmin, the resource value received in two sequential notifications may differ by more than st.

3.1.4. Greater Than (gt)

When present, Greater Than indicates the upper limit value the sampled value SHOULD cross before triggering a notification. A notification is sent whenever the sampled value crosses the specified upper limit value, relative to the last reported value, and the time for pmin has elapsed since the last notification. The sampled value is sent in the notification. If the value continues to rise, no notifications are generated as a result of gt. If the value drops below the upper limit value then a notification is sent, subject again to the pmin time.

The Greater Than parameter can only be supported on resources with a scalar numeric value.

3.1.5. Less Than (lt)

When present, Less Than indicates the lower limit value the resource value SHOULD cross before triggering a notification. A notification is sent when the samples value crosses the specified lower limit value, relative to the last reported value, and the time for pmin has elapsed since the last notification. The sampled value is sent in the notification. If the value continues to fall no notifications are generated as a result of lt. If the value rises above the lower limit value then a new notification is sent, subject to the pmin time..

The Less Than parameter can only be supported on resources with a scalar numeric value.

3.1.6. Notification Band (band)

The notification band attribute allows a bounded or unbounded (based on a minimum or maximum) value range that may trigger multiple notifications. This enables use cases where different ranges results in differing behaviour. For example: monitoring the temperature of machinery. Whilst the temperature is in the normal operating range only periodic observations are needed. However as the temperature moves to more abnormal ranges more frequent synchronization/reporting may be needed.

Without a notification band, a transition across a less than (lt), or greater than (gt) limit only generates one notification. This means that it is not possible to describe a case where multiple notifications are sent so long as the limit is exceeded.

The band attribute works as a modifier to the behaviour of gt and lt. Therefore, if band is present in a query, gt, lt or both, MUST be included.

When band is present with the lt attribute, it defines the lower bound for the notification band (notification band minimum). Notifications occur when the resource value is equal to or above the notification band minimum. If lt is not present there is no minimum value for the band.

When band is present with the gt attribute, it defines the upper bound for the notification band (notification band maximum). Notifications occur when the resource value is equal to or below the notification band maximum. If gt is not present there is no maximum value for the band.

If band is present with both the gt and lt attributes, notification occurs when the resource value is greater than or equal to gt or when the resource value is less than or equal to lt.

If a band is specified in which the value of gt is less than that of lt, in-band notification occurs. That is, notification occurs whenever the resource value is between the gt and lt values, including equal to gt or lt.

If the band is specified in which the value of gt is greater than that of lt, out-of-band notification occurs. That is, notification occurs when the resource value not between the gt and lt values, excluding equal to gt and lt.

The Notification Band parameter can only be supported on resources with a scalar numeric value.

3.2. Server processing of Conditional Notification Attributes

Pmin, pmax, st, gt, lt and band may be present in the same query. However, they are not defined at multiple prioritization levels. The server sends a notification whenever any of the parameter conditions are met, upon which it updates its last notification value and time to prepare for the next notification. Only one notification occurs when there are multiple conditions being met at the same time. The reference code below illustrates the logic to determine when a notification is to be sent.

```

bool notifiable( Resource * r ) {

#define BAND r->band
#define SCALAR_TYPE ( num_type == r->type )
#define STRING_TYPE ( str_type == r->type )
#define BOOLEAN_TYPE ( bool_type == r->type )
#define PMIN_EX ( r->last_sample_time - r->last_rep_time >= r->pmin )
#define PMAX_EX ( r->last_sample_time - r->last_rep_time > r->pmax )
#define LT_EX ( r->v < r->lt ^ r->last_rep_v < r->lt )
#define GT_EX ( r->v > r->gt ^ r->last_rep_v > r->gt )
#define ST_EX ( abs( r->v - r->last_rep_v ) >= r->st )
#define IN_BAND ( ( r->gt <= r->v && r->v <= r->lt ) || ( r->lt <= r->gt && r->g
t <= r->v ) || ( r->v <= r->lt && r->lt <= r->gt ) )
#define VB_CHANGE ( r->vb != r->last_rep_vb )
#define VS_CHANGE ( r->vs != r->last_rep_vs )

    return (
        PMIN_EX &&
        ( SCALAR_TYPE ?
            ( ( !BAND && ( GT_EX || LT_EX || ST_EX || PMAX_EX ) ) ||
              ( BAND && IN_BAND && ( ST_EX || PMAX_EX ) ) )
        : STRING_TYPE ?
            ( VS_CHANGE || PMAX_EX )
        : BOOLEAN_TYPE ?
            ( VB_CHANGE || PMAX_EX )
        : false )
    );
}

```

Figure 1: Code logic for conditional notification attribute interactions

4. Link Bindings

In a M2M RESTful environment, endpoints may directly exchange the content of their resources to operate the distributed system. For example, a light switch may supply on-off control information that may be sent directly to a light resource for on-off control. Beforehand, a configuration phase is necessary to determine how the resources of the different endpoints are related to each other. This can be done either automatically using discovery mechanisms or by means of human intervention and a so-called commissioning tool.

In this specification such an abstract relationship between two resources is defined, called a Link Binding. The configuration phase necessitates the exchange of binding information, so a format recognized by all CoRE endpoints is essential. This specification defines a format based on the CoRE Link-Format to represent binding

information along with the rules to define a binding method which is a specialized relationship between two resources.

The purpose of such a binding is to synchronize content updates between a source resource and a destination resource. The destination resource MAY be a group resource if the authority component of the destination URI contains a group address (either a multicast address or a name that resolves to a multicast address). Since a binding is unidirectional, the binding entry defining a relationship is present only on one endpoint. The binding entry may be located either on the source or the destination endpoint depending on the binding method.

Conditional Notification Attributes defined in Section 3 can be used with Link Bindings in order to customize the notification behavior and timing.

4.1. The "bind" attribute and Binding Methods

A binding method defines the rules to generate the network-transfer exchanges that synchronize state between source and destination resources. By using REST methods content is sent from the source resource to the destination resource.

This specification defines a new CoRE link attribute "bind". This is the identifier for a binding method which defines the rules to synchronize the destination resource. This attribute is mandatory.

| Attribute | Parameter | Value |
|----------------|-----------|-----------|
| Binding method | bind | xs:string |

Table 2: The bind attribute

The following table gives a summary of the binding methods defined in this specification.

| Name | Identifier | Location | Method |
|---------|------------|-------------|---------------|
| Polling | poll | Destination | GET |
| Observe | obs | Destination | GET + Observe |
| Push | push | Source | PUT |

Table 3: Binding Method Summary

The description of a binding method defines the following aspects:

Identifier: This is the value of the "bind" attribute used to identify the method.

Location: This information indicates whether the binding entry is stored on the source or on the destination endpoint.

REST Method: This is the REST method used in the Request/Response exchanges.

Conditional Notification: How Conditional Notification Attributes are used in the binding.

The binding methods are described in more detail below.

4.1.1. Polling

The Polling method consists of sending periodic GET requests from the destination endpoint to the source resource and copying the content to the destination resource. The binding entry for this method **MUST** be stored on the destination endpoint. The destination endpoint **MUST** ensure that the polling frequency does not exceed the limits defined by the pmin and pmax attributes of the binding entry. The copying process **MAY** filter out content from the GET requests using value-based conditions (e.g based on the Change Step, Less Than, Greater Than attributes).

4.1.2. Observe

The Observe method creates an observation relationship between the destination endpoint and the source resource. On each notification the content from the source resource is copied to the destination resource. The creation of the observation relationship requires the CoAP Observation mechanism [RFC7641] hence this method is only permitted when the resources are made available over CoAP. The

binding entry for this method **MUST** be stored on the destination endpoint. The binding conditions are mapped as query parameters in the Observe request (see Section 3).

4.1.3. Push

When the Push method is assigned to a binding, the source endpoint sends PUT requests to the destination resource when the Conditional Notification Attributes are satisfied for the source resource. The source endpoint **SHOULD** only send a notification request if any included Conditional Notification Attributes are met. The binding entry for this method **MUST** be stored on the source endpoint.

4.2. Link Relation

Since Binding involves the creation of a link between two resources, Web Linking and the CoRE Link-Format used to represent binding information. This involves the creation of a new relation type, "boundto". In a Web link with this relation type, the target URI contains the location of the source resource and the context URI points to the destination resource.

5. Binding Table

The Binding Table is a special resource that describes the bindings on an endpoint. An endpoint offering a representation of the Binding Table resource **SHOULD** indicate its presence and enable its discovery by advertising a link at `"/.well-known/core"` [RFC6690]. If so, the Binding Table resource **MUST** be discoverable by using the Resource Type (rt) `'core.bnd'`.

The Methods column defines the REST methods supported by the Binding Table, which are described in more detail below.

| Resource | rt= | Methods | Content-Format |
|---------------|----------|----------|----------------|
| Binding Table | core.bnd | GET, PUT | link-format |

Table 4: Binding Table Description

The REST methods GET and PUT are used to manipulate a Binding Table. A GET request simply returns the current state of a Binding Table. A request with a PUT method and a content format of `application/link-format` is used to clear the bindings to the table or replaces its entire contents. All links in the payload of a PUT request **MUST** have a relation type "boundto".

The following example shows requests for discovering, retrieving and replacing bindings in a binding table.

```
Req: GET /.well-known/core?rt=core.bnd (application/link-format)
Res: 2.05 Content (application/link-format)
</bnd/>;rt=core.bnd;ct=40
```

```
Req: GET /bnd/
Res: 2.05 Content (application/link-format)
<coap://sensor.example.com/a/switch1/>;
    rel=boundto;anchor=/a/fan;;bind="obs",
<coap://sensor.example.com/a/switch2/>;
    rel=boundto;anchor=/a/light;bind="obs"
```

```
Req: PUT /bnd/ (Content-Format: application/link-format)
<coap://sensor.example.com/s/light>;
    rel="boundto";anchor="/a/light";bind="obs";pmin=10;pmax=60
Res: 2.04 Changed
```

```
Req: GET /bnd/
Res: 2.05 Content (application/link-format)
<coap://sensor.example.com/s/light>;
    rel="boundto";anchor="/a/light";bind="obs";pmin=10;pmax=60
```

Figure 2: Binding Table Example

Additional operations on the Binding Table can be specified in future documents. Such operations can include, for example, the usage of the iPATCH or PATCH methods [RFC8132] for fine-grained addition and removal of individual bindings or binding subsets.

6. Implementation Considerations

When using multiple resource bindings (e.g. multiple Observations of resource) with different bands, consideration should be given to the resolution of the resource value when setting sequential bands. For example: Given BandA (Abmn=10, Bbm=20) and BandB (Bbmn=21, Bbm=30). If the resource value returns an integer then notifications for values between and inclusive of 10 and 30 will be triggered. Whereas if the resolution is to one decimal point (0.1) then notifications for values 20.1 to 20.9 will not be triggered.

The use of the notification band minimum and maximum allow for a synchronization whenever a change in the resource value occurs. Theoretically this could occur in-line with the server internal sample period for the determining the resource value. Implementors SHOULD consider the resolution needed before updating the resource,

e.g. updating the resource when a temperature sensor value changes by 0.001 degree versus 1 degree.

The initiation of a Link Binding can be delegated from a client to a link state machine implementation, which can be an embedded client or a configuration tool. Implementation considerations have to be given to how to monitor transactions made by the configuration tool with regards to Link Bindings, as well as any errors that may arise with establishing Link Bindings in addition to established Link Bindings.

7. Security Considerations

Consideration has to be given to what kinds of security credentials the state machine of a configuration tool or an embedded client needs to be configured with, and what kinds of access control lists client implementations should possess, so that transactions on creating Link Bindings and handling error conditions can be processed by the state machine.

8. IANA Considerations

8.1. Resource Type value 'core.bnd'

This specification registers a new Resource Type Link Target Attribute 'core.bnd' in the Resource Type (rt=) registry established as per [RFC6690].

Attribute Value: core.bnd

Description: See Section 5. This attribute value is used to discover the resource representing a binding table, which describes the link bindings between source and destination resources for the purposes of synchronizing their content.

Reference: This specification. Note to RFC editor: please insert the RFC of this specification.

Notes: None

8.2. Link Relation Type

This specification registers the new "boundto" link relation type as per [RFC8288].

Relation Name: boundto

Description: The purpose of a boundto relation type is to indicate that there is a binding between a source resource and a

destination resource for the purposes of synchronizing their content.

Reference: This specification. Note to RFC editor: please insert the RFC of this specification.

Notes: None

Application Data: None

9. Acknowledgements

Acknowledgement is given to colleagues from the SENSEI project who were critical in the initial development of the well-known REST interface concept, to members of the IPSO Alliance where further requirements for interface types have been discussed, and to Szymon Sasin, Cedric Chauvenet, Daniel Gavelle and Carsten Bormann who have provided useful discussion and input to the concepts in this specification. Christian Amsuss supplied a comprehensive review of draft -06. Hannes Tschofenig and Mert Ocaak highlighted syntactical corrections in the usage of pmax and pmin in a query.

10. Contributors

Matthieu Vial
Schneider-Electric
Grenoble
France

Phone: +33 (0)47657 6522
EMail: matthieu.vial@schneider-electric.com

11. Changelog

draft-ietf-core-dynlink-09

- o Corrections in Table 1, Table 2, Figure 2.
- o Clarifications for additional operations to binding table added in section 5
- o Additional examples in Appendix A

draft-ietf-core-dynlink-08

- o Reorganize the draft to introduce Conditional Notification Attributes at the beginning

- o Made pmin and pmax type xs:decimal to accommodate fractional second timing
- o updated the attribute descriptions. lt and gt notify on all crossings, both directions
- o updated Binding Table description, removed interface description but introduced core.bnd rt attribute value

draft-ietf-core-dynlink-07

- o Added reference code to illustrate attribute interactions for observations

draft-ietf-core-dynlink-06

- o Document restructure and refactoring into three main sections
- o Clarifications on band usage
- o Implementation considerations introduced
- o Additional text on security considerations

draft-ietf-core-dynlink-05

- o Addition of a band modifier for gt and lt, adapted from draft-groves-core-obsattr
- o Removed statement prescribing gt MUST be greater than lt

draft-ietf-core-dynlink-03

- o General: Reverted to using "gt" and "lt" from "gth" and "lth" for this draft owing to concerns raised that the attributes are already used in LwM2M with the original names "gt" and "lt".
- o New author and editor added.

draft-ietf-core-dynlink-02

- o General: Changed the name of the greater than attribute "gt" to "gth" and the name of the less than attribute "lt" to "lth" due to conflict with the core resource directory draft lifetime "lt" attribute.
- o Clause 6.1: Addressed the editor's note by changing the link target attribute to "core.binding".

- o Added Appendix A for examples.

draft-ietf-core-dynlink-01

- o General: The term state synchronization has been introduced to describe the process of synchronization between destination and source resources.
- o General: The document has been restructured to make the information flow better.
- o Clause 3.1: The descriptions of the binding attributes have been updated to clarify their usage.
- o Clause 3.1: A new clause has been added to discuss the interactions between the resources.
- o Clause 3.4: Has been simplified to refer to the descriptions in 3.1. As the text was largely duplicated.
- o Clause 4.1: Added a clarification that individual resources may be removed from the binding table.
- o Clause 6: Formalised the IANA considerations.

draft-ietf-core-dynlink Initial Version 00:

- o This is a copy of draft-groves-core-dynlink-00

draft-groves-core-dynlink Draft Initial Version 00:

- o This initial version is based on the text regarding the dynamic linking functionality in I.D.ietf-core-interfaces-05.
- o The WADL description has been dropped in favour of a thorough textual description of the REST API.

12. References

12.1. Normative References

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

- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <<https://www.rfc-editor.org/info/rfc6690>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8288] Nottingham, M., "Web Linking", RFC 8288, DOI 10.17487/RFC8288, October 2017, <<https://www.rfc-editor.org/info/rfc8288>>.

12.2. Informative References

- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<https://www.rfc-editor.org/info/rfc7252>>.
- [RFC7641] Hartke, K., "Observing Resources in the Constrained Application Protocol (CoAP)", RFC 7641, DOI 10.17487/RFC7641, September 2015, <<https://www.rfc-editor.org/info/rfc7641>>.
- [RFC8132] van der Stok, P., Bormann, C., and A. Sehgal, "PATCH and FETCH Methods for the Constrained Application Protocol (CoAP)", RFC 8132, DOI 10.17487/RFC8132, April 2017, <<https://www.rfc-editor.org/info/rfc8132>>.

Appendix A. Examples

This appendix provides some examples of the use of binding attribute / observe attributes.

Note: For brevity the only the method or response code is shown in the header field.

A.1. Minimum Period (pmin) example

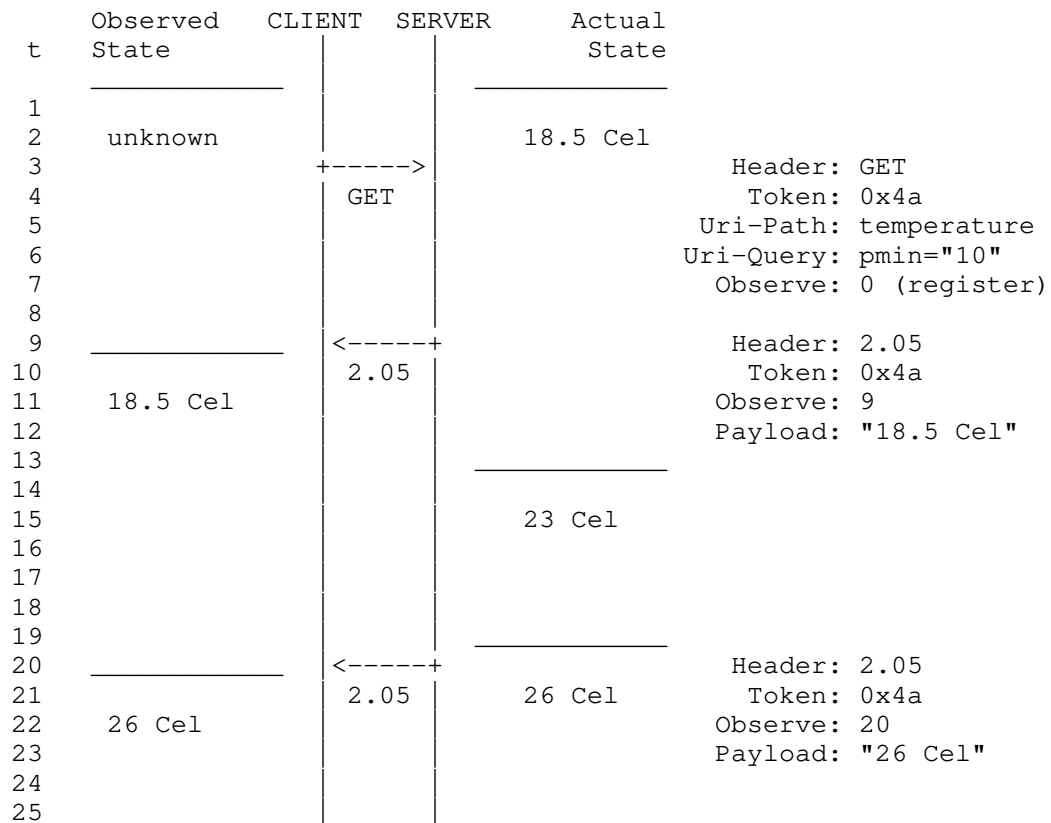
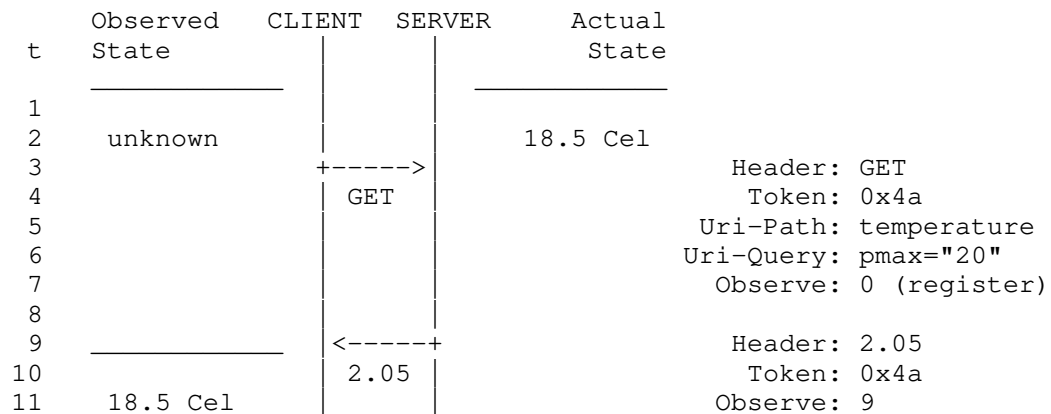


Figure 3: Client registers and receives one notification of the current state and one of a new state state when pmin time expires.

A.2. Maximum Period (pmax) example



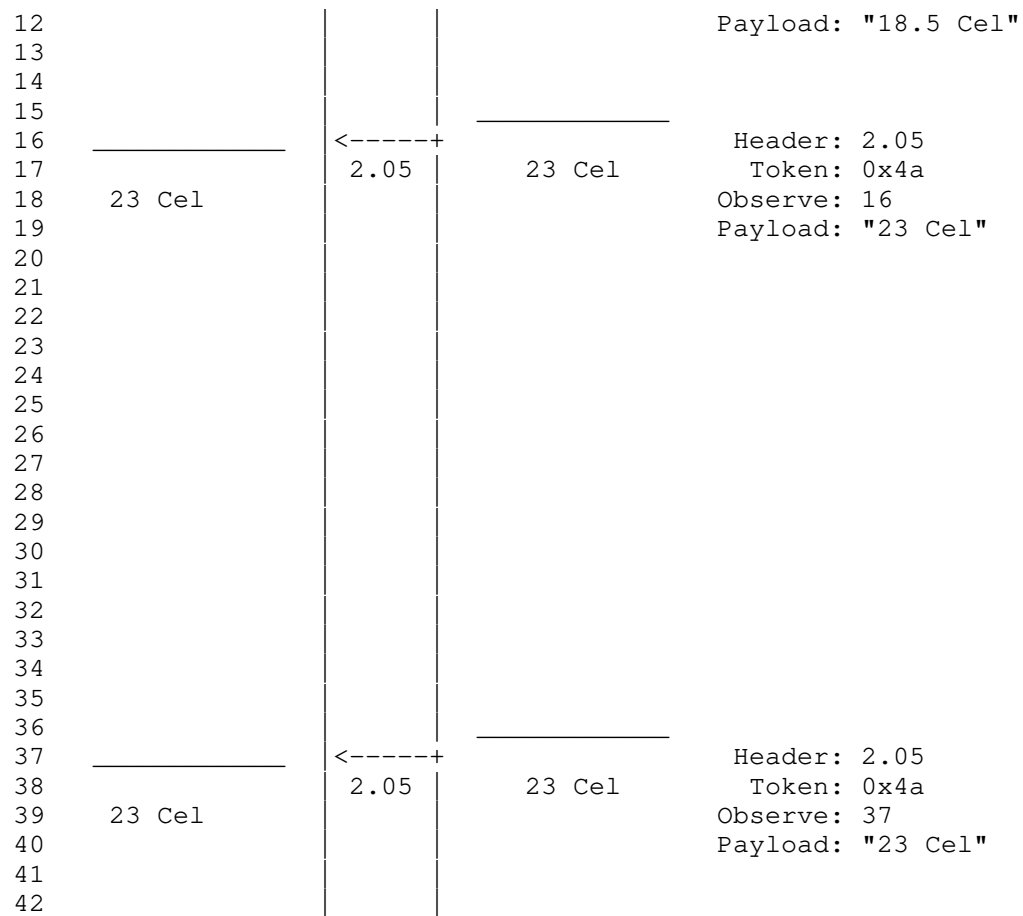


Figure 4: Client registers and receives one notification of the current state, one of a new state and one of an unchanged state when pmax time expires.

A.3. Greater Than (gt) example

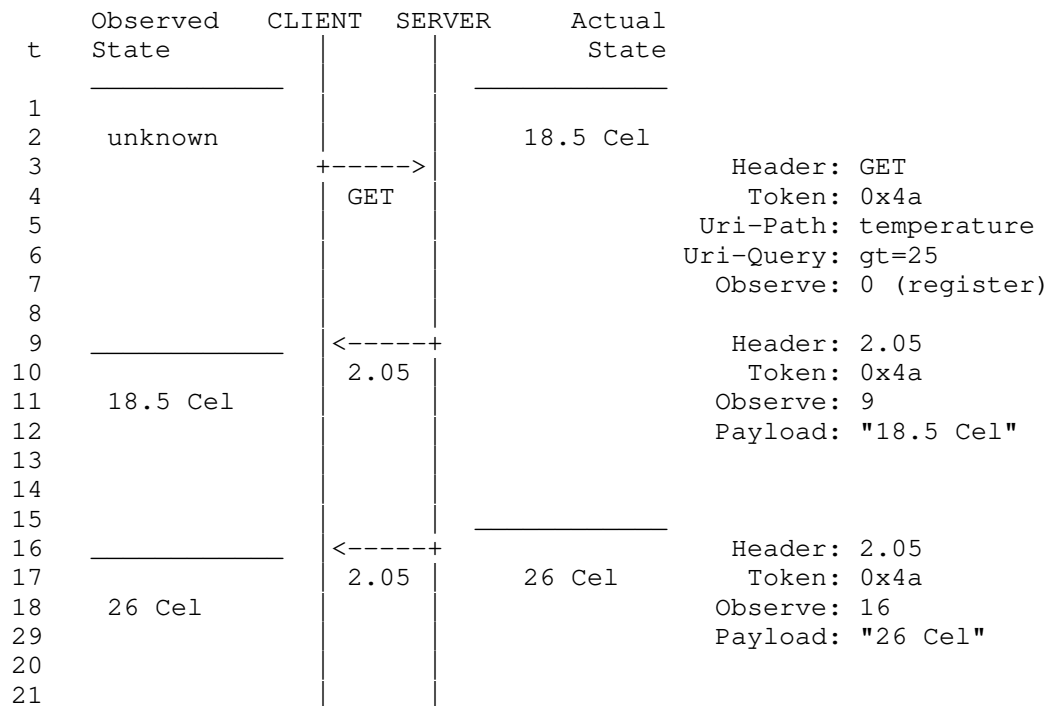
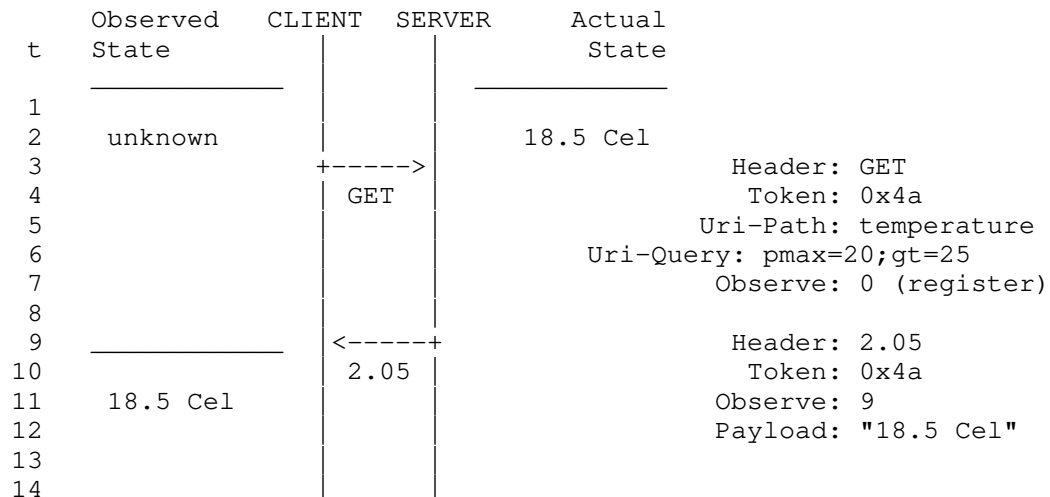


Figure 5: Client registers and receives one notification of the current state and one of a new state when it passes through the greater than threshold of 25.

A.4. Greater Than (gt) and Period Max (pmax) example



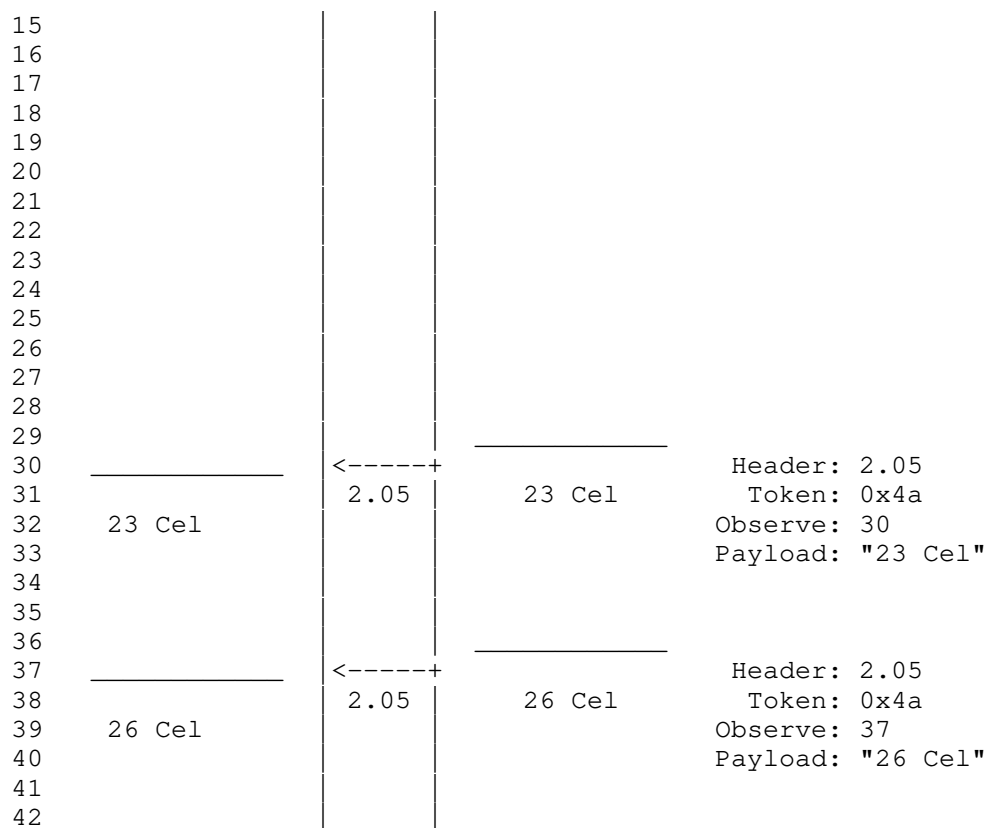


Figure 6: Client registers and receives one notification of the current state, one when pmax time expires and one of a new state when it passes through the greater than threshold of 25.

Authors' Addresses

Zach Shelby
ARM
Kidekuja 2
Vuokatti 88600
FINLAND

Phone: +358407796297
Email: zach.shelby@arm.com

Michael Koster
SmartThings
665 Clyde Avenue
Mountain View 94043
USA

Email: michael.koster@smarththings.com

Christian Groves
Australia

Email: cngroves.std@gmail.com

Jintao Zhu
Huawei
No.127 Jinye Road, Huawei Base, High-Tech Development District
Xi'an, Shaanxi Province
China

Email: jintao.zhu@huawei.com

Bilhanan Silverajan (editor)
Tampere University
Kalevantie 4
Tampere FI-33100
Finland

Email: bilhanan.silverajan@tuni.fi

CoRE Working Group
Internet-Draft
Updates: 7252 (if approved)
Intended status: Standards Track
Expires: November 7, 2019

C. Amsuess
J. Mattsson
G. Selander
Ericsson AB
May 06, 2019

CoAP: Echo, Request-Tag, and Token Processing
draft-ietf-core-echo-request-tag-05

Abstract

This document specifies enhancements to the Constrained Application Protocol (CoAP) that mitigate security issues in particular use cases. The Echo option enables a CoAP server to verify the freshness of a request or to force a client to demonstrate reachability at its claimed network address. The Request-Tag option allows the CoAP server to match Block-Wise message fragments belonging to the same request. The updated Token processing requirements for clients ensure secure binding of responses to requests when CoAP is used with security.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on November 7, 2019.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of

publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

| | |
|--|----|
| 1. Introduction | 2 |
| 1.1. Request Freshness | 3 |
| 1.2. Fragmented Message Body Integrity | 4 |
| 1.3. Request-Response Binding | 4 |
| 1.4. Terminology | 5 |
| 2. The Echo Option | 6 |
| 2.1. Option Format | 6 |
| 2.2. Echo Processing | 7 |
| 2.3. Applications | 10 |
| 3. The Request-Tag Option | 11 |
| 3.1. Option Format | 11 |
| 3.2. Request-Tag Processing by Servers | 12 |
| 3.3. Setting the Request-Tag | 13 |
| 3.4. Applications | 14 |
| 3.4.1. Body Integrity Based on Payload Integrity | 14 |
| 3.4.2. Multiple Concurrent Blockwise Operations | 15 |
| 3.4.3. Simplified Block-Wise Handling for Constrained Proxies | 16 |
| 3.5. Rationale for the Option Properties | 16 |
| 3.6. Rationale for Introducing the Option | 16 |
| 4. Block2 / ETag Processing | 17 |
| 5. Token Processing | 17 |
| 6. Security Considerations | 17 |
| 7. Privacy Considerations | 18 |
| 8. IANA Considerations | 19 |
| 9. References | 19 |
| 9.1. Normative References | 19 |
| 9.2. Informative References | 19 |
| Appendix A. Methods for Generating Echo Option Values | 21 |
| Appendix B. Request-Tag Message Size Impact | 22 |
| Appendix C. Change Log | 22 |
| Acknowledgments | 24 |
| Authors' Addresses | 24 |

1. Introduction

The initial Constrained Application Protocol (CoAP) suite of specifications ([RFC7252], [RFC7641], and [RFC7959]) was designed with the assumption that security could be provided on a separate

layer, in particular by using DTLS ([RFC6347]). However, for some use cases, additional functionality or extra processing is needed to support secure CoAP operations. This document specifies security enhancements to the Constrained Application Protocol (CoAP).

This document specifies two CoAP options, the Echo option and the Request-Tag option: The Echo option enables a CoAP server to verify the freshness of a request, synchronize state, or force a client to demonstrate reachability at its claimed network address. The Request-Tag option allows the CoAP server to match message fragments belonging to the same request, fragmented using the CoAP Block-Wise Transfer mechanism, which mitigates attacks and enables concurrent blockwise operations. These options in themselves do not replace the need for a security protocol; they specify the format and processing of data which, when integrity protected using e.g. DTLS ([RFC6347]), TLS ([RFC8446]), or OSCORE ([I-D.ietf-core-object-security]), provide the additional security features.

The document also updates the Token processing requirements for clients specified in [RFC7252]. The updated processing ensures secure binding of responses to requests when CoAP is used with security, thus mitigating error cases and attacks where the client may erroneously associate the wrong response to a request.

1.1. Request Freshness

A CoAP server receiving a request is in general not able to verify when the request was sent by the CoAP client. This remains true even if the request was protected with a security protocol, such as DTLS. This makes CoAP requests vulnerable to certain delay attacks which are particularly perilous in the case of actuators ([I-D.mattsson-core-coap-actuators]). Some attacks can be mitigated by establishing fresh session keys, e.g. performing a DTLS handshake for each request, but in general this is not a solution suitable for constrained environments, for example, due to increased message overhead and latency. Additionally, if there are proxies, fresh DTLS session keys between server and proxy does not say anything about when the client made the request. In a general hop-by-hop setting, freshness may need to be verified in each hop.

A straightforward mitigation of potential delayed requests is that the CoAP server rejects a request the first time it appears and asks the CoAP client to prove that it intended to make the request at this point in time. The Echo option, defined in this document, specifies such a mechanism which thereby enables a CoAP server to verify the freshness of a request. This mechanism is not only important in the case of actuators, or other use cases where the CoAP operations require freshness of requests, but also in general for synchronizing

state between CoAP client and server, verify aliveness of the client, or force a client to demonstrate reachability at its claimed network address. The same functionality can be provided by echoing freshness tokens in CoAP payloads, but this only works for methods and response codes defined to have a payload. The Echo option provides a convention to transfer freshness tokens that works for all methods and response codes.

1.2. Fragmented Message Body Integrity

CoAP was designed to work over unreliable transports, such as UDP, and include a lightweight reliability feature to handle messages which are lost or arrive out of order. In order for a security protocol to support CoAP operations over unreliable transports, it must allow out-of-order delivery of messages using e.g. a sliding replay window such as described in Section 4.1.2.6 of DTLS ([RFC6347]).

The Block-Wise Transfer mechanism [RFC7959] extends CoAP by defining the transfer of a large resource representation (CoAP message body) as a sequence of blocks (CoAP message payloads). The mechanism uses a pair of CoAP options, Block1 and Block2, pertaining to the request and response payload, respectively. The blockwise functionality does not support the detection of interchanged blocks between different message bodies to the same resource having the same block number. This remains true even when CoAP is used together with a security protocol such as DTLS or OSCORE, within the replay window ([I-D.mattsson-core-coap-actuators]), which is a vulnerability of CoAP when using RFC7959.

A straightforward mitigation of mixing up blocks from different messages is to use unique identifiers for different message bodies, which would provide equivalent protection to the case where the complete body fits into a single payload. The ETag option [RFC7252], set by the CoAP server, identifies a response body fragmented using the Block2 option. This document defines the Request-Tag option for identifying request bodies, similar to ETag, but ephemeral and set by the CoAP client. The Request-Tag option is only used in requests that carry the Block1 option, and in Block2 requests following these.

1.3. Request-Response Binding

A fundamental requirement of secure REST operations is that the client can bind a response to a particular request. If this is not ensured, a client may erroneously associate the wrong response to a request. The wrong response may be an old response for the same resource or for a completely different resource (see e.g. Section 2.3 of [I-D.mattsson-core-coap-actuators]). For example, a

request for the alarm status "GET /status" may be associated to a prior response "on", instead of the correct response "off".

In HTTPS, this type of binding is always assured by the ordered and reliable delivery as well as mandating that the server sends responses in the same order that the requests were received. The same is not true for CoAP where the server (or an attacker) can return responses in any order and where there can be any number of responses to a request (see e.g. [RFC7641]). In CoAP, concurrent requests are differentiated by their Token. Note that the CoAP Message ID cannot be used for this purpose since those are typically different for REST request and corresponding response in case of "separate response", see Section 2.2 of [RFC7252].

CoAP [RFC7252] does not treat Token as a cryptographically important value and does not give stricter guidelines than that the tokens currently "in use" SHOULD (not SHALL) be unique. If used with a security protocol not providing bindings between requests and responses (e.g. DTLS and TLS) token reuse may result in situations where a client matches a response to the wrong request. Note that mismatches can also happen for other reasons than a malicious attacker, e.g. delayed delivery or a server sending notifications to an uninterested client.

A straightforward mitigation is to mandate clients to not reuse tokens until the traffic keys have been replaced. One easy way to accomplish this is to implement the token as a counter starting at zero for each new or rekeyed secure connection. This document updates the Token processing in [RFC7252] to always assure a cryptographically secure binding of responses to requests for secure REST operations like "coaps".

1.4. 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 BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

Unless otherwise specified, the terms "client" and "server" refers to "CoAP client" and "CoAP server", respectively, as defined in [RFC7252]. The term "origin server" is used as in [RFC7252]. The term "origin client" is used in this document to denote the client from which a request originates; to distinguish from clients in proxies.

The terms "payload" and "body" of a message are used as in [RFC7959]. The complete interchange of a request and a response body is called a (REST) "operation". An operation fragmented using [RFC7959] is called a "blockwise operation". A blockwise operation which is fragmenting the request body is called a "blockwise request operation". A blockwise operation which is fragmenting the response body is called a "blockwise response operation".

Two request messages are said to be "matchable" if they occur between the same endpoint pair, have the same code and the same set of options except for elective NoCacheKey options and options involved in block-wise transfer (Block1, Block2 and Request-Tag). Two operations are said to be matchable if any of their messages are.

Two matchable blockwise operations are said to be "concurrent" if a block of the second request is exchanged even though the client still intends to exchange further blocks in the first operation. (Concurrent blockwise request operations are impossible with the options of [RFC7959] because the second operation's block overwrites any state of the first exchange.)

The Echo and Request-Tag options are defined in this document.

2. The Echo Option

A fresh request is one whose age has not yet exceeded the freshness requirements set by the server. The freshness requirements are application specific and may vary based on resource, method, and parameters outside of coap such as policies. The Echo option is a lightweight challenge-response mechanism for CoAP, motivated by a need for a server to verify freshness of a request as described in Section 1.1. The Echo option value is a challenge from the server to the client included in a CoAP response and echoed back to the server in one or more CoAP requests. The Echo option provides a convention to transfer freshness tokens that works for all CoAP methods and response codes.

2.1. Option Format

The Echo Option is elective, safe-to-forward, not part of the cache-key, and not repeatable, see Figure 1, which extends Table 4 of [RFC7252]).

| No. | C | U | N | R | Name | Format | Len. | Default | E | U |
|-----|---|---|---|---|------|--------|------|---------|---|---|
| TBD | | | x | | Echo | opaque | 4-40 | (none) | x | x |

C = Critical, U = Unsafe, N = NoCacheKey, R = Repeatable,
E = Encrypt and Integrity Protect (when using OSCORE)

Figure 1: Echo Option Summary

[Note to RFC editor: If this document is released before core-object-security, then the following paragraph and the "E"/"U" columns above need to move into core-object-security, as they are defined in that draft.]

The Echo option value is generated by a server, and its content and structure are implementation specific. Different methods for generating Echo option values are outlined in Appendix A. Clients and intermediaries MUST treat an Echo option value as opaque and make no assumptions about its content or structure.

When receiving an Echo option in a request, the server MUST be able to verify when the Echo option value was generated. This implies that the server MUST be able to verify that the Echo option value was generated by the server or some other party that the server trusts. Depending on the freshness requirements the server may verify exactly when the Echo option value was generated (time-based freshness) or verify that the Echo option was generated after a specific event (event-based freshness). As the request is bound to the Echo option value, the server can determine that the request is not older than the Echo option value.

When the Echo option is used with OSCORE [I-D.ietf-core-object-security] it MAY be an Inner or Outer option, and the Inner and Outer values are independent. The Inner option is encrypted and integrity protected between the endpoints, whereas the Outer option is not protected by OSCORE and visible between the endpoints to the extent it is not protected by some other security protocol. E.g. in the case of DTLS hop-by-hop between the endpoints, the Outer option is visible to proxies along the path.

2.2. Echo Processing

The Echo option MAY be included in any request or response (see Section 2.3 for different applications), but the Echo option MUST NOT be used with empty CoAP requests (i.e., Code=0.00).

The application decides under what conditions a CoAP request to a resource is required to be fresh. These conditions can for example include what resource is requested, the request method and other data in the request, and conditions in the environment such as the state of the server or the time of the day.

If a certain request is required to be fresh, the request does not contain a fresh Echo option value, and the server cannot verify the freshness of the request in some other way, the server **MUST NOT** process the request further and **SHOULD** send a 4.01 Unauthorized response with an Echo option. The server **MAY** include the same Echo option value in several different responses and to different clients.

The server may use request freshness provided by the Echo option to verify the aliveness of a client or to synchronize state. The server may also include the Echo option in a response to force a client to demonstrate reachability at its claimed network address.

Upon receiving a 4.01 Unauthorized response with the Echo option, the client **SHOULD** resend the original request with the addition of an Echo option with the received Echo option value. The client **MAY** send a different request compared to the original request. Upon receiving any other response with the Echo option, the client **SHOULD** echo the Echo option value in the next request to the server. The client **MAY** include the same Echo option value in several different requests to the server.

Upon receiving a request with the Echo option, the server determines if the request is required to be fresh. If not, the Echo option **MAY** be ignored. If the request is required to be fresh and the server cannot verify the freshness of the request in some other way, the server **MUST** use the Echo option to verify that the request is fresh enough. If the server cannot verify that the request is fresh enough, the request is not processed further, and an error message **MAY** be sent. The error message **SHOULD** include a new Echo option.

One way for the server to verify freshness is that to bind the Echo value to a specific point in time and verify that the request is not older than a certain threshold T . The server can verify this by checking that $(t1 - t0) < T$, where $t1$ is the request receive time and $t0$ is the time when the Echo option value was generated. An example message flow is illustrated in Figure 2.

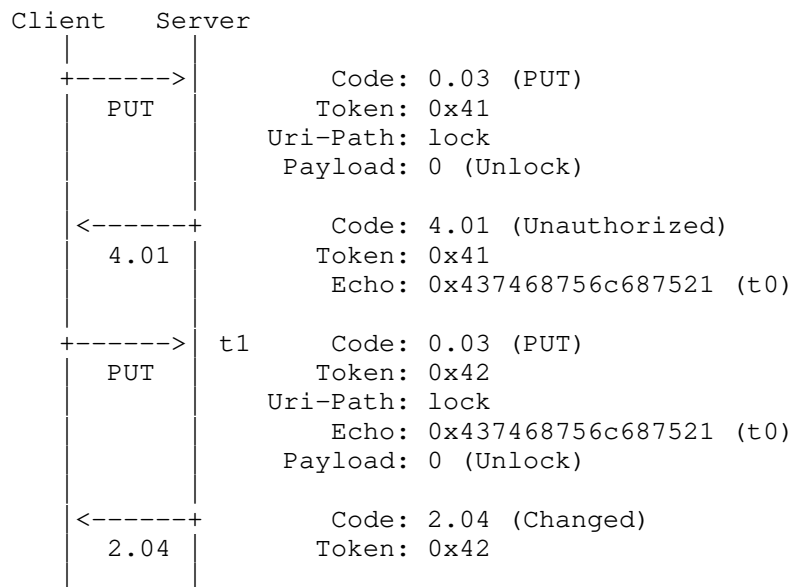


Figure 2: Example Echo Option Message Flow

When used to serve freshness requirements (including client aliveness and state synchronizing), CoAP messages containing the Echo option **MUST** be integrity protected between the intended endpoints, e.g. using DTLS, TLS, or an OSCORE Inner option ([I-D.ietf-core-object-security]). When used to demonstrate reachability at a claimed network address, the Echo option **SHOULD** contain the client's network address, but **MAY** be unprotected.

A CoAP-to-CoAP proxy **MAY** respond to requests with 4.01 with an Echo option to ensure the client's reachability at its claimed address, and **MUST** remove the Echo option it recognizes as one generated by itself on follow-up requests. However, it **MUST** relay the Echo option of responses unmodified, and **MUST** relay the Echo option of requests it does not recognize as generated by itself unmodified.

The CoAP server side of CoAP-to-HTTP proxies **MAY** request freshness, especially if they have reason to assume that access may require it (e.g. because it is a PUT or POST); how this is determined is out of scope for this document. The CoAP client side of HTTP-to-CoAP proxies **SHOULD** respond to Echo challenges themselves if they know from the recent establishing of the connection that the HTTP request is fresh. Otherwise, they **SHOULD** respond with 503 Service Unavailable, Retry-After: 0 and terminate any underlying Keep-Alive connection. They **MAY** also use other mechanisms to establish freshness of the HTTP request that are not specified here.

2.3. Applications

1. Actuation requests often require freshness guarantees to avoid accidental or malicious delayed actuator actions. In general, all non-safe methods (e.g. POST, PUT, DELETE) may require freshness guarantees for secure operation.
 - * The same Echo value may be used for multiple actuation requests to the same server, as long as the total round-trip time since the Echo option value was generated is below the freshness threshold.
 - * For actuator applications with low delay tolerance, to avoid additional round-trips for multiple requests in rapid sequence, the server may include the Echo option with a new value in response to a request containing the Echo option. The client then uses the Echo option with the new value in the next actuation request, and the server compares the receive time accordingly.
2. A server may use the Echo option to synchronize state or time with a requesting client. A server MUST NOT synchronize state or time with clients which are not the authority of the property being synchronized. E.g. if access to a server resource is dependent on time, then the client MUST NOT set the time of the server.
 - * If a server reboots during operation it may need to synchronize state or time before continuing the interaction. For example, with OSCORE it is possible to reuse a partly persistently stored security context by synchronizing the Partial IV (sequence number) using the Echo option, see Section 7.5 of [I-D.ietf-core-object-security].
 - * A device joining a CoAP group communication [RFC7390] protected with OSCORE [I-D.ietf-core-oscore-groupcomm] may be required to initially verify freshness and synchronize state or time with a client by using the Echo option in a unicast response to a multicast request. The client receiving the response with the Echo option includes the Echo option with the same value in a request, either in a unicast request to the responding server, or in a subsequent group request. In the latter case, the Echo option will be ignored expect by responding server.
3. A server that sends large responses to unauthenticated peers SHOULD mitigate amplification attacks such as described in Section 11.3 of [RFC7252] (where an attacker would put a victim's

address in the source address of a CoAP request). For this purpose, a server MAY ask a client to Echo its request to verify its source address. This needs to be done only once per peer and limits the range of potential victims from the general Internet to endpoints that have been previously in contact with the server. For this application, the Echo option can be used in messages that are not integrity protected, for example during discovery.

- * In the presence of a proxy, a server will not be able to distinguish different origin client endpoints. Following from the recommendation above, a proxy that sends large responses to unauthenticated peers SHOULD mitigate amplification attacks. The proxy MAY use Echo to verify origin reachability as described in Section 2.2. The proxy MAY forward idempotent requests immediately to have a cached result available when the client's Echoed request arrives.

4. A server may want to use the request freshness provided by the Echo to verify the aliveness of a client. Note that in a deployment with hop-by-hop security and proxies, the server can only verify aliveness of the closest proxy.

3. The Request-Tag Option

The Request-Tag is intended for use as a short-lived identifier for keeping apart distinct blockwise request operations on one resource from one client, addressing the issue described in Section 1.2. It enables the receiving server to reliably assemble request payloads (blocks) to their message bodies, and, if it chooses to support it, to reliably process simultaneous blockwise request operations on a single resource. The requests must be integrity protected in order to protect against interchange of blocks between different message bodies.

In essence, it is an implementation of the "proxy-safe elective option" used just to "vary the cache key" as suggested in [RFC7959] Section 2.4.

3.1. Option Format

The Request-Tag option is not critical, is safe to forward, repeatable, and part of the cache key, see Figure 3, which extends Table 4 of [RFC7252]).

| No. | C | U | N | R | Name | Format | Len. | Default | E | U |
|-----|---|---|---|---|-------------|--------|------|---------|---|---|
| TBD | | | | x | Request-Tag | opaque | 0-8 | (none) | x | x |

C = Critical, U = Unsafe, N = NoCacheKey, R = Repeatable,
E = Encrypt and Integrity Protect (when using OSCORE)

Figure 3: Request-Tag Option Summary

[Note to RFC editor: If this document is released before core-object-security, then the following paragraph and the "E"/"U" columns above need to move into core-object-security, as they are defined in that draft.]

Request-Tag, like the block options, is both a class E and a class U option in terms of OSCORE processing (see Section 4.1 of [I-D.ietf-core-object-security]): The Request-Tag MAY be an inner or outer option. It influences the inner or outer block operation, respectively. The inner and outer values are therefore independent of each other. The inner option is encrypted and integrity protected between client and server, and provides message body identification in case of end-to-end fragmentation of requests. The outer option is visible to proxies and labels message bodies in case of hop-by-hop fragmentation of requests.

The Request-Tag option is only used in the request messages of blockwise operations.

The Request-Tag mechanism can be applied independently on the server and client sides of CoAP-to-CoAP proxies as are the block options, though given it is safe to forward, a proxy is free to just forward it when processing an operation. CoAP-to-HTTP proxies and HTTP-to-CoAP proxies can use Request-Tag on their CoAP sides; it is not applicable to HTTP requests.

3.2. Request-Tag Processing by Servers

The Request-Tag option does not require any particular processing on the server side outside of the processing already necessary for any unknown elective proxy-safe cache-key option: The option varies the properties that distinguish blockwise operations (which includes all options except elective NoCacheKey and except Block1/2), and thus the server can not treat messages with a different list of Request-Tag options as belonging to the same operation.

To keep utilizing the cache, a server (including proxies) MAY discard the Request-Tag option from an assembled block-wise request when consulting its cache, as the option relates to the operation-on-the-wire and not its semantics. For example, a FETCH request with the same body as an older one can be served from the cache if the older's Max-Age has not expired yet, even if the second operation uses a Request-Tag and the first did not. (This is similar to the situation about ETag in that it is formally part of the cache key, but implementations that are aware of its meaning can cache more efficiently, see [RFC7252] Section 5.4.2).

A server receiving a Request-Tag MUST treat it as opaque and make no assumptions about its content or structure.

Two messages carrying the same Request-Tag is a necessary but not sufficient condition for being part of the same operation. They can still be treated as independent messages by the server (e.g. when it sends 2.01/2.04 responses for every block), or initiate a new operation (overwriting kept context) when the later message carries Block1 number 0.

As it has always been, a server that can only serve a limited number of block-wise operations at the same time can delay the start of the operation by replying with 5.03 (Service unavailable) and a Max-Age indicating how long it expects the existing operation to go on, or it can forget about the state established with the older operation and respond with 4.08 (Request Entity Incomplete) to later blocks on the first operation.

3.3. Setting the Request-Tag

For each separate blockwise request operation, the client can choose a Request-Tag value, or choose not to set a Request-Tag. Starting a request operation matchable to a previous operation and even using the same Request-Tag value is called request tag recycling. The absence of a Request-Tag option is viewed as a value distinct from all values with a single Request-Tag option set; starting a request operation matchable to a previous operation where neither has a Request-Tag option therefore constitutes request tag recycling just as well (also called "recycling the absent option").

Clients MUST NOT recycle a request tag unless the first operation has concluded. What constitutes a concluded operation depends on the application, and is outlined individually in Section 3.4.

When Block1 and Block2 are combined in an operation, the Request-Tag of the Block1 phase is set in the Block2 phase as well for otherwise

the request would have a different set of options and would not be recognized any more.

Clients are encouraged to generate compact messages. This means sending messages without Request-Tag options whenever possible, and using short values when the absent option can not be recycled.

The Request-Tag options MAY be present in request messages that carry a Block2 option even if those messages are not part of a blockwise request operation (this is to allow the operation described in Section 3.4.3). The Request-Tag option MUST NOT be present in response messages, and MUST NOT be present if neither the Block1 nor the Block2 option is present.

3.4. Applications

3.4.1. Body Integrity Based on Payload Integrity

When a client fragments a request body into multiple message payloads, even if the individual messages are integrity protected, it is still possible for a man-in-the-middle to maliciously replace a later operation's blocks with an earlier operation's blocks (see Section 2.5 of [I-D.mattsson-core-coap-actuators]). Therefore, the integrity protection of each block does not extend to the operation's request body.

In order to gain that protection, use the Request-Tag mechanism as follows:

- o The individual exchanges MUST be integrity protected end-to-end between client and server.
- o The client MUST NOT recycle a request tag in a new operation unless the previous operation matchable to the new one has concluded.

If any future security mechanisms allow a block-wise transfer to continue after an endpoint's details (like the IP address) have changed, then the client MUST consider messages sent to `_any_` endpoint address within the new operation's security context.

- o The client MUST NOT regard a blockwise request operation as concluded unless all of the messages the client previously sent in the operation have been confirmed by the message integrity protection mechanism, or are considered invalid by the server if replayed.

Typically, in OSCORE, these confirmations can result either from the client receiving an OSCORE response message matching the request (an empty ACK is insufficient), or because the message's sequence number is old enough to be outside the server's receive window.

In DTLS, this can only be confirmed if the request message was not retransmitted, and was responded to.

Authors of other documents (e.g. [I-D.ietf-core-object-security]) are invited to mandate this behavior for clients that execute blockwise interactions over secured transports. In this way, the server can rely on a conforming client to set the Request-Tag option when required, and thereby conclude on the integrity of the assembled body.

Note that this mechanism is implicitly implemented when the security layer guarantees ordered delivery (e.g. CoAP over TLS [RFC8323]). This is because with each message, any earlier message can not be replayed any more, so the client never needs to set the Request-Tag option unless it wants to perform concurrent operations.

3.4.2. Multiple Concurrent Blockwise Operations

CoAP clients, especially CoAP proxies, may initiate a blockwise request operation to a resource, to which a previous one is already in progress, which the new request should not cancel. A CoAP proxy would be in such a situation when it forwards operations with the same cache-key options but possibly different payloads.

For those cases, Request-Tag is the proxy-safe elective option suggested in [RFC7959] Section 2.4 last paragraph.

When initializing a new blockwise operation, a client has to look at other active operations:

- o If any of them is matchable to the new one, and the client neither wants to cancel the old one nor postpone the new one, it can pick a Request-Tag value that is not in use by the other matchable operations for the new operation.
- o Otherwise, it can start the new operation without setting the Request-Tag option on it.

3.4.3. Simplified Block-Wise Handling for Constrained Proxies

The Block options were defined to be unsafe to forward because a proxy that would forward blocks as plain messages would risk mixing up clients' requests.

The Request-Tag option provides a very simple way for a proxy to keep them separate: if it appends a Request-Tag that is particular to the requesting endpoint to all request carrying any Block option, it does not need to keep track of any further block state.

This is particularly useful to proxies that strive for stateless operation as described in [I-D.hartke-core-stateless] Section 3.1.

3.5. Rationale for the Option Properties

The Request-Tag option can be elective, because to servers unaware of the Request-Tag option, operations with differing request tags will not be matchable.

The Request-Tag option can be safe to forward but part of the cache key, because to proxies unaware of the Request-Tag option will consider operations with differing request tags unmatchable but can still forward them.

The Request-Tag option is repeatable because this easily allows stateless proxies to "chain" their origin address. They can perform the steps of Section 3.4.3 without the need to create an option value that is the concatenation of the received option and their own value, and can simply add a new Request-Tag option unconditionally.

In draft versions of this document, the Request-Tag option used to be critical and unsafe to forward. That design was based on an erroneous understanding of which blocks could be composed according to [RFC7959].

3.6. Rationale for Introducing the Option

An alternative that was considered to the Request-Tag option for coping with the problem of fragmented message body integrity (Section 3.4.1) was to update [RFC7959] to say that blocks could only be assembled if their fragments' order corresponded to the sequence numbers.

That approach would have been difficult to roll out reliably on DTLS where many implementations do not expose sequence numbers, and would still not prevent attacks like in [I-D.mattsson-core-coap-actuators] Section 2.5.2.

4. Block2 / ETag Processing

The same security properties as in Section 3.4.1 can be obtained for blockwise response operations. The threat model here is not an attacker (because the response is made sure to belong to the current request by the security layer), but blocks in the client's cache.

Rules stating that response body reassembly is conditional on matching ETag values are already in place from Section 2.4 of [RFC7959].

To gain equivalent protection to Section 3.4.1, a server **MUST** use the Block2 option in conjunction with the ETag option ([RFC7252], Section 5.10.6), and **MUST NOT** use the same ETag value for different representations of a resource.

5. Token Processing

As described in Section 1.3, the client must be able to verify that a response corresponds to a particular request. This section updates the CoAP Token processing requirements for clients. The Token processing for servers is not updated. Token processing in Section 5.3.1 of [RFC7252] is updated by adding the following text:

When CoAP is used with a security protocol not providing bindings between requests and responses, the tokens have cryptographic importance. The client **MUST** make sure that tokens are not used in a way so that responses risk being associated with the wrong request. One easy way to accomplish this is to implement the Token (or part of the Token) as a sequence number starting at zero for each new or rekeyed secure connection, this approach **SHOULD** be followed.

6. Security Considerations

The availability of a secure pseudorandom number generator and truly random seeds are essential for the security of the Echo option. If no true random number generator is available, a truly random seed must be provided from an external source.

A single active Echo value with 64 (pseudo-)random bits gives the same theoretical security level against forgeries as a 64-bit MAC (as used in e.g. AES_128_CCM_8). In practice, forgery of an Echo option value is much harder as an attacker must also forge the MAC in the security protocol. The Echo option value **MUST** contain 32 (pseudo-)random bits that are not predictable for any other party than the server, and **SHOULD** contain 64 (pseudo-)random bits. A server **MAY** use different security levels for different uses cases

(client aliveness, request freshness, state synchronization, network address reachability, etc.).

The security provided by the Echo and Request-Tag options depends on the security protocol used. CoAP and HTTP proxies require (D)TLS to be terminated at the proxies. The proxies are therefore able to manipulate, inject, delete, or reorder options or packets. The security claims in such architectures only hold under the assumption that all intermediaries are fully trusted and have not been compromised.

Servers SHOULD use a monotonic clock to generate timestamps and compute round-trip times. Use of non-monotonic clocks is not secure as the server will accept expired Echo option values if the clock is moved backward. The server will also reject fresh Echo option values if the clock is moved forward. Non-monotonic clocks MAY be used as long as they have deviations that are acceptable given the freshness requirements. If the deviations from a monotonic clock are known, it may be possible to adjust the threshold accordingly.

Servers SHOULD NOT use wall clock time for timestamps, as wall clock time have large deviations from a monotonic clock. Furthermore, an attacker may be able to affect the server's wall clock time in various ways such as setting up a fake NTP server or broadcasting false time signals to radio-controlled clocks.

Servers MAY use the time since reboot measured in some unit of time. Servers MAY reset the timer at certain times and MAY generate a random offset applied to all timestamps. When resetting the timer, the server MUST reject all Echo values that was created before the reset.

Servers that use the List of Cached Random Values and Timestamps method described in Appendix A may be vulnerable to resource exhaustion attacks. One way to minimize state is to use the Integrity Protected Timestamp method described in Appendix A.

When the Token (or part of the Token) contains a sequence number, the encoding of the sequence number has to be chosen in a way to avoid any collisions. This is especially true when the Token contains more information than just the sequence number (e.g. serialized state).

7. Privacy Considerations

Implementations SHOULD NOT put any privacy sensitive information in the Echo or Request-Tag option values. Unencrypted timestamps MAY reveal information about the server such as location or time since reboot. The use of wall clock time is not allowed (see Section 6)

and there also privacy reasons, e.g. it may reveal that the server will accept expired certificates. Timestamps MAY be used if Echo is encrypted between the client and the server, e.g. in the case of DTLS without proxies or when using OSCORE with an Inner Echo option.

8. IANA Considerations

This document adds the following option numbers to the "CoAP Option Numbers" registry defined by [RFC7252]:

| Number | Name | Reference |
|--------|-------------|-------------------|
| TBD1 | Echo | [[this document]] |
| TBD2 | Request-Tag | [[this document]] |

Figure 4: CoAP Option Numbers

9. References

9.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<https://www.rfc-editor.org/info/rfc7252>>.
- [RFC7959] Bormann, C. and Z. Shelby, Ed., "Block-Wise Transfers in the Constrained Application Protocol (CoAP)", RFC 7959, DOI 10.17487/RFC7959, August 2016, <<https://www.rfc-editor.org/info/rfc7959>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.

9.2. Informative References

- [I-D.hartke-core-stateless]
Hartke, K., "Extended Tokens and Stateless Clients in the Constrained Application Protocol (CoAP)", draft-hartke-core-stateless-02 (work in progress), October 2018.
- [I-D.ietf-core-object-security]
Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security for Constrained RESTful Environments (OSCORE)", draft-ietf-core-object-security-16 (work in progress), March 2019.
- [I-D.ietf-core-oscore-groupcomm]
Tiloca, M., Selander, G., Palombini, F., and J. Park, "Group OSCORE - Secure Group Communication for CoAP", draft-ietf-core-oscore-groupcomm-04 (work in progress), March 2019.
- [I-D.mattsson-core-coap-actuators]
Mattsson, J., Fornehed, J., Selander, G., Palombini, F., and C. Amsuess, "Controlling Actuators with CoAP", draft-mattsson-core-coap-actuators-06 (work in progress), September 2018.
- [RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <<https://www.rfc-editor.org/info/rfc6347>>.
- [RFC7390] Rahman, A., Ed. and E. Dijk, Ed., "Group Communication for the Constrained Application Protocol (CoAP)", RFC 7390, DOI 10.17487/RFC7390, October 2014, <<https://www.rfc-editor.org/info/rfc7390>>.
- [RFC7641] Hartke, K., "Observing Resources in the Constrained Application Protocol (CoAP)", RFC 7641, DOI 10.17487/RFC7641, September 2015, <<https://www.rfc-editor.org/info/rfc7641>>.
- [RFC8323] Bormann, C., Lemay, S., Tschafenig, H., Hartke, K., Silverajan, B., and B. Raymor, Ed., "CoAP (Constrained Application Protocol) over TCP, TLS, and WebSockets", RFC 8323, DOI 10.17487/RFC8323, February 2018, <<https://www.rfc-editor.org/info/rfc8323>>.
- [RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <<https://www.rfc-editor.org/info/rfc8446>>.

Appendix A. Methods for Generating Echo Option Values

The content and structure of the Echo option value are implementation specific and determined by the server. Two simple mechanisms are outlined in this section, the first is RECOMMENDED in general, and the second is RECOMMENDED in case the Echo option is encrypted between the client and the server.

Different mechanisms have different tradeoffs between the size of the Echo option value, the amount of server state, the amount of computation, and the security properties offered. A server MAY use different methods and security levels for different uses cases (client aliveness, request freshness, state synchronization, network address reachability, etc.).

1. List of Cached Random Values and Timestamps. The Echo option value is a (pseudo-)random byte string. The server caches a list containing the random byte strings and their transmission times. Assuming 72-bit random values and 32-bit timestamps, the size of the Echo option value is 9 bytes and the amount of server state is $13n$ bytes, where n is the number of active Echo Option values. The security against forged echo values is given by $s = \text{bit length of } r - \log_2(n)$. The length of r and the maximum allowed n should be set so that the security level is harmonized with other parts of the deployment, e.g., $s \geq 64$. If the server loses time continuity, e.g. due to reboot, the entries in the old list MUST be deleted.

Echo option value: random value r
Server State: random value r , timestamp t_0

2. Integrity Protected Timestamp. The Echo option value is an integrity protected timestamp. The timestamp can have different resolution and range. A 32-bit timestamp can e.g. give a resolution of 1 second with a range of 136 years. The (pseudo-)random secret key is generated by the server and not shared with any other party. The use of truncated HMAC-SHA-256 is RECOMMENDED. With a 32-bit timestamp and a 64-bit MAC, the size of the Echo option value is 12 bytes and the Server state is small and constant. The security against forged echo values is given by the MAC length. If the server loses time continuity, e.g. due to reboot, the old key MUST be deleted and replaced by a new random secret key. Note that the privacy considerations in Section 7 may apply to the timestamp. A server MAY want to encrypt its timestamps, and, depending on the choice of encryption algorithms, this may require a nonce to be included in the Echo option value.

Echo option value: timestamp t_0 , MAC(k , t_0)
Server State: secret key k

Other mechanisms complying with the security and privacy considerations may be used. The use of encrypted timestamps in the Echo option typically requires an IV to be included in the Echo option value, which adds overhead and makes the specification of such a mechanism slightly more complicated than the two mechanisms specified here.

Appendix B. Request-Tag Message Size Impact

In absence of concurrent operations, the Request-Tag mechanism for body integrity (Section 3.4.1) incurs no overhead if no messages are lost (more precisely: in OSCORE, if no operations are aborted due to repeated transmission failure; in DTLS, if no packages are lost), or when blockwise request operations happen rarely (in OSCORE, if there is always only one request blockwise operation in the replay window).

In those situations, no message has any Request-Tag option set, and that can be recycled indefinitely.

When the absence of a Request-Tag option cannot be recycled any more within a security context, the messages with a present but empty Request-Tag option can be used (1 Byte overhead), and when that is used-up, 256 values from one byte long options (2 Bytes overhead) are available.

In situations where those overheads are unacceptable (e.g. because the payloads are known to be at a fragmentation threshold), the absent Request-Tag value can be made usable again:

- o In DTLS, a new session can be established.
- o In OSCORE, the sequence number can be artificially increased so that all lost messages are outside of the replay window by the time the first request of the new operation gets processed, and all earlier operations can therefore be regarded as concluded.

Appendix C. Change Log

[The editor is asked to remove this section before publication.]

- o Changes since draft-ietf-core-echo-request-tag-04:
 - * Editorial fixes
 - + Moved paragraph on collision-free encoding of data in the token to Security Considerations and rephrased it
 - + "easiest" -> "one easy"

- o Changes since draft-ietf-core-echo-request-tag-03:
 - * Mention token processing changes in title
 - * Abstract reworded
 - * Clarify updates to token processing
 - * Describe security levels from Echo length
 - * Allow non-monotonic clocks under certain conditions for freshness
 - * Simplify freshness expressions
 - * Describe when a Request-Tag can be set
 - * Add note on application-level freshness mechanisms
 - * Minor editorial changes
- o Changes since draft-ietf-core-echo-request-tag-02:
 - * Define "freshness"
 - * Note limitations of "aliveness"
 - * Clarify proxy and OSCORE handling in presence of "echo"
 - * Clarify when Echo values may be reused
 - * Update security considerations
 - * Various minor clarifications
 - * Minor editorial changes
- o Major changes since draft-ietf-core-echo-request-tag-01:
 - * Follow-up changes after the "relying on blockwise" change in -01:
 - + Simplify the description of Request-Tag and matchability
 - + Do not update RFC7959 any more
 - * Make Request-Tag repeatable.

- * Add rationale on not relying purely on sequence numbers.
- o Major changes since draft-ietf-core-echo-request-tag-00:
 - * Reworded the Echo section.
 - * Added rules for Token processing.
 - * Added security considerations.
 - * Added actual IANA section.
 - * Made Request-Tag optional and safe-to-forward, relying on blockwise to treat it as part of the cache-key
 - * Dropped use case about OSCORE outer-blockwise (the case went away when its Partial IV was moved into the Object-Security option)
- o Major changes since draft-amsuess-core-repeat-request-tag-00:
 - * The option used for establishing freshness was renamed from "Repeat" to "Echo" to reduce confusion about repeatable options.
 - * The response code that goes with Echo was changed from 4.03 to 4.01 because the client needs to provide better credentials.
 - * The interaction between the new option and (cross) proxies is now covered.
 - * Two messages being "Request-Tag matchable" was introduced to replace the older concept of having a request tag value with its slightly awkward equivalence definition.

Acknowledgments

The authors want to thank Jim Schaad and Carsten Bormann for providing valuable input to the draft.

Authors' Addresses

Christian Amsuess

Email: christian@amsuess.com

John Mattsson
Ericsson AB

Email: john.mattsson@ericsson.com

Goeran Selander
Ericsson AB

Email: goran.selander@ericsson.com

CORE
Internet-Draft
Intended status: Standards Track
Expires: January 4, 2020

M. Boucadair
Orange
T. Reddy
McAfee
J. Shallow
July 3, 2019

Constrained Application Protocol (CoAP) Hop-Limit Option
draft-ietf-core-hop-limit-04

Abstract

The presence of Constrained Application Protocol (CoAP) proxies may lead to infinite forwarding loops, which is undesirable. To prevent and detect such loops, this document specifies the Hop-Limit CoAP option.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on January 4, 2020.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of

the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

| | |
|--|---|
| 1. Introduction | 2 |
| 1.1. Intended Usage | 2 |
| 2. Terminology | 3 |
| 3. Hop-Limit Option | 3 |
| 4. HTTP-Mapping Considerations | 5 |
| 5. IANA Considerations | 5 |
| 5.1. CoAP Response Code | 5 |
| 5.2. CoAP Option Number | 6 |
| 6. Security Considerations | 6 |
| 7. Acknowledgements | 6 |
| 8. References | 7 |
| 8.1. Normative References | 7 |
| 8.2. Informative References | 7 |
| Authors' Addresses | 8 |

1. Introduction

More and more applications are using the Constrained Application Protocol (CoAP) [RFC7252] as a communication protocol between involved application agents. For example, [I-D.ietf-dots-signal-channel] specifies how CoAP is used as a distributed denial-of-service (DDoS) attack signaling protocol for seeking for help from DDoS mitigation providers. In such contexts, a CoAP client can communicate directly with a server or indirectly via proxies.

When multiple proxies are involved, infinite forwarding loops may be experienced (e.g., routing misconfiguration, policy conflicts). To prevent such loops, this document defines a new CoAP option, called Hop-Limit (Section 3). Also, the document defines a new CoAP Response Code (Section 5.1) to report loops together with relevant diagnostic information to ease troubleshooting.

1.1. Intended Usage

The Hop-Limit option has originally been designed for a specific use case [I-D.ietf-dots-signal-channel]. However, its intended usage is general: CoAP proxies that do not have specific knowledge that proxy forwarding loops are avoided in some other way, are expected to implement this option and have it enabled by default.

Note that this means that a server that receives requests both via proxies and directly from clients may see otherwise identical

requests with and without the Hop-Limit option included; servers with internal caching will therefore also want to implement this option.

2. 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 BCP 14 [RFC2119][RFC8174] when, and only when, they appear in all capitals, as shown here.

Readers should be familiar with the terms and concepts defined in [RFC7252].

3. Hop-Limit Option

The Hop-Limit option (see Section 5.2) is an elective option used to detect and prevent infinite loops when proxies are involved. The option is not repeatable. Therefore, any message carrying multiple Hop-Limit options MUST be handled following the procedure specified in Section 5.4.5 of [RFC7252].

The value of the Hop-Limit option is encoded as an unsigned integer (see Section 3.2 of [RFC7252]). This value MUST be between 1 and 255 inclusive. CoAP messages received with a Hop-Limit option set to '0' or greater than '255' MUST be rejected by a CoAP server/proxy using 4.00 (Bad Request).

The Hop-Limit option is safe to forward. That is, a CoAP proxy which does not understand the Hop-Limit option should forward it on. The option is also part of the cache key. As such, a CoAP proxy which does not understand the Hop-Limit option must follow the recommendations in Section 5.7.1 of [RFC7252] for caching. Note that loops which involve only such proxies won't be detected. Nevertheless, the presence of such proxies won't prevent infinite loop detection if at least one CoAP proxy which support the Hop-Limit option is involved in the loop.

A CoAP proxy which understands the Hop-Limit option MAY be instructed, using a configuration parameter, to insert a Hop-Limit option when relaying a request which do not include the Hop-Limit option.

The initial Hop-Limit value should be configurable. If no initial value is explicitly provided, the default initial Hop-Limit value of 16 MUST be used. This value is chosen to be sufficiently large to guarantee that a CoAP request would not be dropped in networks when there were no loops, but not so large as to consume CoAP proxy

resources when a loop does occur. Lower values should be used with caution and only in networks where topologies are known by the CoAP client (or proxy) inserting the Hop-Limit option.

Because forwarding errors may occur if inadequate Hop-Limit values are used, proxies at the boundaries of an administrative domain MAY be instructed to remove or rewrite the value of Hop-Limit carried in received messages (i.e., ignore the value of Hop-Limit received in a message). This modification should be done with caution in case proxy-forwarded traffic repeatedly crosses the administrative domain boundary in a loop and so Hop-Limit detection gets broken.

Otherwise, a CoAP proxy which understands the Hop-Limit option MUST decrement the value of the option by 1 prior to forwarding it. A CoAP proxy which understands the Hop-Limit option MUST NOT use a stored TBA1 (Hop Limit Reached) error response unless the value of the Hop-Limit option in the presented request is less than or equal to the value of the Hop-Limit option in the request used to obtain the stored response. Otherwise, the CoAP proxy follows the behavior in Section 5.6 of [RFC7252].

Note: If a request with a given value of Hop-Limit failed to reach a server because the hop limit is exhausted, then the same failure will be observed if a less value of the Hop-Limit option is used instead.

CoAP messages MUST NOT be forwarded if the Hop-Limit option is set to '0' after decrement. Messages that cannot be forwarded because of exhausted Hop-Limit SHOULD be logged with a TBA1 (Hop Limit Reached) error response sent back to the CoAP peer. It is RECOMMENDED that CoAP implementations support means to alert administrators about loop errors so that appropriate actions are undertaken.

To ease debugging and troubleshooting, the CoAP proxy which detects a loop includes its information in the diagnostic payload under the conditions detailed in Section 5.5.2 of [RFC7252]. That information MUST NOT include any space character. The information inserted by a CoAP proxy can be, for example, a proxy name (e.g., p11.example.net), proxy alias (e.g., myproxyalias), or IP address (e.g., 2001:db8::1).

Each intermediate proxy involved in relaying a TBA1 (Hop Limit Reached) error message prepends its own information in the diagnostic payload with a space character used as separator. Only one information per proxy should appear in the diagnostic payload. Doing so allows to limit the size of the TBA1 (Hop Limit Reached) error message, and to ease correlation with hops count. Note that an intermediate proxy prepends its information only if there is enough space as determined by the Path MTU (Section 4.6 of [RFC7252]). If

not, an intermediate proxy forwards the TBA1 (Hop Limit Reached) error message to the next hop without updating the diagnostic payload.

4. HTTP-Mapping Considerations

This section focuses on the HTTP mappings specific to the CoAP extension specified in this document. As a reminder, the basic normative requirements on HTTP/CoAP mappings are defined in Section 10 of [RFC7252]. The implementation guidelines for HTTP/CoAP mappings are elaborated in [RFC8075].

By default, the HTTP-to-CoAP Proxy inserts a Hop-Limit option following the guidelines in Section 3. The HTTP-to-CoAP Proxy MAY be instructed by policy to insert a Hop-Limit option only if a Via (Section 5.7.1 of [RFC7230]) or CDN-Loop header field [RFC8586] is present in the HTTP request.

The HTTP-to-CoAP Proxy uses 508 (Loop Detected) as the HTTP response status code to map TBA1 (Hop Limit Reached). Furthermore, it maps the diagnostic payload of TBA1 (Hop Limit Reached) as per Section 6.6 of [RFC8075].

By default, the CoAP-to-HTTP Proxy inserts a Via header field in the HTTP request if the CoAP request includes a Hop-Limit option. The CoAP-to-HTTP Proxy MAY be instructed by policy to insert a CDN-Loop header field instead of the Via header field.

The CoAP-to-HTTP Proxy maps the 508 (Loop Detected) HTTP response status code to TBA1 (Hop Limit Reached). Moreover, the CoAP-to-HTTP Proxy inserts its information following the guidelines in Section 3.

When both HTTP-to-CoAP and CoAP-to-HTTP proxies are involved, the loop detection may get broken if the proxy-forwarded traffic repeatedly crosses the HTTP-to-CoAP and CoAP-to-HTTP proxies. Nevertheless, if the loop is within the CoAP or HTTP legs, the loop detection is still functional.

5. IANA Considerations

5.1. CoAP Response Code

IANA is requested to add the following entry to the "CoAP Response Codes" sub-registry available at <https://www.iana.org/assignments/core-parameters/core-parameters.xhtml#response-codes>:

| Code | Description | Reference |
|------|-------------------|-----------|
| TBA1 | Hop Limit Reached | [RFCXXXX] |

Table 1: CoAP Response Codes

This document suggests 5.08 as a code to be assigned for the new response code.

Editorial Note: Please update TBA1 statements within the document with the assigned code.

5.2. CoAP Option Number

IANA is requested to add the following entry to the "CoAP Option Numbers" sub-registry available at <https://www.iana.org/assignments/core-parameters/core-parameters.xhtml#option-numbers>:

| Number | C | U | N | R | Name | Reference |
|--------|---|---|---|---|-----------|-----------|
| TBA2 | | | | | Hop-Limit | [RFCXXXX] |

C=Critical, U=Unsafe, N=NoCacheKey, R=Repeatable

Table 2: CoAP Option Number

6. Security Considerations

Security considerations related to CoAP proxying are discussed in Section 11.2 of [RFC7252].

The diagnostic payload of a TBA1 (Hop Limit Reached) error message may leak sensitive information revealing the topology of an administrative domain. To prevent that, a CoAP proxy which is located at the boundary of an administrative domain MAY be instructed to strip the diagnostic payload or part of it before forwarding on the TBA1 (Hop Limit Reached) response.

7. Acknowledgements

This specification was part of [I-D.ietf-dots-signal-channel]. Many thanks to those who reviewed DOTS specifications.

Thanks to Klaus Hartke, Carsten Bormann, Peter van der Stok, and Jim Schaad for the reviews.

Carsten Bormann provided the "Intended Usage" text.

8. References

8.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <<https://www.rfc-editor.org/info/rfc7230>>.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<https://www.rfc-editor.org/info/rfc7252>>.
- [RFC8075] Castellani, A., Loreto, S., Rahman, A., Fossati, T., and E. Dijk, "Guidelines for Mapping Implementations: HTTP to the Constrained Application Protocol (CoAP)", RFC 8075, DOI 10.17487/RFC8075, February 2017, <<https://www.rfc-editor.org/info/rfc8075>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.

8.2. Informative References

- [I-D.ietf-dots-signal-channel] K, R., Boucadair, M., Patil, P., Mortensen, A., and N. Teague, "Distributed Denial-of-Service Open Threat Signaling (DOTS) Signal Channel Specification", draft-ietf-dots-signal-channel-34 (work in progress), May 2019.
- [RFC8586] Ludin, S., Nottingham, M., and N. Sullivan, "Loop Detection in Content Delivery Networks (CDNs)", RFC 8586, DOI 10.17487/RFC8586, April 2019, <<https://www.rfc-editor.org/info/rfc8586>>.

Authors' Addresses

Mohamed Boucadair
Orange
Rennes 35000
France

Email: mohamed.boucadair@orange.com

Tirumaleswar Reddy
McAfee, Inc.
Embassy Golf Link Business Park
Bangalore, Karnataka 560071
India

Email: kondtir@gmail.com

Jon Shallow
United Kingdom

Email: supjps-ietf@jpshallow.com

CoRE
Internet-Draft
Intended status: Standards Track
Expires: September 9, 2019

T. Fossati
ARM
K. Hartke
Ericsson
C. Bormann
Universitaet Bremen TZI
March 08, 2019

Multipart Content-Format for CoAP
draft-ietf-core-multipart-ct-03

Abstract

This memo defines application/multipart-core, an application-independent media-type that can be used to combine representations of zero or more different media types into a single message, such as a CoAP request or response body, with minimal framing overhead, each along with a CoAP Content-Format identifier.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 9, 2019.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must

include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

| | |
|--|---|
| 1. Introduction | 2 |
| 1.1. Requirements Language | 3 |
| 2. Multipart Content-Format Encoding | 3 |
| 3. Usage Examples | 4 |
| 3.1. Observing Resources | 4 |
| 4. Implementation hints | 4 |
| 5. IANA Considerations | 5 |
| 5.1. Registration of media type application/multipart-core . . | 5 |
| 5.2. Registration of a Content-Format identifier for application/multipart-core | 7 |
| 6. Security Considerations | 7 |
| 7. References | 7 |
| 7.1. Normative References | 7 |
| 7.2. Informative References | 7 |
| Acknowledgements | 8 |
| Authors' Addresses | 8 |

1. Introduction

This memo defines application/multipart-core, an application-independent media-type that can be used to combine representations of zero or more different media types into a single message, such as a CoAP [RFC7252] request or response body, with minimal framing overhead, each along with a CoAP Content-Format identifier.

This simple and efficient binary framing mechanism can be employed to create application specific request and response bodies which build on multiple already existing media types.

The individual representations in an application/multipart-core body occur in a sequence, which may be employed by an application where such a sequence is natural, e.g. for a number of audio snippets in different formats to be played out in that sequence.

In other cases, an application may be more interested in a bag of representations, which are distinguished by their Content-Format identifier, such as an audio snippet and a text representation accompanying it. In such a case, the sequence in which these occur may not be relevant to the application. This specification allows to indicate that an optional part is not present by substituting a null value for the representation of the part.

A third situation that is common only ever has a single representation in the sequence, which is one of a set of formats possible. This kind of union of formats may also make the presence of the actual representation optional, the omission of which leads to a zero-length array.

Where these rules are not sufficient for an application, it might still use the general format defined here, but register a new media type and an associated Content-Format identifier to associate the representation with these more specific semantics instead of using application/multipart-core.

1.1. Requirements Language

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 BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

2. Multipart Content-Format Encoding

A representation of media-type application/multipart-core contains a collection of zero or more representations, each along with their respective content format.

The collection is encoded as a CBOR [RFC7049] array with an even number of elements. The second, fourth, sixth, etc. element is a byte string containing a representation, or the value "null" if an optional part is indicated as not given. The first, third, fifth, etc. element is an unsigned integer specifying the content format ID of the representation following it.

For example, a collection containing two representations, one with content format ID 42 and one with content format ID 0, looks like this in CBOR diagnostic notation:

```
[42, h'0123456789abcdef', 0, h'3031323334']
```

For illustration, the structure of an application/multipart-core representation can be described by the CDDL [I-D.ietf-cbor-cddl] specification in Figure 1:

```
multipart-core = [* multipart-part]
multipart-part = (type: uint .size 2, part: bytes / null)
```

Figure 1: CDDL for application/multipart-core

This format is intended as a strict specification: An implementation **MUST** stop processing the representation if there is a CBOR well-formedness error, a deviation from the structure defined above, or any residual data left after processing the CBOR data item. (This generally means the representation is not processed at all except if some streaming processing has already happened.)

3. Usage Examples

3.1. Observing Resources

When a client registers to observe a resource [RFC7641] for which no representation is available yet, the server may send one or more 2.05 (Content) notifications before sending the first actual 2.05 (Content) or 2.03 (Valid) notification. The possible resulting sequence of notifications is shown in Figure 1.

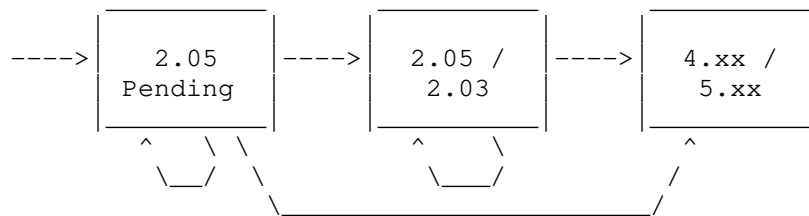


Figure 2: Sequence of Notifications:

The specification of the Observe option requires that all notifications carry the same Content-Format. The application/multipart-core media type can be used to provide that Content-Format: e.g., carrying an empty list of representations in the case marked as "Pending" in Figure 2, and carrying a single representation specified as the target content-format in the case in the middle of the figure.

4. Implementation hints

This section describes the serialization for readers that may be new to CBOR. It does not contain any new information.

An application/multipart-core representation carrying no representations is represented by an empty CBOR array, which is serialized as a single byte with the value 0x80.

An application/multipart-core representation carrying a single representation is represented by a two-element CBOR array, which is serialized as 0x82 followed by the two elements. The first element is an unsigned integer for the Content-Format value, which is

represented as described in Table 1. The second element is the object as a byte string, which is represented as a length as described in Table 2 followed by the bytes of the object.

| Serialization | Value |
|----------------|------------|
| 0x00..0x17 | 0..23 |
| 0x18 0xnn | 24..255 |
| 0x19 0xnn 0xnn | 256..65535 |

Table 1: Serialization of content-format

| Serialization | Length |
|-----------------------------|-------------------|
| 0x40..0x57 | 0..23 |
| 0x58 0xnn | 24..255 |
| 0x59 0xnn 0xnn | 256..65535 |
| 0x5a 0xnn 0xnn 0xnn 0xnn | 65536..4294967295 |
| 0x5b 0xnn .. 0xnn (8 bytes) | 4294967296.. |

Table 2: Serialization of object length

For example, a single text/plain object (content-format 0) of value "Hello World" (11 characters) would be serialized as

```
0x82 0x00 0x4b H e l l o 0x20 w o r l d
```

In effect, the serialization for a single object is done by prefixing the object with information that there is one object (here: 0x82), about its content-format (here: 0x00) and its length (here: 0x4b).

For more than one representation included in an application/multipart-core representation, the head of the CBOR array is adjusted (0x84 for two representations, 0x86 for three, ...) and the sequences of content-format and embedded representations follow.

5. IANA Considerations

5.1. Registration of media type application/multipart-core

IANA is requested to register the following media type [RFC6838]:

Type name: application

Subtype name: multipart-core

Required parameters: N/A

Optional parameters: N/A

Encoding considerations: binary

Security considerations: See the Security Considerations Section of RFCthis

Interoperability considerations: N/A

Published specification: RFCthis

Applications that use this media type: Applications that need to combine representations of zero or more different media types into one, e.g., EST-CoAP [I-D.ietf-ace-coap-est]

Fragment identifier considerations: The syntax and semantics of fragment identifiers specified for "application/multipart-core" is as specified for "application/cbor". (At publication of this document, there is no fragment identification syntax defined for "application/cbor".)

Additional information:

Deprecated alias names for this type: N/A

Magic number(s): N/A

File extension(s): N/A

Macintosh file type code(s): N/A

Person & email address to contact for further information: iesg&ietf.org

Intended usage: COMMON

Restrictions on usage: N/A

Author: CoRE WG

Change controller: IESG

Provisional registration? (standards tree only): no

5.2. Registration of a Content-Format identifier for application/multipart-core

IANA is requested to register the following Content-Format to the "CoAP Content-Formats" subregistry, within the "Constrained RESTful Environments (CoRE) Parameters" registry, from the Expert Review space (0..255):

| Media Type | Encoding | ID | Reference |
|----------------------------|----------|------|-----------|
| application/multipart-core | -- | TBD1 | RFCthis |

6. Security Considerations

The security considerations of [RFC7049] apply. In particular, resource exhaustion attacks may employ large values for the byte string size fields, or deeply nested structures of recursively embedded application/multipart-core representations.

7. References

7.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://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, <<https://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, <<https://www.rfc-editor.org/info/rfc7252>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.

7.2. Informative References

- [I-D.ietf-ace-coap-est]
Stok, P., Kampanakis, P., Richardson, M., and S. Raza,
"EST over secure CoAP (EST-coaps)", draft-ietf-ace-coap-
est-10 (work in progress), March 2019.
- [I-D.ietf-cbor-cddl]
Birkholz, H., Vigano, C., and C. Bormann, "Concise data
definition language (CDDL): a notational convention to
express CBOR and JSON data structures", draft-ietf-cbor-
cddl-07 (work in progress), February 2019.
- [RFC6838] Freed, N., Klensin, J., and T. Hansen, "Media Type
Specifications and Registration Procedures", BCP 13,
RFC 6838, DOI 10.17487/RFC6838, January 2013,
<<https://www.rfc-editor.org/info/rfc6838>>.
- [RFC7641] Hartke, K., "Observing Resources in the Constrained
Application Protocol (CoAP)", RFC 7641,
DOI 10.17487/RFC7641, September 2015,
<<https://www.rfc-editor.org/info/rfc7641>>.

Acknowledgements

Most of the text in this draft is from earlier contributions by two of the authors, Thomas Fossati and Klaus Hartke. The re-mix in this document is based on the requirements in [I-D.ietf-ace-coap-est], based on discussions with Michael Richardson, Panos Kampanis and Peter van der Stok.

Authors' Addresses

Thomas Fossati
ARM

Email: thomas.fossati@arm.com

Klaus Hartke
Ericsson
Torshamnsgatan 23
Stockholm SE-16483
Sweden

Email: klaus.hartke@ericsson.com

Carsten Bormann
Universitaet Bremen TZI
Postfach 330440
Bremen D-28359
Germany

Phone: +49-421-218-63921
Email: cabo@tzi.org

CoRE Working Group
Internet-Draft
Updates: 7252 (if approved)
Intended status: Standards Track
Expires: September 7, 2019

G. Selander
J. Mattsson
F. Palombini
Ericsson AB
L. Seitz
RISE SICS
March 06, 2019

Object Security for Constrained RESTful Environments (OSCORE)
draft-ietf-core-object-security-16

Abstract

This document defines Object Security for Constrained RESTful Environments (OSCORE), a method for application-layer protection of the Constrained Application Protocol (CoAP), using CBOR Object Signing and Encryption (COSE). OSCORE provides end-to-end protection between endpoints communicating using CoAP or CoAP-mappable HTTP. OSCORE is designed for constrained nodes and networks supporting a range of proxy operations, including translation between different transport protocols.

Although being an optional functionality of CoAP, OSCORE alters CoAP options processing and IANA registration. Therefore, this document updates [RFC7252].

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 7, 2019.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

| | |
|--|----|
| 1. Introduction | 4 |
| 1.1. Terminology | 6 |
| 2. The OSCORE Option | 7 |
| 3. The Security Context | 7 |
| 3.1. Security Context Definition | 8 |
| 3.2. Establishment of Security Context Parameters | 10 |
| 3.3. Requirements on the Security Context Parameters | 12 |
| 4. Protected Message Fields | 13 |
| 4.1. CoAP Options | 14 |
| 4.2. CoAP Header Fields and Payload | 23 |
| 4.3. Signaling Messages | 23 |
| 5. The COSE Object | 24 |
| 5.1. ID Context and 'kid context' | 25 |
| 5.2. AEAD Nonce | 26 |
| 5.3. Plaintext | 27 |
| 5.4. Additional Authenticated Data | 28 |
| 6. OSCORE Header Compression | 29 |
| 6.1. Encoding of the OSCORE Option Value | 30 |
| 6.2. Encoding of the OSCORE Payload | 31 |
| 6.3. Examples of Compressed COSE Objects | 31 |
| 7. Message Binding, Sequence Numbers, Freshness, and Replay Protection | 34 |
| 7.1. Message Binding | 34 |
| 7.2. Sequence Numbers | 34 |
| 7.3. Freshness | 34 |
| 7.4. Replay Protection | 35 |
| 7.5. Losing Part of the Context State | 36 |
| 8. Processing | 37 |
| 8.1. Protecting the Request | 37 |
| 8.2. Verifying the Request | 37 |
| 8.3. Protecting the Response | 39 |

| | |
|---|----|
| 8.4. Verifying the Response | 40 |
| 9. Web Linking | 42 |
| 10. CoAP-to-CoAP Forwarding Proxy | 42 |
| 11. HTTP Operations | 43 |
| 11.1. The HTTP OSCORE Header Field | 43 |
| 11.2. CoAP-to-HTTP Mapping | 44 |
| 11.3. HTTP-to-CoAP Mapping | 45 |
| 11.4. HTTP Endpoints | 45 |
| 11.5. Example: HTTP Client and CoAP Server | 46 |
| 11.6. Example: CoAP Client and HTTP Server | 47 |
| 12. Security Considerations | 48 |
| 12.1. End-to-end Protection | 48 |
| 12.2. Security Context Establishment | 49 |
| 12.3. Master Secret | 49 |
| 12.4. Replay Protection | 50 |
| 12.5. Client Aliveness | 50 |
| 12.6. Cryptographic Considerations | 50 |
| 12.7. Message Segmentation | 51 |
| 12.8. Privacy Considerations | 51 |
| 13. IANA Considerations | 52 |
| 13.1. COSE Header Parameters Registry | 52 |
| 13.2. CoAP Option Numbers Registry | 53 |
| 13.3. CoAP Signaling Option Numbers Registry | 54 |
| 13.4. Header Field Registrations | 54 |
| 13.5. Media Type Registrations | 54 |
| 13.6. CoAP Content-Formats Registry | 56 |
| 13.7. OSCORE Flag Bits Registry | 56 |
| 13.8. Expert Review Instructions | 57 |
| 14. References | 58 |
| 14.1. Normative References | 58 |
| 14.2. Informative References | 59 |
| Appendix A. Scenario Examples | 62 |
| A.1. Secure Access to Sensor | 62 |
| A.2. Secure Subscribe to Sensor | 63 |
| Appendix B. Deployment Examples | 64 |
| B.1. Security Context Derived Once | 64 |
| B.2. Security Context Derived Multiple Times | 66 |
| Appendix C. Test Vectors | 71 |
| C.1. Test Vector 1: Key Derivation with Master Salt | 72 |
| C.2. Test Vector 2: Key Derivation without Master Salt | 73 |
| C.3. Test Vector 3: Key Derivation with ID Context | 75 |
| C.4. Test Vector 4: OSCORE Request, Client | 76 |
| C.5. Test Vector 5: OSCORE Request, Client | 77 |
| C.6. Test Vector 6: OSCORE Request, Client | 79 |
| C.7. Test Vector 7: OSCORE Response, Server | 80 |
| C.8. Test Vector 8: OSCORE Response with Partial IV, Server | 81 |
| Appendix D. Overview of Security Properties | 82 |
| D.1. Threat Model | 82 |

| | |
|--|----|
| D.2. Supporting Proxy Operations | 83 |
| D.3. Protected Message Fields | 84 |
| D.4. Uniqueness of (key, nonce) | 85 |
| D.5. Unprotected Message Fields | 86 |
| Appendix E. CDDL Summary | 89 |
| Acknowledgments | 90 |
| Authors' Addresses | 90 |

1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] is a web transfer protocol, designed for constrained nodes and networks [RFC7228], and may be mapped from HTTP [RFC8075]. CoAP specifies the use of proxies for scalability and efficiency and references DTLS [RFC6347] for security. CoAP-to-CoAP, HTTP-to-CoAP, and CoAP-to-HTTP proxies require DTLS or TLS [RFC8446] to be terminated at the proxy. The proxy therefore not only has access to the data required for performing the intended proxy functionality, but is also able to eavesdrop on, or manipulate any part of, the message payload and metadata in transit between the endpoints. The proxy can also inject, delete, or reorder packets since they are no longer protected by (D)TLS.

This document defines the Object Security for Constrained RESTful Environments (OSCORE) security protocol, protecting CoAP and CoAP-mappable HTTP requests and responses end-to-end across intermediary nodes such as CoAP forward proxies and cross-protocol translators including HTTP-to-CoAP proxies [RFC8075]. In addition to the core CoAP features defined in [RFC7252], OSCORE supports the Observe [RFC7641], Block-wise [RFC7959], and No-Response [RFC7967] options, as well as the PATCH and FETCH methods [RFC8132]. An analysis of end-to-end security for CoAP messages through some types of intermediary nodes is performed in [I-D.hartke-core-e2e-security-reqs]. OSCORE essentially protects the RESTful interactions; the request method, the requested resource, the message payload, etc. (see Section 4). OSCORE protects neither the CoAP Messaging Layer nor the CoAP Token which may change between the endpoints, and those are therefore processed as defined in [RFC7252]. Additionally, since the message formats for CoAP over unreliable transport [RFC7252] and for CoAP over reliable transport [RFC8323] differ only in terms of CoAP Messaging Layer, OSCORE can be applied to both unreliable and reliable transports (see Figure 1).

OSCORE works in very constrained nodes and networks, thanks to its small message size and the restricted code and memory requirements in addition to what is required by CoAP. Examples of the use of OSCORE are given in Appendix A. OSCORE may be used over any underlying layer, such as e.g. UDP or TCP, and with non-IP transports (e.g.,

[I-D.bormann-6lo-coap-802-15-4e]). OSCORE may also be used in different ways with HTTP. OSCORE messages may be transported in HTTP, and OSCORE may also be used to protect CoAP-mappable HTTP messages, as described below.

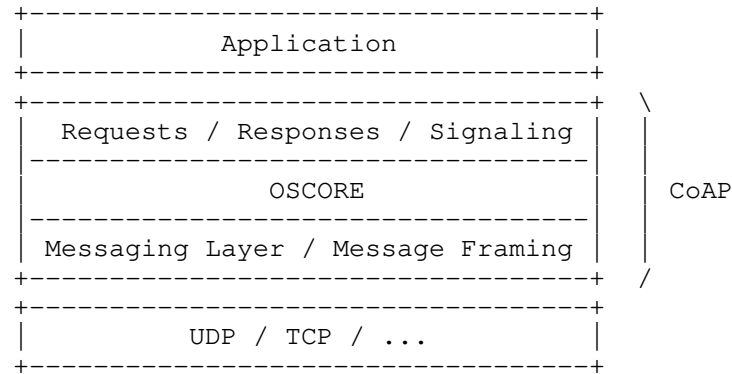


Figure 1: Abstract Layering of CoAP with OSCORE

OSCORE is designed to protect as much information as possible while still allowing CoAP proxy operations (Section 10). It works with existing CoAP-to-CoAP forward proxies [RFC7252], but an OSCORE-aware proxy will be more efficient. HTTP-to-CoAP proxies [RFC8075] and CoAP-to-HTTP proxies can also be used with OSCORE, as specified in Section 11. OSCORE may be used together with TLS or DTLS over one or more hops in the end-to-end path, e.g. transported with HTTPS in one hop and with plain CoAP in another hop. The use of OSCORE does not affect the URI scheme and OSCORE can therefore be used with any URI scheme defined for CoAP or HTTP. The application decides the conditions for which OSCORE is required.

OSCORE uses pre-shared keys which may have been established out-of-band or with a key establishment protocol (see Section 3.2). The technical solution builds on CBOR Object Signing and Encryption (COSE) [RFC8152], providing end-to-end encryption, integrity, replay protection, and binding of response to request. A compressed version of COSE is used, as specified in Section 6. The use of OSCORE is signaled in CoAP with a new option (Section 2), and in HTTP with a new header field (Section 11.1) and content type (Section 13.5). The solution transforms a CoAP/HTTP message into an "OSCORE message" before sending, and vice versa after receiving. The OSCORE message is a CoAP/HTTP message related to the original message in the following way: the original CoAP/HTTP message is translated to CoAP (if not already in CoAP) and protected in a COSE object. The encrypted message fields of this COSE object are transported in the CoAP payload/HTTP body of the OSCORE message, and the OSCORE option/

header field is included in the message. A sketch of an exchange of OSCORE messages, in the case of the original message being CoAP, is provided in Figure 2. The use of OSCORE with HTTP is detailed in Section 11.

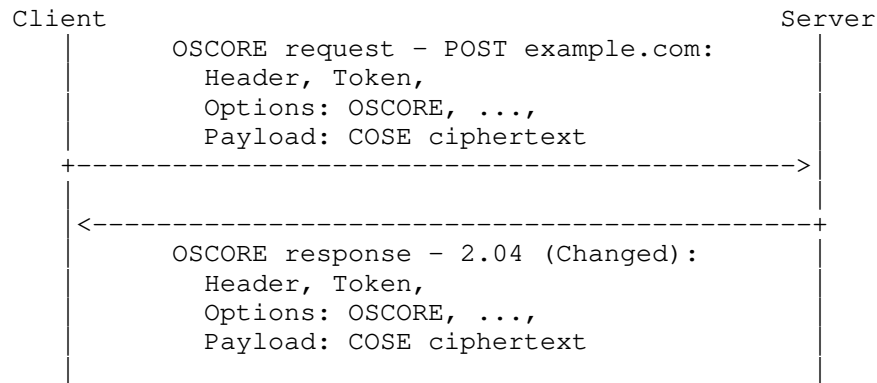


Figure 2: Sketch of CoAP with OSCORE

An implementation supporting this specification MAY implement only the client part, MAY implement only the server part, or MAY implement only one of the proxy parts.

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 BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

Readers are expected to be familiar with the terms and concepts described in CoAP [RFC7252], Observe [RFC7641], Block-wise [RFC7959], COSE [RFC8152], CBOR [RFC7049], CDDL [I-D.ietf-cbor-cddl] as summarized in Appendix E, and constrained environments [RFC7228].

The term "hop" is used to denote a particular leg in the end-to-end path. The concept "hop-by-hop" (as in "hop-by-hop encryption" or "hop-by-hop fragmentation") opposed to "end-to-end", is used in this document to indicate that the messages are processed accordingly in the intermediaries, rather than just forwarded to the next node.

The term "stop processing" is used throughout the document to denote that the message is not passed up to the CoAP Request/Response Layer (see Figure 1).

The terms Common/Sender/Recipient Context, Master Secret/Salt, Sender ID/Key, Recipient ID/Key, ID Context, and Common IV are defined in Section 3.1.

2. The OSCORE Option

The OSCORE option defined in this section (see Figure 3, which extends Table 4: Options of [RFC7252]) indicates that the CoAP message is an OSCORE message and that it contains a compressed COSE object (see Sections 5 and 6). The OSCORE option is critical, safe to forward, part of the cache key, and not repeatable.

| No. | C | U | N | R | Name | Format | Length | Default |
|------|---|---|---|---|--------|--------|--------|---------|
| TBD1 | x | | | | OSCORE | (*) | 0-255 | (none) |

C = Critical, U = Unsafe, N = NoCacheKey, R = Repeatable
 (*) See below.

Figure 3: The OSCORE Option

The OSCORE option includes the OSCORE flag bits (Section 6), the Sender Sequence Number, the Sender ID, and the ID Context when these fields are present (Section 3). The detailed format and length is specified in Section 6. If the OSCORE flag bits are all zero (0x00) the Option value SHALL be empty (Option Length = 0). An endpoint receiving a CoAP message without payload, that also contains an OSCORE option SHALL treat it as malformed and reject it.

A successful response to a request with the OSCORE option SHALL contain the OSCORE option. Whether error responses contain the OSCORE option depends on the error type (see Section 8).

For CoAP proxy operations, see Section 10.

3. The Security Context

OSCORE requires that client and server establish a shared security context used to process the COSE objects. OSCORE uses COSE with an Authenticated Encryption with Additional Data (AEAD, [RFC5116]) algorithm for protecting message data between a client and a server. In this section, we define the security context and how it is derived in client and server based on a shared secret and a key derivation function.

3.1. Security Context Definition

The security context is the set of information elements necessary to carry out the cryptographic operations in OSCORE. For each endpoint, the security context is composed of a "Common Context", a "Sender Context", and a "Recipient Context".

The endpoints protect messages to send using the Sender Context and verify messages received using the Recipient Context, both contexts being derived from the Common Context and other data. Clients and servers need to be able to retrieve the correct security context to use.

An endpoint uses its Sender ID (SID) to derive its Sender Context, and the other endpoint uses the same ID, now called Recipient ID (RID), to derive its Recipient Context. In communication between two endpoints, the Sender Context of one endpoint matches the Recipient Context of the other endpoint, and vice versa. Thus, the two security contexts identified by the same IDs in the two endpoints are not the same, but they are partly mirrored. Retrieval and use of the security context are shown in Figure 4.

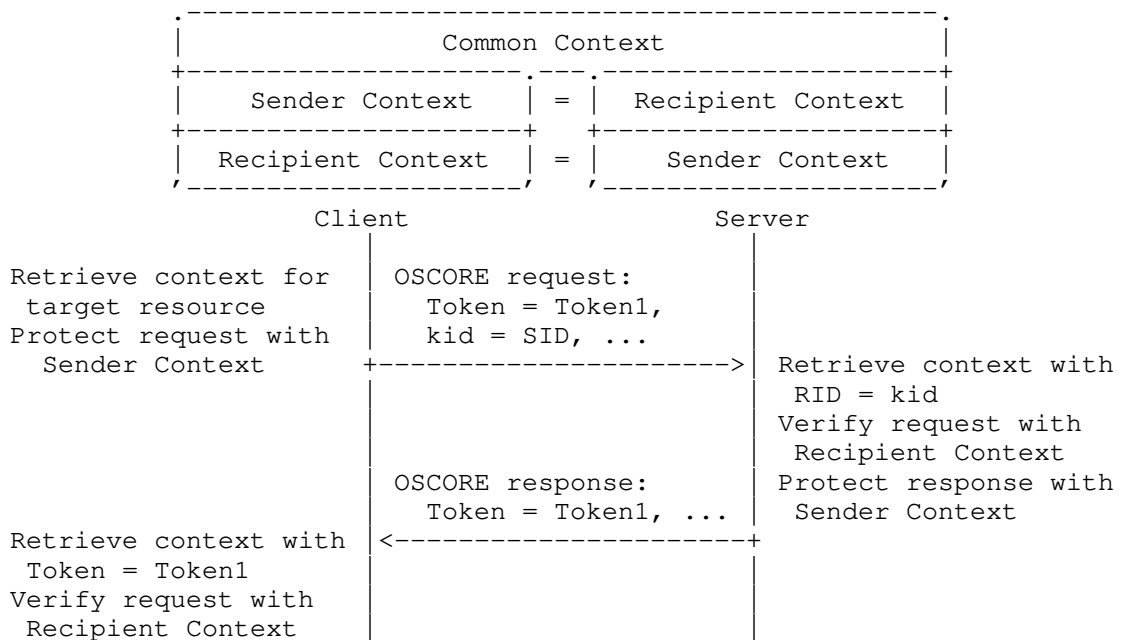


Figure 4: Retrieval and Use of the Security Context

The Common Context contains the following parameters:

- o AEAD Algorithm. The COSE AEAD algorithm to use for encryption.
- o HKDF Algorithm. An HMAC-based key derivation function HKDF [RFC5869] used to derive Sender Key, Recipient Key, and Common IV.
- o Master Secret. Variable length, random byte string (see Section 12.3) used to derive AEAD keys and Common IV.
- o Master Salt. Optional variable length byte string containing the salt used to derive AEAD keys and Common IV.
- o ID Context. Optional variable length byte string providing additional information to identify the Common Context and to derive AEAD keys and Common IV. The use of ID Context is described in Section 5.1.
- o Common IV. Byte string derived from Master Secret, Master Salt, and ID Context. Used to generate the AEAD Nonce (see Section 5.2). Same length as the nonce of the AEAD Algorithm.

The Sender Context contains the following parameters:

- o Sender ID. Byte string used to identify the Sender Context, to derive AEAD keys and Common IV, and to assure unique AEAD nonces. Maximum length is determined by the AEAD Algorithm.
- o Sender Key. Byte string containing the symmetric AEAD key to protect messages to send. Derived from Common Context and Sender ID. Length is determined by the AEAD Algorithm.
- o Sender Sequence Number. Non-negative integer used by the sender to enumerate requests and certain responses, e.g. Observe notifications. Used as 'Partial IV' [RFC8152] to generate unique AEAD nonces. Maximum value is determined by the AEAD Algorithm. Initialization is described in Section 3.2.2.

The Recipient Context contains the following parameters:

- o Recipient ID. Byte string used to identify the Recipient Context, to derive AEAD keys and Common IV, and to assure unique AEAD nonces. Maximum length is determined by the AEAD Algorithm.
- o Recipient Key. Byte string containing the symmetric AEAD key to verify messages received. Derived from Common Context and Recipient ID. Length is determined by the AEAD Algorithm.

- o Replay Window (Server only). The replay window to verify requests received. Replay protection is described in Section 7.4 and Section 3.2.2.

All parameters except Sender Sequence Number and Replay Window are immutable once the security context is established. An endpoint may free up memory by not storing the Common IV, Sender Key, and Recipient Key, deriving them when needed. Alternatively, an endpoint may free up memory by not storing the Master Secret and Master Salt after the other parameters have been derived.

Endpoints MAY operate as both client and server and use the same security context for those roles. Independent of being client or server, the endpoint protects messages to send using its Sender Context, and verifies messages received using its Recipient Context. The endpoints MUST NOT change the Sender/Recipient ID when changing roles. In other words, changing the roles does not change the set of AEAD keys to be used.

3.2. Establishment of Security Context Parameters

Each endpoint derives the parameters in the security context from a small set of input parameters. The following input parameters SHALL be pre-established:

- o Master Secret
- o Sender ID
- o Recipient ID

The following input parameters MAY be pre-established. In case any of these parameters is not pre-established, the default value indicated below is used:

- o AEAD Algorithm
 - * Default is AES-CCM-16-64-128 (COSE algorithm encoding: 10)
- o Master Salt
 - * Default is the empty byte string
- o HKDF Algorithm
 - * Default is HKDF SHA-256
- o Replay Window

- * Default is DTLS-type replay protection with a window size of 32 [RFC6347]

All input parameters need to be known to and agreed on by both endpoints, but the replay window may be different in the two endpoints. The way the input parameters are pre-established, is application specific. Considerations of security context establishment are given in Section 12.2 and examples of deploying OSCORE in Appendix B.

3.2.1. Derivation of Sender Key, Recipient Key, and Common IV

The HKDF MUST be one of the HMAC-based HKDF [RFC5869] algorithms defined for COSE [RFC8152]. HKDF SHA-256 is mandatory to implement. The security context parameters Sender Key, Recipient Key, and Common IV SHALL be derived from the input parameters using the HKDF, which consists of the composition of the HKDF-Extract and HKDF-Expand steps [RFC5869]:

```
output parameter = HKDF(salt, IKM, info, L)
```

where:

- o salt is the Master Salt as defined above
- o IKM is the Master Secret as defined above
- o info is the serialization of a CBOR array consisting of (the notation follows Appendix E):

```
info = [  
  id : bstr,  
  id_context : bstr / nil,  
  alg_aead : int / tstr,  
  type : tstr,  
  L : uint,  
]
```

where:

- o id is the Sender ID or Recipient ID when deriving Sender Key and Recipient Key, respectively, and the empty byte string when deriving the Common IV.
- o id_context is the ID Context, or nil if ID Context is not provided.
- o alg_aead is the AEAD Algorithm, encoded as defined in [RFC8152].

- o type is "Key" or "IV". The label is an ASCII string, and does not include a trailing NUL byte.
- o L is the size of the key/nonce for the AEAD algorithm used, in bytes.

For example, if the algorithm AES-CCM-16-64-128 (see Section 10.2 in [RFC8152]) is used, the integer value for alg_aead is 10, the value for L is 16 for keys and 13 for the Common IV. Assuming use of the default algorithms HKDF SHA-256 and AES-CCM-16-64-128, the extract phase of HKDF produces a pseudorandom key (PRK) as follows:

PRK = HMAC-SHA-256(Master Salt, Master Secret)

and as L is smaller than the hash function output size, the expand phase of HKDF consists of a single HMAC invocation, and the Sender Key, Recipient Key, and Common IV are therefore the first 16 or 13 bytes of

output parameter = HMAC-SHA-256(PRK, info || 0x01)

where different info are used for each derived parameter and where || denotes byte string concatenation.

Note that [RFC5869] specifies that if the salt is not provided, it is set to a string of zeros. For implementation purposes, not providing the salt is the same as setting the salt to the empty byte string. OSCORE sets the salt default value to empty byte string, which is converted to a string of zeroes (see Section 2.2 of [RFC5869]).

3.2.2. Initial Sequence Numbers and Replay Window

The Sender Sequence Number is initialized to 0.

The supported types of replay protection and replay window length is application specific and depends on how OSCORE is transported, see Section 7.4. The default is DTLS-type replay protection with a window size of 32 initiated as described in Section 4.1.2.6 of [RFC6347].

3.3. Requirements on the Security Context Parameters

To ensure unique Sender Keys, the quartet (Master Secret, Master Salt, ID Context, Sender ID) MUST be unique, i.e. the pair (ID Context, Sender ID) SHALL be unique in the set of all security contexts using the same Master Secret and Master Salt. This means that Sender ID SHALL be unique in the set of all security contexts

using the same Master Secret, Master Salt, and ID Context; such a requirement guarantees unique (key, nonce) pairs for the AEAD.

Different methods can be used to assign Sender IDs: a protocol that allows the parties to negotiate locally unique identifiers, a trusted third party (e.g., [I-D.ietf-ace-oauth-authz]), or the identifiers can be assigned out-of-band. The Sender IDs can be very short (note that the empty string is a legitimate value). The maximum length of Sender ID in bytes equals the length of AEAD nonce minus 6, see Section 5.2. For AES-CCM-16-64-128 the maximum length of Sender ID is 7 bytes.

To simplify retrieval of the right Recipient Context, the Recipient ID SHOULD be unique in the sets of all Recipient Contexts used by an endpoint. If an endpoint has the same Recipient ID with different Recipient Contexts, i.e. the Recipient Contexts are derived from different Common Contexts, then the endpoint may need to try multiple times before verifying the right security context associated to the Recipient ID.

The ID Context is used to distinguish between security contexts. The methods used for assigning Sender ID can also be used for assigning the ID Context. Additionally, the ID Context can be used to introduce randomness into new Sender and Recipient Contexts (see Appendix B.2). ID Context can be arbitrarily long.

4. Protected Message Fields

OSCORE transforms a CoAP message (which may have been generated from an HTTP message) into an OSCORE message, and vice versa. OSCORE protects as much of the original message as possible while still allowing certain proxy operations (see Sections 10 and 11). This section defines how OSCORE protects the message fields and transfers them end-to-end between client and server (in any direction).

The remainder of this section and later sections focus on the behavior in terms of CoAP messages. If HTTP is used for a particular hop in the end-to-end path, then this section applies to the conceptual CoAP message that is mappable to/from the original HTTP message as discussed in Section 11. That is, an HTTP message is conceptually transformed to a CoAP message and then to an OSCORE message, and similarly in the reverse direction. An actual implementation might translate directly from HTTP to OSCORE without the intervening CoAP representation.

Protection of Signaling messages (Section 5 of [RFC8323]) is specified in Section 4.3. The other parts of this section target Request/Response messages.

Message fields of the CoAP message may be protected end-to-end between CoAP client and CoAP server in different ways:

- o Class E: encrypted and integrity protected,
- o Class I: integrity protected only, or
- o Class U: unprotected.

The sending endpoint SHALL transfer Class E message fields in the ciphertext of the COSE object in the OSCORE message. The sending endpoint SHALL include Class I message fields in the Additional Authenticated Data (AAD) of the AEAD algorithm, allowing the receiving endpoint to detect if the value has changed in transfer. Class U message fields SHALL NOT be protected in transfer. Class I and Class U message field values are transferred in the header or options part of the OSCORE message, which is visible to proxies.

Message fields not visible to proxies, i.e., transported in the ciphertext of the COSE object, are called "Inner" (Class E). Message fields transferred in the header or options part of the OSCORE message, which is visible to proxies, are called "Outer" (Class I or U). There are currently no Class I options defined.

An OSCORE message may contain both an Inner and an Outer instance of a certain CoAP message field. Inner message fields are intended for the receiving endpoint, whereas Outer message fields are used to enable proxy operations.

4.1. CoAP Options

A summary of how options are protected is shown in Figure 5. Note that some options may have both Inner and Outer message fields which are protected accordingly. Certain options require special processing as is described in Section 4.1.3.

Options that are unknown or for which OSCORE processing is not defined SHALL be processed as class E (and no special processing). Specifications of new CoAP options SHOULD define how they are processed with OSCORE. A new COAP option SHOULD be of class E unless it requires proxy processing. If a new CoAP option is of class U, the potential issues with the option being unprotected SHOULD be documented (see Appendix D.5).

4.1.1.1. Inner Options

Inner option message fields (class E) are used to communicate directly with the other endpoint.

The sending endpoint SHALL write the Inner option message fields present in the original CoAP message into the plaintext of the COSE object (Section 5.3), and then remove the Inner option message fields from the OSCORE message.

The processing of Inner option message fields by the receiving endpoint is specified in Sections 8.2 and 8.4.

| No. | Name | E | U |
|------|----------------|---|---|
| 1 | If-Match | x | |
| 3 | Uri-Host | | x |
| 4 | ETag | x | |
| 5 | If-None-Match | x | |
| 6 | Observe | x | x |
| 7 | Uri-Port | | x |
| 8 | Location-Path | x | |
| TBD1 | OSCORE | | x |
| 11 | Uri-Path | x | |
| 12 | Content-Format | x | |
| 14 | Max-Age | x | x |
| 15 | Uri-Query | x | |
| 17 | Accept | x | |
| 20 | Location-Query | x | |
| 23 | Block2 | x | x |
| 27 | Block1 | x | x |
| 28 | Size2 | x | x |
| 35 | Proxy-Uri | | x |
| 39 | Proxy-Scheme | | x |
| 60 | Size1 | x | x |
| 258 | No-Response | x | x |

E = Encrypt and Integrity Protect (Inner)
 U = Unprotected (Outer)

Figure 5: Protection of CoAP Options

4.1.2. Outer Options

Outer option message fields (Class U or I) are used to support proxy operations, see Appendix D.2.

The sending endpoint SHALL include the Outer option message field present in the original message in the options part of the OSCORE message. All Outer option message fields, including the OSCORE option, SHALL be encoded as described in Section 3.1 of [RFC7252], where the delta is the difference to the previously included instance of Outer option message field.

The processing of Outer options by the receiving endpoint is specified in Sections 8.2 and 8.4.

A procedure for integrity-protection-only of Class I option message fields is specified in Section 5.4. Specifications that introduce repeatable Class I options MUST specify that proxies MUST NOT change the order of the instances of such an option in the CoAP message.

Note: There are currently no Class I option message fields defined.

4.1.3. Special Options

Some options require special processing as specified in this section.

4.1.3.1. Max-Age

An Inner Max-Age message field is used to indicate the maximum time a response may be cached by the client (as defined in [RFC7252]), end-to-end from the server to the client, taking into account that the option is not accessible to proxies. The Inner Max-Age SHALL be processed by OSCORE as a normal Inner option, specified in Section 4.1.1.

An Outer Max-Age message field is used to avoid unnecessary caching of error responses caused by OSCORE processing at OSCORE-unaware intermediary nodes. A server MAY set a Class U Max-Age message field with value zero to such error responses, described in Sections 7.4, 8.2, and 8.4, since these error responses are cacheable, but subsequent OSCORE requests would never create a hit in the intermediary caching it. Setting the Outer Max-Age to zero relieves the intermediary from uselessly caching responses. Successful OSCORE responses do not need to include an Outer Max-Age option since the responses appear to the OSCORE-unaware intermediary as 2.04 (Changed) responses, which are non-cacheable (see Section 4.2).

The Outer Max-Age message field is processed according to Section 4.1.2.

4.1.3.2. Uri-Host and Uri-Port

When the Uri-Host and Uri-Port are set to their default values (see Section 5.10.1 [RFC7252]), they are omitted from the message (Section 5.4.4 of [RFC7252]), which is favorable both for overhead and privacy.

In order to support forward proxy operations, Proxy-Scheme, Uri-Host, and Uri-Port need to be Class U. For the use of Proxy-Uri, see Section 4.1.3.3.

Manipulation of unprotected message fields (including Uri-Host, Uri-Port, destination IP/port or request scheme) MUST NOT lead to an OSCORE message becoming verified by an unintended server. Different servers SHALL have different security contexts.

4.1.3.3. Proxy-Uri

When Proxy-Uri is present, the client SHALL first decompose the Proxy-Uri value of the original CoAP message into the Proxy-Scheme, Uri-Host, Uri-Port, Uri-Path, and Uri-Query options according to Section 6.4 of [RFC7252].

Uri-Path and Uri-Query are class E options and SHALL be protected and processed as Inner options (Section 4.1.1).

The Proxy-Uri option of the OSCORE message SHALL be set to the composition of Proxy-Scheme, Uri-Host, and Uri-Port options as specified in Section 6.5 of [RFC7252], and processed as an Outer option of Class U (Section 4.1.2).

Note that replacing the Proxy-Uri value with the Proxy-Scheme and Uri-* options works by design for all CoAP URIs (see Section 6 of [RFC7252]). OSCORE-aware HTTP servers should not use the userinfo component of the HTTP URI (as defined in Section 3.2.1 of [RFC3986]), so that this type of replacement is possible in the presence of CoAP-to-HTTP proxies (see Section 11.2). In future specifications of cross-protocol proxying behavior using different URI structures, it is expected that the authors will create Uri-* options that allow decomposing the Proxy-Uri, and specifying the OSCORE processing.

An example of how Proxy-Uri is processed is given here. Assume that the original CoAP message contains:

```
o Proxy-Uri = "coap://example.com/resource?q=1"
```

During OSCORE processing, Proxy-Uri is split into:

- o Proxy-Scheme = "coap"
- o Uri-Host = "example.com"
- o Uri-Port = "5683"
- o Uri-Path = "resource"
- o Uri-Query = "q=1"

Uri-Path and Uri-Query follow the processing defined in Section 4.1.1, and are thus encrypted and transported in the COSE object:

- o Uri-Path = "resource"
- o Uri-Query = "q=1"

The remaining options are composed into the Proxy-Uri included in the options part of the OSCORE message, which has value:

- o Proxy-Uri = "coap://example.com"

See Sections 6.1 and 12.6 of [RFC7252] for more details.

4.1.3.4. The Block Options

Block-wise [RFC7959] is an optional feature. An implementation MAY support [RFC7252] and the OSCORE option without supporting block-wise transfers. The Block options (Block1, Block2, Size1, Size2), when Inner message fields, provide secure message segmentation such that each segment can be verified. The Block options, when Outer message fields, enables hop-by-hop fragmentation of the OSCORE message. Inner and Outer block processing may have different performance properties depending on the underlying transport. The end-to-end integrity of the message can be verified both in case of Inner and Outer Block-wise transfers provided all blocks are received.

4.1.3.4.1. Inner Block Options

The sending CoAP endpoint MAY fragment a CoAP message as defined in [RFC7959] before the message is processed by OSCORE. In this case the Block options SHALL be processed by OSCORE as normal Inner options (Section 4.1.1). The receiving CoAP endpoint SHALL process the OSCORE message before processing Block-wise as defined in [RFC7959].

4.1.3.4.2. Outer Block Options

Proxies MAY fragment an OSCORE message using [RFC7959], by introducing Block option message fields that are Outer (Section 4.1.2). Note that the Outer Block options are neither encrypted nor integrity protected. As a consequence, a proxy can maliciously inject block fragments indefinitely, since the receiving endpoint needs to receive the last block (see [RFC7959]) to be able to compose the OSCORE message and verify its integrity. Therefore, applications supporting OSCORE and [RFC7959] MUST specify a security policy defining a maximum unfragmented message size (MAX_UNFRAGMENTED_SIZE) considering the maximum size of message which can be handled by the endpoints. Messages exceeding this size SHOULD be fragmented by the sending endpoint using Inner Block options (Section 4.1.3.4.1).

An endpoint receiving an OSCORE message with an Outer Block option SHALL first process this option according to [RFC7959], until all blocks of the OSCORE message have been received, or the cumulated message size of the blocks exceeds MAX_UNFRAGMENTED_SIZE. In the former case, the processing of the OSCORE message continues as defined in this document. In the latter case the message SHALL be discarded.

Because of encryption of Uri-Path and Uri-Query, messages to the same server may, from the point of view of a proxy, look like they also target the same resource. A proxy SHOULD mitigate a potential mix-up of blocks from concurrent requests to the same server, for example using the Request-Tag processing specified in Section 3.3.2 of [I-D.ietf-core-echo-request-tag].

4.1.3.5. Observe

Observe [RFC7641] is an optional feature. An implementation MAY support [RFC7252] and the OSCORE option without supporting [RFC7641], in which case the Observe related processing can be omitted.

The support for Observe [RFC7641] with OSCORE targets the requirements on forwarding of Section 2.2.1 of [I-D.hartke-core-e2e-security-reqs], i.e. that observations go through intermediary nodes, as illustrated in Figure 8 of [RFC7641].

Inner Observe SHALL be used to protect the value of the Observe option between the endpoints. Outer Observe SHALL be used to support forwarding by intermediary nodes.

The server SHALL include a new Partial IV (see Section 5) in responses (with or without the Observe option) to Observe

registrations, except for the first response where Partial IV MAY be omitted.

For cancellations, Section 3.6 of [RFC7641] specifies that all options MUST be identical to those in the registration request except for Observe and the set of ETag Options. For OSCORE messages, this matching is to be done to the options in the decrypted message.

[RFC7252] does not specify how the server should act upon receiving the same Token in different requests. When using OSCORE, the server SHOULD NOT remove an active observation just because it receives a request with the same Token.

Since POST with Observe is not defined, for messages with Observe, the Outer Code MUST be set to 0.05 (FETCH) for requests and to 2.05 (Content) for responses (see Section 4.2).

4.1.3.5.1. Registrations and Cancellations

The Inner and Outer Observe in the request MUST contain the Observe value of the original CoAP request; 0 (registration) or 1 (cancellation).

Every time a client issues a new Observe request, a new Partial IV MUST be used (see Section 5), and so the payload and OSCORE option are changed. The server uses the Partial IV of the new request as the 'request_piv' of all associated notifications (see Section 5.4).

Intermediaries are not assumed to have access to the OSCORE security context used by the endpoints, and thus cannot make requests or transform responses with the OSCORE option which verify at the receiving endpoint as coming from the other endpoint. This has the following consequences and limitations for Observe operations.

- o An intermediary node removing the Outer Observe 0 does not change the registration request to a request without Observe (see Section 2 of [RFC7641]). Instead other means for cancellation may be used as described in Section 3.6 of [RFC7641].
- o An intermediary node is not able to transform a normal response into an OSCORE protected Observe notification (see figure 7 of [RFC7641]) which verifies as coming from the server.
- o An intermediary node is not able to initiate an OSCORE protected Observe registration (Observe with value 0) which verifies as coming from the client. An OSCORE-aware intermediary SHALL NOT initiate registrations of observations (see Section 10). If an OSCORE-unaware proxy re-sends an old registration message from a

client this will trigger the replay protection mechanism in the server. To prevent this from resulting in the OSCORE-unaware proxy to cancel of the registration, a server MAY respond to a replayed registration request with a replay of a cached notification. Alternatively, the server MAY send a new notification.

- o An intermediary node is not able to initiate an OSCORE protected Observe cancellation (Observe with value 1) which verifies as coming from the client. An application MAY decide to allow intermediaries to cancel Observe registrations, e.g. to send Observe with value 1 (see Section 3.6 of [RFC7641]), but that can also be done with other methods, e.g. reusing the Token in a different request or sending a RST message. This is out of scope for this specification.

4.1.3.5.2. Notifications

If the server accepts an Observe registration, a Partial IV MUST be included in all notifications (both successful and error), except for the first one where Partial IV MAY be omitted. To protect against replay, the client SHALL maintain a Notification Number for each Observation it registers. The Notification Number is a non-negative integer containing the largest Partial IV of the received notifications for the associated Observe registration. Further details of replay protection of notifications are specified in Section 7.4.1.

For notifications, the Inner Observe value MUST be empty (see Section 3.2 of [RFC7252]). The Outer Observe in a notification is needed for intermediary nodes to allow multiple responses to one request, and may be set to the value of Observe in the original CoAP message. The client performs ordering of notifications and replay protection by comparing their Partial IVs and SHALL ignore the outer Observe value.

If the client receives a response to an Observe request without an Inner Observe option, then it verifies the response as a non-Observe response, as specified in Section 8.4. If the client receives a response to a non-Observe request with an Inner Observe option, then it stops processing the message, as specified in Section 8.4.

A client MUST consider the notification with the highest Partial IV as the freshest, regardless of the order of arrival. In order to support existing Observe implementations the OSCORE client implementation MAY set the Observe value to the three least significant bytes of the Partial IV. Implementations need to make

sure that the notification without Partial IV is considered the oldest.

4.1.3.6. No-Response

No-Response [RFC7967] is an optional feature used by the client to communicate its disinterest in certain classes of responses to a particular request. An implementation MAY support [RFC7252] and the OSCORE option without supporting [RFC7967].

If used, No-Response MUST be Inner. The Inner No-Response SHALL be processed by OSCORE as specified in Section 4.1.1. The Outer option SHOULD NOT be present. The server SHALL ignore the Outer No-Response option. The client MAY set the Outer No-Response value to 26 ('suppress all known codes') if the Inner value is set to 26. The client MUST be prepared to receive and discard 5.04 (Gateway Timeout) error messages from intermediaries potentially resulting from destination time out due to no response.

4.1.3.7. OSCORE

The OSCORE option is only defined to be present in OSCORE messages, as an indication that OSCORE processing have been performed. The content in the OSCORE option is neither encrypted nor integrity protected as a whole but some part of the content of this option is protected (see Section 5.4). Nested use of OSCORE is not supported: If OSCORE processing detects an OSCORE option in the original CoAP message, then processing SHALL be stopped.

| Field | E | U |
|------------------|---|---|
| Version (UDP) | | x |
| Type (UDP) | | x |
| Length (TCP) | | x |
| Token Length | | x |
| Code | x | |
| Message ID (UDP) | | x |
| Token | | x |
| Payload | x | |

E = Encrypt and Integrity Protect (Inner)
U = Unprotected (Outer)

Figure 6: Protection of CoAP Header Fields and Payload

4.2. CoAP Header Fields and Payload

A summary of how the CoAP header fields and payload are protected is shown in Figure 6, including fields specific to CoAP over UDP and CoAP over TCP (marked accordingly in the table).

Most CoAP Header fields (i.e. the message fields in the fixed 4-byte header) are required to be read and/or changed by CoAP proxies and thus cannot in general be protected end-to-end between the endpoints. As mentioned in Section 1, OSCORE protects the CoAP Request/Response Layer only, and not the Messaging Layer (Section 2 of [RFC7252]), so fields such as Type and Message ID are not protected with OSCORE.

The CoAP Header field Code is protected by OSCORE. Code SHALL be encrypted and integrity protected (Class E) to prevent an intermediary from eavesdropping on or manipulating the Code (e.g., changing from GET to DELETE).

The sending endpoint SHALL write the Code of the original CoAP message into the plaintext of the COSE object (see Section 5.3). After that, the sending endpoint writes an Outer Code to the OSCORE message. With one exception (see Section 4.1.3.5) the Outer Code SHALL be set to 0.02 (POST) for requests and to 2.04 (Changed) for responses. The receiving endpoint SHALL discard the Outer Code in the OSCORE message and write the Code of the COSE object plaintext (Section 5.3) into the decrypted CoAP message.

The other currently defined CoAP Header fields are Unprotected (Class U). The sending endpoint SHALL write all other header fields of the original message into the header of the OSCORE message. The receiving endpoint SHALL write the header fields from the received OSCORE message into the header of the decrypted CoAP message.

The CoAP Payload, if present in the original CoAP message, SHALL be encrypted and integrity protected and is thus an Inner message field. The sending endpoint writes the payload of the original CoAP message into the plaintext (Section 5.3) input to the COSE object. The receiving endpoint verifies and decrypts the COSE object, and recreates the payload of the original CoAP message.

4.3. Signaling Messages

Signaling messages (CoAP Code 7.00–7.31) were introduced to exchange information related to an underlying transport connection in the specific case of CoAP over reliable transports [RFC8323].

OSCORE MAY be used to protect Signaling if the endpoints for OSCORE coincide with the endpoints for the signaling message. If OSCORE is used to protect Signaling then:

- o To comply with [RFC8323], an initial empty CSM message SHALL be sent. The subsequent signaling message SHALL be protected.
- o Signaling messages SHALL be protected as CoAP Request messages, except in the case the Signaling message is a response to a previous Signaling message, in which case it SHALL be protected as a CoAP Response message. For example, 7.02 (Ping) is protected as a CoAP Request and 7.03 (Pong) as a CoAP response.
- o The Outer Code for Signaling messages SHALL be set to 0.02 (POST), unless it is a response to a previous Signaling message, in which case it SHALL be set to 2.04 (Changed).
- o All Signaling options, except the OSCORE option, SHALL be Inner (Class E).

NOTE: Option numbers for Signaling messages are specific to the CoAP Code (see Section 5.2 of [RFC8323]).

If OSCORE is not used to protect Signaling, Signaling messages SHALL be unaltered by OSCORE.

5. The COSE Object

This section defines how to use COSE [RFC8152] to wrap and protect data in the original message. OSCORE uses the untagged COSE_Encrypt0 structure with an Authenticated Encryption with Additional Data (AEAD) algorithm. The AEAD key lengths, AEAD nonce length, and maximum Sender Sequence Number are algorithm dependent.

The AEAD algorithm AES-CCM-16-64-128 defined in Section 10.2 of [RFC8152] is mandatory to implement. For AES-CCM-16-64-128 the length of Sender Key and Recipient Key is 128 bits, the length of AEAD nonce and Common IV is 13 bytes. The maximum Sender Sequence Number is specified in Section 12.

As specified in [RFC5116], plaintext denotes the data that is to be encrypted and integrity protected, and Additional Authenticated Data (AAD) denotes the data that is to be integrity protected only.

The COSE Object SHALL be a COSE_Encrypt0 object with fields defined as follows

- o The 'protected' field is empty.

- o The 'unprotected' field includes:
 - * The 'Partial IV' parameter. The value is set to the Sender Sequence Number. All leading bytes of value zero SHALL be removed when encoding the Partial IV, except in the case of Partial IV of value 0 which is encoded to the byte string 0x00. This parameter SHALL be present in requests. The Partial IV SHALL be present in responses to Observe registrations (see Section 4.1.3.5.1), otherwise the Partial IV will not typically be present in responses (for one exception, see Appendix B.1.2).
 - * The 'kid' parameter. The value is set to the Sender ID. This parameter SHALL be present in requests and will not typically be present in responses. An example where the Sender ID is included in a response is the extension of OSCORE to group communication [I-D.ietf-core-oscore-groupcomm].
 - * Optionally, a 'kid context' parameter (see Section 5.1). This parameter MAY be present in requests, and if so, MUST contain an ID Context (see Section 3.1). This parameter SHOULD NOT be present in responses: an example of how 'kid context' can be used in responses is given in Appendix B.2. If 'kid context' is present in the request, then the server SHALL use a security context with that ID Context when verifying the request.
- o The 'ciphertext' field is computed from the secret key (Sender Key or Recipient Key), AEAD nonce (see Section 5.2), plaintext (see Section 5.3), and the Additional Authenticated Data (AAD) (see Section 5.4) following Section 5.2 of [RFC8152].

The encryption process is described in Section 5.3 of [RFC8152].

5.1. ID Context and 'kid context'

For certain use cases, e.g. deployments where the same Sender ID is used with multiple contexts, it is possible (and sometimes necessary, see Section 3.3) for the client to use an ID Context to distinguish the security contexts (see Section 3.1). For example:

- o If the client has a unique identifier in some namespace then that identifier can be used as ID Context.
- o The ID Context may be used to add randomness into new Sender and Recipient Contexts, see Appendix B.2.

- o In case of group communication [I-D.ietf-core-oscore-groupcomm], a group identifier is used as ID Context to enable different security contexts for a server belonging to multiple groups.

The Sender ID and ID Context are used to establish the necessary input parameters and in the derivation of the security context (see Section 3.2).

Whereas the 'kid' parameter is used to transport the Sender ID, the new COSE header parameter 'kid context' is used to transport the ID Context in requests, see Figure 7.

| name | label | value type | value registry | description |
|-------------|-------|------------|----------------|--------------------------------|
| kid context | TBD2 | bstr | | Identifies the context for kid |

Figure 7: Common Header Parameter 'kid context' for the COSE object

If ID Context is non-empty and the client sends a request without 'kid context' which results in an error indicating that the server could not find the security context, then the client could include the ID Context in the 'kid context' when making another request. Note that since the error is unprotected it may have been spoofed and the real response blocked by an on-path attacker.

5.2. AEAD Nonce

The high level design of the AEAD nonce follows Section 4.4 of [I-D.mcgregw-iv-gen], here follows the detailed construction (see Figure 8):

1. left-pad the Partial IV (PIV) with zeroes to exactly 5 bytes,
2. left-pad the Sender ID of the endpoint that generated the Partial IV (ID_PIV) with zeroes to exactly nonce length minus 6 bytes,
3. concatenate the size of the ID_PIV (a single byte S) with the padded ID_PIV and the padded PIV,
4. and then XOR with the Common IV.

Note that in this specification only AEAD algorithms that use nonces equal or greater than 7 bytes are supported. The nonce construction with S, ID_PIV, and PIV together with endpoint unique IDs and

encryption keys makes it easy to verify that the nonces used with a specific key will be unique, see Appendix D.4.

If the Partial IV is not present in a response, the nonce from the request is used. For responses that are not notifications (i.e. when there is a single response to a request), the request and the response should typically use the same nonce to reduce message overhead. Both alternatives provide all the required security properties, see Section 7.4 and Appendix D.4. The only non-Observe scenario where a Partial IV must be included in a response is when the server is unable to perform replay protection, see Appendix B.1.2. For processing instructions see Section 8.

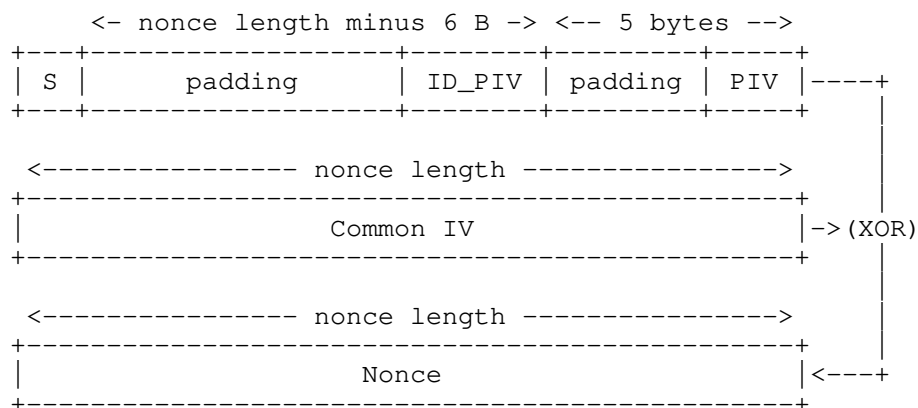


Figure 8: AEAD Nonce Formation

5.3. Plaintext

The plaintext is formatted as a CoAP message without Header (see Figure 9) consisting of:

- o the Code of the original CoAP message as defined in Section 3 of [RFC7252]; and
- o all Inner option message fields (see Section 4.1.1) present in the original CoAP message (see Section 4.1). The options are encoded as described in Section 3.1 of [RFC7252], where the delta is the difference to the previously included instance of Class E option; and
- o the Payload of original CoAP message, if present, and in that case prefixed by the one-byte Payload Marker (0xff).

NOTE: The plaintext contains all CoAP data that needs to be encrypted end-to-end between the endpoints.

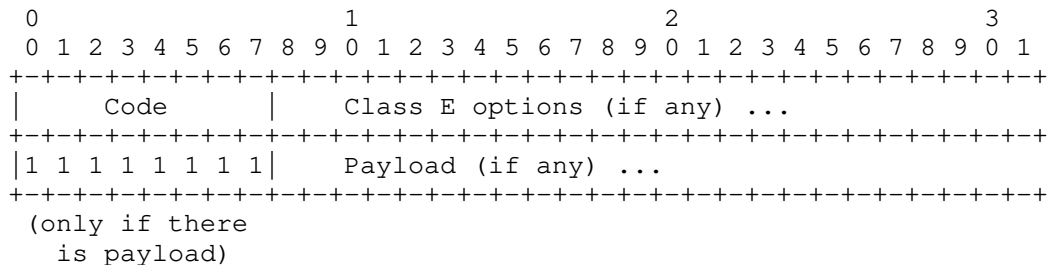


Figure 9: Plaintext

5.4. Additional Authenticated Data

The `external_aad` SHALL be a CBOR array wrapped in a `bstr` object as defined below:

```

external_aad = bstr .cbor aad_array

aad_array = [
  oscore_version : uint,
  algorithms : [ alg_aead : int / tstr ],
  request_kid : bstr,
  request_piv : bstr,
  options : bstr,
]

```

where:

- o `oscore_version`: contains the OSCORE version number. Implementations of this specification MUST set this field to 1. Other values are reserved for future versions.
- o `algorithms`: contains (for extensibility) an array of algorithms, according to this specification only containing `alg_aead`.
- o `alg_aead`: contains the AEAD Algorithm from the security context used for the exchange (see Section 3.1).
- o `request_kid`: contains the value of the 'kid' in the COSE object of the request (see Section 5).
- o `request_piv`: contains the value of the 'Partial IV' in the COSE object of the request (see Section 5).

- o options: contains the Class I options (see Section 4.1.2) present in the original CoAP message encoded as described in Section 3.1 of [RFC7252], where the delta is the difference to the previously included instance of class I option.

The `oscore_version` and `algorithms` parameters are established out-of-band and are thus never transported in OSCORE, but the `external_aad` allows to verify that they are the same in both endpoints.

NOTE: The format of the `external_aad` is for simplicity the same for requests and responses, although some parameters, e.g. `request_kid`, need not be integrity protected in all requests.

The Additional Authenticated Data (AAD) is composed from the `external_aad` as described in Section 5.3 of [RFC8152]:

```
AAD = Enc_structure = [ "Encrypt0", h'', external_aad ]
```

The following is an example of AAD constructed using AEAD Algorithm = AES-CCM-16-64-128 (10), `request_kid` = 0x00, `request_piv` = 0x25 and no Class I options:

- o `oscore_version`: 0x01 (1 byte)
- o `algorithms`: 0x810a (2 bytes)
- o `request_kid`: 0x00 (1 byte)
- o `request_piv`: 0x25 (1 byte)
- o `options`: 0x (0 bytes)
- o `aad_array`: 0x8501810a4100412540 (9 bytes)
- o `external_aad`: 0x498501810a4100412540 (10 bytes)
- o `AAD`: 0x8368456e63727970743040498501810a4100412540 (21 bytes)

Note that the AAD consists of a fixed string of 11 bytes concatenated with the `external_aad`.

6. OSCORE Header Compression

The Concise Binary Object Representation (CBOR) [RFC7049] combines very small message sizes with extensibility. The CBOR Object Signing and Encryption (COSE) [RFC8152] uses CBOR to create compact encoding of signed and encrypted data. COSE is however constructed to support a large number of different stateless use cases, and is not fully

optimized for use as a stateful security protocol, leading to a larger than necessary message expansion. In this section, we define a stateless header compression mechanism, simply removing redundant information from the COSE objects, which significantly reduces the per-packet overhead. The result of applying this mechanism to a COSE object is called the "compressed COSE object".

The COSE_Encrypt0 object used in OSCORE is transported in the OSCORE option and in the Payload. The Payload contains the Ciphertext of the COSE object. The headers of the COSE object are compactly encoded as described in the next section.

6.1. Encoding of the OSCORE Option Value

The value of the OSCORE option SHALL contain the OSCORE flag bits, the Partial IV parameter, the 'kid context' parameter (length and value), and the 'kid' parameter as follows:

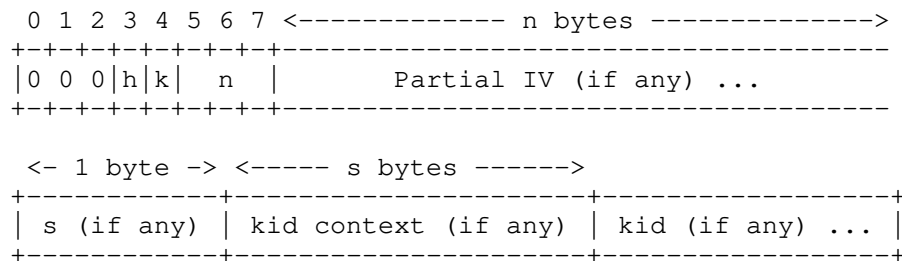


Figure 10: The OSCORE Option Value

- o The first byte, containing the OSCORE flag bits, encodes the following set of bits and the length of the Partial IV parameter:
 - * The three least significant bits encode the Partial IV length n. If n = 0 then the Partial IV is not present in the compressed COSE object. The values n = 6 and n = 7 are reserved.
 - * The fourth least significant bit is the 'kid' flag, k: it is set to 1 if the kid is present in the compressed COSE object.
 - * The fifth least significant bit is the 'kid context' flag, h: it is set to 1 if the compressed COSE object contains a 'kid context' (see Section 5.1).
 - * The sixth to eighth least significant bits are reserved for future use. These bits SHALL be set to zero when not in use. According to this specification, if any of these bits are set

to 1 the message is considered to be malformed and decompression fails as specified in item 2 of Section 8.2.

The flag bits are registered in the OSCORE Flag Bits registry specified in Section 13.7.

- o The following n bytes encode the value of the Partial IV, if the Partial IV is present ($n > 0$).
- o The following 1 byte encode the length of the 'kid context' (Section 5.1) s , if the 'kid context' flag is set ($h = 1$).
- o The following s bytes encode the 'kid context', if the 'kid context' flag is set ($h = 1$).
- o The remaining bytes encode the value of the 'kid', if the 'kid' is present ($k = 1$).

Note that the 'kid' MUST be the last field of the OSCORE option value, even in case reserved bits are used and additional fields are added to it.

The length of the OSCORE option thus depends on the presence and length of Partial IV, 'kid context', 'kid', as specified in this section, and on the presence and length of the other parameters, as defined in the separate documents.

6.2. Encoding of the OSCORE Payload

The payload of the OSCORE message SHALL encode the ciphertext of the COSE object.

6.3. Examples of Compressed COSE Objects

This section covers a list of OSCORE Header Compression examples for requests and responses. The examples assume the COSE_Encrypt0 object is set (which means the CoAP message and cryptographic material is known). Note that the full CoAP unprotected message, as well as the full security context, is not reported in the examples, but only the input necessary to the compression mechanism, i.e. the COSE_Encrypt0 object. The output is the compressed COSE object as defined in Section 6, divided into two parts, since the object is transported in two CoAP fields: OSCORE option and payload.

1. Request with ciphertext = 0xaea0155667924dff8a24e4cb35b9, kid = 0x25, and Partial IV = 0x05

Before compression (24 bytes):

```
[
  h'',
  { 4:h'25', 6:h'05' },
  h'aea0155667924dff8a24e4cb35b9',
]
```

After compression (17 bytes):

Flag byte: 0b00001001 = 0x09 (1 byte)

Option Value: 0x090525 (3 bytes)

Payload: 0xaea0155667924dff8a24e4cb35b9 (14 bytes)

2. Request with ciphertext = 0xaea0155667924dff8a24e4cb35b9, kid = empty string, and Partial IV = 0x00

Before compression (23 bytes):

```
[
  h'',
  { 4:h'', 6:h'00' },
  h'aea0155667924dff8a24e4cb35b9',
]
```

After compression (16 bytes):

Flag byte: 0b00001001 = 0x09 (1 byte)

Option Value: 0x0900 (2 bytes)

Payload: 0xaea0155667924dff8a24e4cb35b9 (14 bytes)

3. Request with ciphertext = 0xaea0155667924dff8a24e4cb35b9, kid = empty string, Partial IV = 0x05, and kid context = 0x44616c656b

Before compression (30 bytes):

```
[
  h'',
  { 4:h'', 6:h'05', 8:h'44616c656b' },
  h'aea0155667924dff8a24e4cb35b9',
]
```

After compression (22 bytes):

Flag byte: 0b00011001 = 0x19 (1 byte)

Option Value: 0x19050544616c656b (8 bytes)

Payload: 0xae a0155667924dff8a24e4cb35b9 (14 bytes)

4. Response with ciphertext = 0xaea0155667924dff8a24e4cb35b9 and no Partial IV

Before compression (18 bytes):

```
[
  h'',
  {},
  h'aea0155667924dff8a24e4cb35b9',
]
```

After compression (14 bytes):

Flag byte: 0b00000000 = 0x00 (1 byte)

Option Value: 0x (0 bytes)

Payload: 0xaea0155667924dff8a24e4cb35b9 (14 bytes)

5. Response with ciphertext = 0xaea0155667924dff8a24e4cb35b9 and Partial IV = 0x07

Before compression (21 bytes):

```
[
  h'',
  { 6:h'07' },
  h'aea0155667924dff8a24e4cb35b9',
]
```

After compression (16 bytes):

Flag byte: 0b00000001 = 0x01 (1 byte)

Option Value: 0x0107 (2 bytes)

Payload: 0xaea0155667924dff8a24e4cb35b9 (14 bytes)

7. Message Binding, Sequence Numbers, Freshness, and Replay Protection

7.1. Message Binding

In order to prevent response delay and mismatch attacks [I-D.mattsson-core-coap-actuators] from on-path attackers and compromised intermediaries, OSCORE binds responses to the requests by including the 'kid' and Partial IV of the request in the AAD of the response. The server therefore needs to store the 'kid' and Partial IV of the request until all responses have been sent.

7.2. Sequence Numbers

An AEAD nonce MUST NOT be used more than once per AEAD key. The uniqueness of (key, nonce) pairs is shown in Appendix D.4, and in particular depends on a correct usage of Partial IVs (which encode the Sender Sequence Numbers, see Section 5). If messages are processed concurrently, the operation of reading and increasing the Sender Sequence Number MUST be atomic.

7.2.1. Maximum Sequence Number

The maximum Sender Sequence Number is algorithm dependent (see Section 12), and SHALL be less than 2^{40} . If the Sender Sequence Number exceeds the maximum, the endpoint MUST NOT process any more messages with the given Sender Context. If necessary, the endpoint SHOULD acquire a new security context before this happens. The latter is out of scope of this document.

7.3. Freshness

For requests, OSCORE provides only the guarantee that the request is not older than the security context. For applications having stronger demands on request freshness (e.g., control of actuators), OSCORE needs to be augmented with mechanisms providing freshness, for example as specified in [I-D.ietf-core-echo-request-tag].

Assuming an honest server (see Appendix D), the message binding guarantees that a response is not older than its request. For responses that are not notifications (i.e. when there is a single response to a request), this gives absolute freshness. For notifications, the absolute freshness gets weaker with time, and it is RECOMMENDED that the client regularly re-register the observation. Note that the message binding does not guarantee that misbehaving server created the response before receiving the request, i.e. it does not verify server aliveness.

For requests and notifications, OSCORE also provides relative freshness in the sense that the received Partial IV allows a recipient to determine the relative order of requests or responses.

7.4. Replay Protection

In order to protect from replay of requests, the server's Recipient Context includes a Replay Window. A server SHALL verify that a Partial IV = Sender Sequence Number received in the COSE object has not been received before. If this verification fails, the server SHALL stop processing the message, and MAY optionally respond with a 4.01 (Unauthorized) error message. Also, the server MAY set an Outer Max-Age option with value zero, to inform any intermediary that the response is not to be cached. The diagnostic payload MAY contain the "Replay detected" string. The size and type of the Replay Window depends on the use case and the protocol with which the OSCORE message is transported. In case of reliable and ordered transport from endpoint to endpoint, e.g. TCP, the server MAY just store the last received Partial IV and require that newly received Partial IVs equals the last received Partial IV + 1. However, in case of mixed reliable and unreliable transports and where messages may be lost, such a replay mechanism may be too restrictive and the default replay window be more suitable (see Section 3.2.2).

Responses (with or without Partial IV) are protected against replay as they are bound to the request and the fact that only a single response is accepted. Note that the Partial IV is not used for replay protection in this case.

The operation of validating the Partial IV and updating the replay protection MUST be atomic.

7.4.1. Replay Protection of Notifications

The following applies additionally when Observe is supported.

The Notification Number is initialized to the Partial IV of the first successfully verified notification in response to the registration request. A client MUST only accept at most one Observe notifications without Partial IV, and treat it as the oldest notification received. A client receiving a notification containing a Partial IV SHALL compare the Partial IV with the Notification Number associated to that Observe registration. The client MUST stop processing notifications with a Partial IV which has been previously received. Applications MAY decide that a client only processes notifications which have greater Partial IV than the Notification Number.

If the verification of the response succeeds, and the received Partial IV was greater than the Notification Number then the client SHALL overwrite the corresponding Notification Number with the received Partial IV.

7.5. Losing Part of the Context State

To prevent reuse of an AEAD nonce with the same AEAD key, or from accepting replayed messages, an endpoint needs to handle the situation of losing rapidly changing parts of the context, such as the Sender Sequence Number, and Replay Window. These are typically stored in RAM and therefore lost in the case of e.g. an unplanned reboot. There are different alternatives to recover, for example:

1. The endpoints can reuse an existing Security Context after updating the mutable parts of the security context (Sender Sequence Number, and Replay Window). This requires that the mutable parts of the security context are available throughout the lifetime of the device, or that the device can establish safe security context after loss of mutable security context data. Examples is given based on careful use of non-volatile memory, see Appendix B.1.1, and additionally the use of the Echo option, see Appendix B.1.2. If an endpoint makes use of a partial security context stored in non-volatile memory, it MUST NOT reuse a previous Sender Sequence Number and MUST NOT accept previously received messages.
2. The endpoints can reuse an existing shared Master Secret and derive new Sender and Recipient Contexts, see Appendix B.2 for an example. This typically requires a good source of randomness.
3. The endpoints can use a trusted-third party assisted key establishment protocol such as [I-D.ietf-ace-oscore-profile]. This requires the execution of three-party protocol and may require a good source of randomness.
4. The endpoints can run a key exchange protocol providing forward secrecy resulting in a fresh Master Secret, from which an entirely new Security Context is derived. This requires a good source of randomness, and additionally, the transmission and processing of the protocol may have a non-negligible cost, e.g. in terms of power consumption.

The endpoints need to be configured with information about which method is used. The choice of method may depend on capabilities of the devices deployed and the solution architecture. Using a key exchange protocol is necessary for deployments that require forward secrecy.

8. Processing

This section describes the OSCORE message processing. Additional processing for Observe or Block-wise are described in subsections.

Note that, analogously to [RFC7252] where the Token and source/destination pair are used to match a response with a request, both endpoints MUST keep the association (Token, {Security Context, Partial IV of the request}), in order to be able to find the Security Context and compute the AAD to protect or verify the response. The association MAY be forgotten after it has been used to successfully protect or verify the response, with the exception of Observe processing, where the association MUST be kept as long as the Observation is active.

The processing of the Sender Sequence Number follows the procedure described in Section 3 of [I-D.mcgregor-iv-gen].

8.1. Protecting the Request

Given a CoAP request, the client SHALL perform the following steps to create an OSCORE request:

1. Retrieve the Sender Context associated with the target resource.
2. Compose the Additional Authenticated Data and the plaintext, as described in Sections 5.3 and 5.4.
3. Encode the Partial IV (Sender Sequence Number in network byte order) and increment the Sender Sequence Number by one. Compute the AEAD nonce from the Sender ID, Common IV, and Partial IV as described in Section 5.2.
4. Encrypt the COSE object using the Sender Key. Compress the COSE Object as specified in Section 6.
5. Format the OSCORE message according to Section 4. The OSCORE option is added (see Section 4.1.2).

8.2. Verifying the Request

A server receiving a request containing the OSCORE option SHALL perform the following steps:

1. Discard Code and all class E options (marked in Figure 5 with 'x' in column E) present in the received message. For example, an If-Match Outer option is discarded, but an Uri-Host Outer option is not discarded.

2. Decompress the COSE Object (Section 6) and retrieve the Recipient Context associated with the Recipient ID in the 'kid' parameter, additionally using the 'kid context', if present. If either the decompression or the COSE message fails to decode, or the server fails to retrieve a Recipient Context with Recipient ID corresponding to the 'kid' parameter received, then the server SHALL stop processing the request.
 - * If either the decompression or the COSE message fails to decode, the server MAY respond with a 4.02 (Bad Option) error message. The server MAY set an Outer Max-Age option with value zero. The diagnostic payload MAY contain the string "Failed to decode COSE".
 - * If the server fails to retrieve a Recipient Context with Recipient ID corresponding to the 'kid' parameter received, the server MAY respond with a 4.01 (Unauthorized) error message. The server MAY set an Outer Max-Age option with value zero. The diagnostic payload MAY contain the string "Security context not found".
3. Verify that the 'Partial IV' has not been received before using the Replay Window, as described in Section 7.4.
4. Compose the Additional Authenticated Data, as described in Section 5.4.
5. Compute the AEAD nonce from the Recipient ID, Common IV, and the 'Partial IV' parameter, received in the COSE Object.
6. Decrypt the COSE object using the Recipient Key, as per [RFC8152] Section 5.3. (The decrypt operation includes the verification of the integrity.)
 - * If decryption fails, the server MUST stop processing the request and MAY respond with a 4.00 (Bad Request) error message. The server MAY set an Outer Max-Age option with value zero. The diagnostic payload MAY contain the "Decryption failed" string.
 - * If decryption succeeds, update the Replay Window, as described in Section 7.
7. Add decrypted Code, options, and payload to the decrypted request. The OSCORE option is removed.
8. The decrypted CoAP request is processed according to [RFC7252].

8.2.1. Supporting Block-wise

If Block-wise is supported, insert the following step before any other:

A. If Block-wise is present in the request, then process the Outer Block options according to [RFC7959], until all blocks of the request have been received (see Section 4.1.3.4).

8.3. Protecting the Response

If a CoAP response is generated in response to an OSCORE request, the server SHALL perform the following steps to create an OSCORE response. Note that CoAP error responses derived from CoAP processing (step 8 in Section 8.2) are protected, as well as successful CoAP responses, while the OSCORE errors (steps 2, 3, and 6 in Section 8.2) do not follow the processing below, but are sent as simple CoAP responses, without OSCORE processing.

1. Retrieve the Sender Context in the Security Context associated with the Token.
2. Compose the Additional Authenticated Data and the plaintext, as described in Sections 5.3 and 5.4.
3. Compute the AEAD nonce as described in Section 5.2:
 - * Either use the AEAD nonce from the request, or
 - * Encode the Partial IV (Sender Sequence Number in network byte order) and increment the Sender Sequence Number by one. Compute the AEAD nonce from the Sender ID, Common IV, and Partial IV.
4. Encrypt the COSE object using the Sender Key. Compress the COSE Object as specified in Section 6. If the AEAD nonce was constructed from a new Partial IV, this Partial IV MUST be included in the message. If the AEAD nonce from the request was used, the Partial IV MUST NOT be included in the message.
5. Format the OSCORE message according to Section 4. The OSCORE option is added (see Section 4.1.2).

8.3.1. Supporting Observe

If Observe is supported, insert the following step between step 2 and 3 of Section 8.3:

- A. If the response is an observe notification:
 - o If the response is the first notification:
 - * compute the AEAD nonce as described in Section 5.2:
 - + Either use the AEAD nonce from the request, or
 - + Encode the Partial IV (Sender Sequence Number in network byte order) and increment the Sender Sequence Number by one. Compute the AEAD nonce from the Sender ID, Common IV, and Partial IV.
 - Then go to 4.
- o If the response is not the first notification:
 - * encode the Partial IV (Sender Sequence Number in network byte order) and increment the Sender Sequence Number by one. Compute the AEAD nonce from the Sender ID, Common IV, and Partial IV, then go to 4.

8.4. Verifying the Response

A client receiving a response containing the OSCORE option SHALL perform the following steps:

1. Discard Code and all class E options (marked in Figure 5 with 'x' in column E) present in the received message. For example, ETag Outer option is discarded, as well as Max-Age Outer option.
2. Retrieve the Recipient Context in the Security Context associated with the Token. Decompress the COSE Object (Section 6). If either the decompression or the COSE message fails to decode, then go to 8.
3. Compose the Additional Authenticated Data, as described in Section 5.4.
4. Compute the AEAD nonce
 - * If the Partial IV is not present in the response, the AEAD nonce from the request is used.
 - * If the Partial IV is present in the response, compute the AEAD nonce from the Recipient ID, Common IV, and the 'Partial IV' parameter, received in the COSE Object.

5. Decrypt the COSE object using the Recipient Key, as per [RFC8152] Section 5.3. (The decrypt operation includes the verification of the integrity.) If decryption fails, then go to 8.
6. Add decrypted Code, options and payload to the decrypted request. The OSCORE option is removed.
7. The decrypted CoAP response is processed according to [RFC7252].
8. In case any of the previous erroneous conditions apply: the client SHALL stop processing the response.

8.4.1. Supporting Block-wise

If Block-wise is supported, insert the following step before any other:

- A. If Block-wise is present in the request, then process the Outer Block options according to [RFC7959], until all blocks of the request have been received (see Section 4.1.3.4).

8.4.2. Supporting Observe

If Observe is supported:

Insert the following step between step 5 and step 6:

- A. If the request was an Observe registration, then:
 - o If the Partial IV is not present in the response, and Inner Observe is present, and the AEAD nonce from the request was already used once, then go to 8.
 - o If the Partial IV is present in the response and Inner Observe is present, then follow the processing described in Section 4.1.3.5.2 and Section 7.4.1, then:
 - * initialize the Notification Number (if first successfully verified notification), or
 - * overwrite the Notification Number (if the received Partial IV was greater than the Notification Number).

Replace step 8 of Section 8.4 with:

- B. In case any of the previous erroneous conditions apply: the client SHALL stop processing the response. An error condition occurring while processing a response to an observation request does

not cancel the observation. A client MUST NOT react to failure by re-registering the observation immediately.

9. Web Linking

The use of OSCORE MAY be indicated by a target attribute "osc" in a web link [RFC8288] to a resource, e.g. using a link-format document [RFC6690] if the resource is accessible over CoAP.

The "osc" attribute is a hint indicating that the destination of that link is only accessible using OSCORE, and unprotected access to it is not supported. Note that this is simply a hint, it does not include any security context material or any other information required to run OSCORE.

A value MUST NOT be given for the "osc" attribute; any present value MUST be ignored by parsers. The "osc" attribute MUST NOT appear more than once in a given link-value; occurrences after the first MUST be ignored by parsers.

The example in Figure 11 shows a use of the "osc" attribute: the client does resource discovery on a server, and gets back a list of resources, one of which includes the "osc" attribute indicating that the resource is protected with OSCORE. The link-format notation (see Section 5 of [RFC6690]) is used.

```
REQ: GET /.well-known/core
RES: 2.05 Content
    </sensors/temp>;osc,
    </sensors/light>;if="sensor"
```

Figure 11: The web link

10. CoAP-to-CoAP Forwarding Proxy

CoAP is designed for proxy operations (see Section 5.7 of [RFC7252]).

OSCORE is designed to work with OSCORE-unaware CoAP proxies. Security requirements for forwarding are listed in Section 2.2.1 of [I-D.hartke-core-e2e-security-reqs]. Proxy processing of the (Outer) Proxy-Uri option works as defined in [RFC7252]. Proxy processing of the (Outer) Block options works as defined in [RFC7959].

However, not all CoAP proxy operations are useful:

- o Since a CoAP response is only applicable to the original CoAP request, caching is in general not useful. In support of existing

proxies, OSCORE uses the outer Max-Age option, see Section 4.1.3.1.

- o Proxy processing of the (Outer) Observe option as defined in [RFC7641] is specified in Section 4.1.3.5.

Optionally, a CoAP proxy MAY detect OSCORE and act accordingly. An OSCORE-aware CoAP proxy:

- o SHALL bypass caching for the request if the OSCORE option is present
- o SHOULD avoid caching responses to requests with an OSCORE option

In the case of Observe (see Section 4.1.3.5) the OSCORE-aware CoAP proxy:

- o SHALL NOT initiate an Observe registration
- o MAY verify the order of notifications using Partial IV rather than the Observe option

11. HTTP Operations

The CoAP request/response model may be mapped to HTTP and vice versa as described in Section 10 of [RFC7252]. The HTTP-CoAP mapping is further detailed in [RFC8075]. This section defines the components needed to map and transport OSCORE messages over HTTP hops. By mapping between HTTP and CoAP and by using cross-protocol proxies OSCORE may be used end-to-end between e.g. an HTTP client and a CoAP server. Examples are provided at the end of the section.

11.1. The HTTP OSCORE Header Field

The HTTP OSCORE Header Field (see Section 13.4) is used for carrying the content of the CoAP OSCORE option when transporting OSCORE messages over HTTP hops.

The HTTP OSCORE header field is only used in POST requests and 200 (OK) responses. When used, the HTTP header field Content-Type is set to 'application/oscore' (see Section 13.5) indicating that the HTTP body of this message contains the OSCORE payload (see Section 6.2). No additional semantics is provided by other message fields.

Using the Augmented Backus-Naur Form (ABNF) notation of [RFC5234], including the following core ABNF syntax rules defined by that specification: ALPHA (letters) and DIGIT (decimal digits), the HTTP OSCORE header field value is as follows.

base64url-char = ALPHA / DIGIT / "-" / "_"

OSCORE = 2*base64url-char

The HTTP OSCORE header field is not appropriate to list in the Connection header field (see Section 6.1 of [RFC7230]) since it is not hop-by-hop. OSCORE messages are generally not useful when served from cache (i.e., they will generally be marked Cache-Control: no-cache) and so interaction with Vary is not relevant (Section 7.1.4 of [RFC7231]). Since the HTTP OSCORE header field is critical for message processing, moving it from headers to trailers renders the message unusable in case trailers are ignored (see Section 4.1 of [RFC7230]).

Intermediaries are in general not allowed to insert, delete, or modify the OSCORE header. Changes to the HTTP OSCORE header field will in general violate the integrity of the OSCORE message resulting in an error. For the same reason the HTTP OSCORE header field is in general not preserved across redirects.

Since redirects are not defined in the mappings between HTTP and CoAP [RFC8075][RFC7252], a number of conditions need to be fulfilled for redirects to work. For CoAP client to HTTP server, such conditions include:

- o the CoAP-to-HTTP proxy follows the redirect, instead of the CoAP client as in the HTTP case
- o the CoAP-to-HTTP proxy copies the HTTP OSCORE header field and body to the new request
- o the target of the redirect has the necessary OSCORE security context required to decrypt and verify the message

Since OSCORE requires HTTP body to be preserved across redirects, the HTTP server is RECOMMENDED to reply with 307 or 308 instead of 301 or 302.

For the case of HTTP client to CoAP server, although redirect is not defined for CoAP servers [RFC7252], an HTTP client receiving a redirect should generate a new OSCORE request for the server it was redirected to.

11.2. CoAP-to-HTTP Mapping

Section 10.1 of [RFC7252] describes the fundamentals of the CoAP-to-HTTP cross-protocol mapping process. The additional rules for OSCORE messages are:

- o The HTTP OSCORE header field value is set to
 - * AA if the CoAP OSCORE option is empty, otherwise
 - * the value of the CoAP OSCORE option (Section 6.1) in base64url (Section 5 of [RFC4648]) encoding without padding. Implementation notes for this encoding are given in Appendix C of [RFC7515].
- o The HTTP Content-Type is set to 'application/oscore' (see Section 13.5), independent of CoAP Content-Format.

11.3. HTTP-to-CoAP Mapping

Section 10.2 of [RFC7252] and [RFC8075] specify the behavior of an HTTP-to-CoAP proxy. The additional rules for HTTP messages with the OSCORE header field are:

- o The CoAP OSCORE option is set as follows:
 - * empty if the value of the HTTP OSCORE header field is a single zero byte (0x00) represented by AA, otherwise
 - * the value of the HTTP OSCORE header field decoded from base64url (Section 5 of [RFC4648]) without padding. Implementation notes for this encoding are given in Appendix C of [RFC7515].
- o The CoAP Content-Format option is omitted, the content format for OSCORE (Section 13.6) MUST NOT be used.

11.4. HTTP Endpoints

Restricted to subsets of HTTP and CoAP supporting a bijective mapping, OSCORE can be originated or terminated in HTTP endpoints.

The sending HTTP endpoint uses [RFC8075] to translate the HTTP message into a CoAP message. The CoAP message is then processed with OSCORE as defined in this document. The OSCORE message is then mapped to HTTP as described in Section 11.2 and sent in compliance with the rules in Section 11.1.

The receiving HTTP endpoint maps the HTTP message to a CoAP message using [RFC8075] and Section 11.3. The resulting OSCORE message is processed as defined in this document. If successful, the plaintext CoAP message is translated to HTTP for normal processing in the endpoint.

11.5. Example: HTTP Client and CoAP Server

This section is giving an example of how a request and a response between an HTTP client and a CoAP server could look like. The example is not a test vector but intended as an illustration of how the message fields are translated in the different steps.

Mapping and notation here is based on "Simple Form" (Section 5.4.1 of [RFC8075]).

[HTTP request -- Before client object security processing]

```
GET http://proxy.url/hc/?target_uri=coap://server.url/orders
HTTP/1.1
```

[HTTP request -- HTTP Client to Proxy]

```
POST http://proxy.url/hc/?target_uri=coap://server.url/ HTTP/1.1
Content-Type: application/oscore
OSCORE: CSU
Body: 09 07 01 13 61 f7 0f d2 97 b1 [binary]
```

[CoAP request -- Proxy to CoAP Server]

```
POST coap://server.url/
OSCORE: 09 25
Payload: 09 07 01 13 61 f7 0f d2 97 b1 [binary]
```

[CoAP request -- After server object security processing]

```
GET coap://server.url/orders
```

[CoAP response -- Before server object security processing]

```
2.05 Content
Content-Format: 0
Payload: Exterminate! Exterminate!
```

[CoAP response -- CoAP Server to Proxy]

```
2.04 Changed
OSCORE: [empty]
Payload: 00 31 d1 fc f6 70 fb 0c 1d d5 ... [binary]
```

[HTTP response -- Proxy to HTTP Client]

```
HTTP/1.1 200 OK
Content-Type: application/oscore
OSCORE: AA
Body: 00 31 d1 fc f6 70 fb 0c 1d d5 ... [binary]
```

[HTTP response -- After client object security processing]

```
HTTP/1.1 200 OK
Content-Type: text/plain
Body: Exterminate! Exterminate!
```

Note that the HTTP Status Code 200 in the next-to-last message is the mapping of CoAP Code 2.04 (Changed), whereas the HTTP Status Code 200 in the last message is the mapping of the CoAP Code 2.05 (Content), which was encrypted within the compressed COSE object carried in the Body of the HTTP response.

11.6. Example: CoAP Client and HTTP Server

This section is giving an example of how a request and a response between a CoAP client and an HTTP server could look like. The example is not a test vector but intended as an illustration of how the message fields are translated in the different steps

[CoAP request -- Before client object security processing]

```
GET coap://proxy.url/
Proxy-Uri=http://server.url/orders
```

[CoAP request -- CoAP Client to Proxy]

```
POST coap://proxy.url/
Proxy-Uri=http://server.url/
OSCORE: 09 25
Payload: 09 07 01 13 61 f7 0f d2 97 b1 [binary]
```

[HTTP request -- Proxy to HTTP Server]

```
POST http://server.url/ HTTP/1.1
Content-Type: application/oscore
OSCORE: CSU
Body: 09 07 01 13 61 f7 0f d2 97 b1 [binary]
```

[HTTP request -- After server object security processing]

```
GET http://server.url/orders HTTP/1.1
```

[HTTP response -- Before server object security processing]

HTTP/1.1 200 OK
Content-Type: text/plain
Body: Exterminate! Exterminate!

[HTTP response -- HTTP Server to Proxy]

HTTP/1.1 200 OK
Content-Type: application/oscore
OSCORE: AA
Body: 00 31 d1 fc f6 70 fb 0c 1d d5 ... [binary]

[CoAP response -- Proxy to CoAP Client]

2.04 Changed
OSCORE: [empty]
Payload: 00 31 d1 fc f6 70 fb 0c 1d d5 ... [binary]

[CoAP response -- After client object security processing]

2.05 Content
Content-Format: 0
Payload: Exterminate! Exterminate!

Note that the HTTP Code 2.04 (Changed) in the next-to-last message is the mapping of HTTP Status Code 200, whereas the CoAP Code 2.05 (Content) in the last message is the value that was encrypted within the compressed COSE object carried in the Body of the HTTP response.

12. Security Considerations

An overview of the security properties is given in Appendix D.

12.1. End-to-end Protection

In scenarios with intermediary nodes such as proxies or gateways, transport layer security such as (D)TLS only protects data hop-by-hop. As a consequence, the intermediary nodes can read and modify any information. The trust model where all intermediary nodes are considered trustworthy is problematic, not only from a privacy perspective, but also from a security perspective, as the intermediaries are free to delete resources on sensors and falsify commands to actuators (such as "unlock door", "start fire alarm", "raise bridge"). Even in the rare cases where all the owners of the intermediary nodes are fully trusted, attacks and data breaches make such an architecture brittle.

(D)TLS protects hop-by-hop the entire message. OSCORE protects end-to-end all information that is not required for proxy operations (see Section 4). (D)TLS and OSCORE can be combined, thereby enabling end-to-end security of the message payload, in combination with hop-by-hop protection of the entire message, during transport between endpoint and intermediary node. In particular when OSCORE is used with HTTP, the additional TLS protection of HTTP hops is RECOMMENDED, e.g. between an HTTP endpoint and a proxy translating between HTTP and CoAP.

Applications need to consider that certain message fields and messages types are not protected end-to-end and may be spoofed or manipulated. The consequences of unprotected message fields are analyzed in Appendix D.5.

12.2. Security Context Establishment

The use of COSE_Encrypt0 and AEAD to protect messages as specified in this document requires an established security context. The method to establish the security context described in Section 3.2 is based on a common Master Secret and unique Sender IDs. The necessary input parameters may be pre-established or obtained using a key establishment protocol augmented with establishment of Sender/Recipient ID, such as a key exchange protocol or the OSCORE profile of the ACE framework [I-D.ietf-ace-oscore-profile]. Such a procedure must ensure that the requirements of the security context parameters for the intended use are complied with (see Section 3.3) and also in error situations. While recipient IDs are allowed to coincide between different security contexts (see Section 3.3), this may cause a server to process multiple verifications before finding the right security context or rejecting a message. Considerations for deploying OSCORE with a fixed Master Secret are given in Appendix B.

12.3. Master Secret

OSCORE uses HKDF [RFC5869] and the established input parameters to derive the security context. The required properties of the security context parameters are discussed in Section 3.3, in this section we focus on the Master Secret. HKDF denotes in this specification the composition of the expand and extract functions as defined in [RFC5869] and the Master Secret is used as Input Key Material (IKM).

Informally, HKDF takes as source an IKM containing some good amount of randomness but not necessarily distributed uniformly (or for which an attacker has some partial knowledge) and derive from it one or more cryptographically strong secret keys [RFC5869].

Therefore, the main requirement for the OSCORE Master Secret, in addition to being secret, is that it has a good amount of randomness. The selected key establishment schemes must ensure that the necessary properties for the Master Secret are fulfilled. For pre-shared key deployments and key transport solutions such as [I-D.ietf-ace-oscore-profile], the Master Secret can be generated offline using a good random number generator. Randomness requirements for security are described in [RFC4086].

12.4. Replay Protection

Replay attacks need to be considered in different parts of the implementation. Most AEAD algorithms require a unique nonce for each message, for which the sender sequence numbers in the COSE message field 'Partial IV' is used. If the recipient accepts any sequence number larger than the one previously received, then the problem of sequence number synchronization is avoided. With reliable transport, it may be defined that only messages with sequence number which are equal to previous sequence number + 1 are accepted. An adversary may try to induce a device reboot for the purpose of replaying a message (see Section 7.5).

Note that sharing a security context between servers may open up for replay attacks, for example if the replay windows are not synchronized.

12.5. Client Aliveness

A verified OSCORE request enables the server to verify the identity of the entity who generated the message. However, it does not verify that the client is currently involved in the communication, since the message may be a delayed delivery of a previously generated request which now reaches the server. To verify the aliveness of the client the server may use the Echo option in the response to a request from the client (see [I-D.ietf-core-echo-request-tag]).

12.6. Cryptographic Considerations

The maximum sender sequence number is dependent on the AEAD algorithm. The maximum sender sequence number is $2^{40} - 1$, or any algorithm specific lower limit, after which a new security context must be generated. The mechanism to build the AEAD nonce (Section 5.2) assumes that the nonce is at least 56 bits, and the Partial IV is at most 40 bits. The mandatory-to-implement AEAD algorithm AES-CCM-16-64-128 is selected for compatibility with CCM*. AEAD algorithms that require unpredictable nonces are not supported.

In order to prevent cryptanalysis when the same plaintext is repeatedly encrypted by many different users with distinct AEAD keys, the AEAD nonce is formed by mixing the sequence number with a secret per-context initialization vector (Common IV) derived along with the keys (see Section 3.1 of [RFC8152]), and by using a Master Salt in the key derivation (see [MF00] for an overview). The Master Secret, Sender Key, Recipient Key, and Common IV must be secret, the rest of the parameters may be public. The Master Secret must have a good amount of randomness (see Section 12.3).

The ID Context, Sender ID, and Partial IV are always at least implicitly integrity protected, as manipulation leads to the wrong nonce or key being used and therefore results in decryption failure.

12.7. Message Segmentation

The Inner Block options enable the sender to split large messages into OSCORE-protected blocks such that the receiving endpoint can verify blocks before having received the complete message. The Outer Block options allow for arbitrary proxy fragmentation operations that cannot be verified by the endpoints, but can by policy be restricted in size since the Inner Block options allow for secure fragmentation of very large messages. A maximum message size (above which the sending endpoint fragments the message and the receiving endpoint discards the message, if complying to the policy) may be obtained as part of normal resource discovery.

12.8. Privacy Considerations

Privacy threats executed through intermediary nodes are considerably reduced by means of OSCORE. End-to-end integrity protection and encryption of the message payload and all options that are not used for proxy operations, provide mitigation against attacks on sensor and actuator communication, which may have a direct impact on the personal sphere.

The unprotected options (Figure 5) may reveal privacy sensitive information, see Appendix D.5. CoAP headers sent in plaintext allow, for example, matching of CON and ACK (CoAP Message Identifier), matching of request and responses (Token) and traffic analysis. OSCORE does not provide protection for HTTP header fields which are not both CoAP-mappable and class E. The HTTP message fields which are visible to on-path entity are only used for the purpose of transporting the OSCORE message, whereas the application layer message is encoded in CoAP and encrypted.

COSE message fields, i.e. the OSCORE option, may reveal information about the communicating endpoints. E.g. 'kid' and 'kid context',

which are intended to help the server find the right context, may reveal information about the client. Tracking 'kid' and 'kid context' to one server may be used for correlating requests from one client.

Unprotected error messages reveal information about the security state in the communication between the endpoints. Unprotected signaling messages reveal information about the reliable transport used on a leg of the path. Using the mechanisms described in Section 7.5 may reveal when a device goes through a reboot. This can be mitigated by the device storing the precise state of sender sequence number and replay window on a clean shutdown.

The length of message fields can reveal information about the message. Applications may use a padding scheme to protect against traffic analysis.

13. IANA Considerations

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

Note to IANA: Please note all occurrences of "TBD1" in this specification should be assigned the same number.

13.1. COSE Header Parameters Registry

The 'kid context' parameter is added to the "COSE Header Parameters Registry":

- o Name: kid context
- o Label: TBD2
- o Value Type: bstr
- o Value Registry:
- o Description: Identifies the context for 'kid'
- o Reference: Section 5.1 of this document

Note to IANA: Label assignment in (Integer value between 1 and 255) is requested. (RFC Editor: Delete this note after IANA assignment)

13.2. CoAP Option Numbers Registry

The OSCORE option is added to the CoAP Option Numbers registry:

| Number | Name | Reference |
|--------|--------|-------------------|
| TBD1 | OSCORE | [[this document]] |

Note to IANA: Label assignment in (Integer value between 0 and 12) is requested. We also request Expert review if possible, to make sure a correct number for the option is selected (RFC Editor: Delete this note after IANA assignment)

Furthermore, the following existing entries in the CoAP Option Numbers registry are updated with a reference to the document specifying OSCORE processing of that option:

| Number | Name | Reference |
|--------|----------------|---------------------------------------|
| 1 | If-Match | [RFC7252] [[this document]] |
| 3 | Uri-Host | [RFC7252] [[this document]] |
| 4 | ETag | [RFC7252] [[this document]] |
| 5 | If-None-Match | [RFC7252] [[this document]] |
| 6 | Observe | [RFC7641] [[this document]] |
| 7 | Uri-Port | [RFC7252] [[this document]] |
| 8 | Location-Path | [RFC7252] [[this document]] |
| 11 | Uri-Path | [RFC7252] [[this document]] |
| 12 | Content-Format | [RFC7252] [[this document]] |
| 14 | Max-Age | [RFC7252] [[this document]] |
| 15 | Uri-Query | [RFC7252] [[this document]] |
| 17 | Accept | [RFC7252] [[this document]] |
| 20 | Location-Query | [RFC7252] [[this document]] |
| 23 | Block2 | [RFC7959] [RFC8323] [[this document]] |
| 27 | Block1 | [RFC7959] [RFC8323] [[this document]] |
| 28 | Size2 | [RFC7959] [[this document]] |
| 35 | Proxy-Uri | [RFC7252] [[this document]] |
| 39 | Proxy-Scheme | [RFC7252] [[this document]] |
| 60 | Size1 | [RFC7252] [[this document]] |
| 258 | No-Response | [RFC7967] [[this document]] |

Future additions to the CoAP Option Numbers registry need to provide a reference to the document where the OSCORE processing of that CoAP Option is defined.

13.3. CoAP Signaling Option Numbers Registry

The OSCORE option is added to the CoAP Signaling Option Numbers registry:

| Applies to | Number | Name | Reference |
|------------|--------|--------|-------------------|
| 7.xx (all) | TBD1 | OSCORE | [[this document]] |

Note to IANA: The value in the "Number" field is the same value that's being assigned to the new Option Number. Please make sure TBD1 is not the same as any value in Numbers for any existing entry in the CoAP Signaling Option Numbers registry (at the time of writing this, that means make sure TBD1 is not 2 or 4) (RFC Editor: Delete this note after IANA assignment)

13.4. Header Field Registrations

The HTTP OSCORE header field is added to the Message Headers registry:

| Header Field Name | Protocol | Status | Reference |
|-------------------|----------|----------|------------------------------------|
| OSCORE | http | standard | [[this document]], Section 11.1 |

13.5. Media Type Registrations

This section registers the 'application/oscore' media type in the "Media Types" registry. These media types are used to indicate that the content is an OSCORE message. The OSCORE body cannot be understood without the OSCORE header field value and the security context.

Type name: application

Subtype name: oscore

Required parameters: N/A

Optional parameters: N/A

Encoding considerations: binary

Security considerations: See the Security Considerations section of [[This document]].

Interoperability considerations: N/A

Published specification: [[This document]]

Applications that use this media type: IoT applications sending security content over HTTP(S) transports.

Fragment identifier considerations: N/A

Additional information:

- * Deprecated alias names for this type: N/A

- * Magic number(s): N/A

- * File extension(s): N/A

- * Macintosh file type code(s): N/A

Person & email address to contact for further information:
iesg@ietf.org

Intended usage: COMMON

Restrictions on usage: N/A

Author: Goeran Selander, goran.selander@ericsson.com

Change Controller: IESG

Provisional registration? No

13.6. CoAP Content-Formats Registry

Note to IANA: ID assignment in the 10000-64999 range is requested.
(RFC Editor: Delete this note after IANA assignment)

This section registers the media type 'application/oscore' media type in the "CoAP Content-Formats" registry. This Content-Format for the OSCORE payload is defined for potential future use cases and SHALL NOT be used in the OSCORE message. The OSCORE payload cannot be understood without the OSCORE option value and the security context.

| Media Type | Encoding | ID | Reference |
|--------------------|----------|------|-------------------|
| application/oscore | | TBD3 | [[this document]] |

13.7. OSCORE Flag Bits Registry

This document defines a sub-registry for the OSCORE flag bits within the "CoRE Parameters" registry. The name of the sub-registry is "OSCORE Flag Bits". The registry should be created with the Expert Review policy. Guidelines for the experts are provided in Section 13.8.

The columns of the registry are:

- o bit position: This indicates the position of the bit in the set of OSCORE flag bits, starting at 0 for the most significant bit. The bit position must be an integer or a range of integers, in the range 0 to 63.
- o name: The name is present to make it easier to refer to and discuss the registration entry. The value is not used in the protocol. Names are to be unique in the table.
- o description: This contains a brief description of the use of the bit.
- o specification: This contains a pointer to the specification defining the entry.

The initial contents of the registry can be found in the table below. The specification column for all rows in that table should be this document. The entries with Bit Position of 0 and 1 are to be marked as 'Reserved'. The entry with Bit Position of 1 is going to be specified in a future document, and will be used to expand the space

for the OSCORE flag bits in Section 6.1, so that entries 8-63 of the registry are defined.

| Bit Position | Name | Description | Specification |
|--------------|-------------------|--|-------------------|
| 0 | Reserved | | |
| 1 | Reserved | | |
| 2 | Unassigned | | |
| 3 | Kid Context Flag | Set to 1 if 'kid context' is present in the compressed COSE object | [[this document]] |
| 4 | Kid Flag | Set to 1 if kid is present in the compressed COSE object | [[this document]] |
| 5-7 | Partial IV Length | Encodes the Partial IV length; can have value 0 to 5 | [[this document]] |
| 8-63 | Unassigned | | |

13.8. Expert Review Instructions

The expert reviewers for the registry defined in this document are expected to ensure that the usage solves a valid use case that could not be solved better in a different way, that it is not going to duplicate one that is already registered, and that the registered point is likely to be used in deployments. They are furthermore expected to check the clarity of purpose and use of the requested code points. Experts should take into account the expected usage of entries when approving point assignment, and the length of the encoded value should be weighed against the number of code points left that encode to that size and the size of device it will be used on. Experts should block registration for entries 8-63 until these points are defined (i.e. until the mechanism for the OSCORE flag bits expansion via bit 1 is specified).

14. References

14.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC4086] Eastlake 3rd, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, DOI 10.17487/RFC4086, June 2005, <<https://www.rfc-editor.org/info/rfc4086>>.
- [RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <<https://www.rfc-editor.org/info/rfc4648>>.
- [RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <<https://www.rfc-editor.org/info/rfc5234>>.
- [RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <<https://www.rfc-editor.org/info/rfc6347>>.
- [RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <<https://www.rfc-editor.org/info/rfc7049>>.
- [RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <<https://www.rfc-editor.org/info/rfc7230>>.
- [RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <<https://www.rfc-editor.org/info/rfc7231>>.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<https://www.rfc-editor.org/info/rfc7252>>.

- [RFC7641] Hartke, K., "Observing Resources in the Constrained Application Protocol (CoAP)", RFC 7641, DOI 10.17487/RFC7641, September 2015, <<https://www.rfc-editor.org/info/rfc7641>>.
- [RFC7959] Bormann, C. and Z. Shelby, Ed., "Block-Wise Transfers in the Constrained Application Protocol (CoAP)", RFC 7959, DOI 10.17487/RFC7959, August 2016, <<https://www.rfc-editor.org/info/rfc7959>>.
- [RFC8075] Castellani, A., Loreto, S., Rahman, A., Fossati, T., and E. Dijk, "Guidelines for Mapping Implementations: HTTP to the Constrained Application Protocol (CoAP)", RFC 8075, DOI 10.17487/RFC8075, February 2017, <<https://www.rfc-editor.org/info/rfc8075>>.
- [RFC8132] van der Stok, P., Bormann, C., and A. Sehgal, "PATCH and FETCH Methods for the Constrained Application Protocol (CoAP)", RFC 8132, DOI 10.17487/RFC8132, April 2017, <<https://www.rfc-editor.org/info/rfc8132>>.
- [RFC8152] Schaad, J., "CBOR Object Signing and Encryption (COSE)", RFC 8152, DOI 10.17487/RFC8152, July 2017, <<https://www.rfc-editor.org/info/rfc8152>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8288] Nottingham, M., "Web Linking", RFC 8288, DOI 10.17487/RFC8288, October 2017, <<https://www.rfc-editor.org/info/rfc8288>>.
- [RFC8323] Bormann, C., Lemay, S., Tschafenig, H., Hartke, K., Silverajan, B., and B. Raymor, Ed., "CoAP (Constrained Application Protocol) over TCP, TLS, and WebSockets", RFC 8323, DOI 10.17487/RFC8323, February 2018, <<https://www.rfc-editor.org/info/rfc8323>>.
- [RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <<https://www.rfc-editor.org/info/rfc8446>>.

14.2. Informative References

- [I-D.bormann-6lo-coap-802-15-ie]
Bormann, C., "Constrained Application Protocol (CoAP) over IEEE 802.15.4 Information Element for IETF", draft-bormann-6lo-coap-802-15-ie-00 (work in progress), April 2016.
- [I-D.hartke-core-e2e-security-reqs]
Selander, G., Palombini, F., and K. Hartke, "Requirements for CoAP End-To-End Security", draft-hartke-core-e2e-security-reqs-03 (work in progress), July 2017.
- [I-D.ietf-ace-oauth-authz]
Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "Authentication and Authorization for Constrained Environments (ACE) using the OAuth 2.0 Framework (ACE-OAuth)", draft-ietf-ace-oauth-authz-22 (work in progress), March 2019.
- [I-D.ietf-ace-oscore-profile]
Palombini, F., Seitz, L., Selander, G., and M. Gunnarsson, "OSCORE profile of the Authentication and Authorization for Constrained Environments Framework", draft-ietf-ace-oscore-profile-07 (work in progress), February 2019.
- [I-D.ietf-cbor-cddl]
Birkholz, H., Vigano, C., and C. Bormann, "Concise data definition language (CDDL): a notational convention to express CBOR and JSON data structures", draft-ietf-cbor-cddl-07 (work in progress), February 2019.
- [I-D.ietf-core-echo-request-tag]
Amsuess, C., Mattsson, J., and G. Selander, "Echo and Request-Tag", draft-ietf-core-echo-request-tag-03 (work in progress), October 2018.
- [I-D.ietf-core-oscore-groupcomm]
Tiloca, M., Selander, G., Palombini, F., and J. Park, "Group OSCORE - Secure Group Communication for CoAP", draft-ietf-core-oscore-groupcomm-03 (work in progress), October 2018.
- [I-D.mattsson-core-coap-actuators]
Mattsson, J., Fornehed, J., Selander, G., Palombini, F., and C. Amsuess, "Controlling Actuators with CoAP", draft-mattsson-core-coap-actuators-06 (work in progress), September 2018.

- [I-D.mcgrew-iv-gen] McGrew, D., "Generation of Deterministic Initialization Vectors (IVs) and Nonces", draft-mcgrew-iv-gen-03 (work in progress), October 2013.
- [MF00] McGrew, D. and S. Fluhrer, "Attacks on Encryption of Redundant Plaintext and Implications on Internet Security", the Proceedings of the Seventh Annual Workshop on Selected Areas in Cryptography (SAC 2000), Springer-Verlag. , 2000.
- [RFC3552] Rescorla, E. and B. Korver, "Guidelines for Writing RFC Text on Security Considerations", BCP 72, RFC 3552, DOI 10.17487/RFC3552, July 2003, <<https://www.rfc-editor.org/info/rfc3552>>.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <<https://www.rfc-editor.org/info/rfc3986>>.
- [RFC5116] McGrew, D., "An Interface and Algorithms for Authenticated Encryption", RFC 5116, DOI 10.17487/RFC5116, January 2008, <<https://www.rfc-editor.org/info/rfc5116>>.
- [RFC5869] Krawczyk, H. and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)", RFC 5869, DOI 10.17487/RFC5869, May 2010, <<https://www.rfc-editor.org/info/rfc5869>>.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <<https://www.rfc-editor.org/info/rfc6690>>.
- [RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, DOI 10.17487/RFC7228, May 2014, <<https://www.rfc-editor.org/info/rfc7228>>.
- [RFC7515] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", RFC 7515, DOI 10.17487/RFC7515, May 2015, <<https://www.rfc-editor.org/info/rfc7515>>.
- [RFC7967] Bhattacharyya, A., Bandyopadhyay, S., Pal, A., and T. Bose, "Constrained Application Protocol (CoAP) Option for No Server Response", RFC 7967, DOI 10.17487/RFC7967, August 2016, <<https://www.rfc-editor.org/info/rfc7967>>.

Appendix A. Scenario Examples

This section gives examples of OSCORE, targeting scenarios in Section 2.2.1.1 of [I-D.hartke-core-e2e-security-reqs]. The message exchanges are made, based on the assumption that there is a security context established between client and server. For simplicity, these examples only indicate the content of the messages without going into detail of the (compressed) COSE message format.

A.1. Secure Access to Sensor

This example illustrates a client requesting the alarm status from a server.

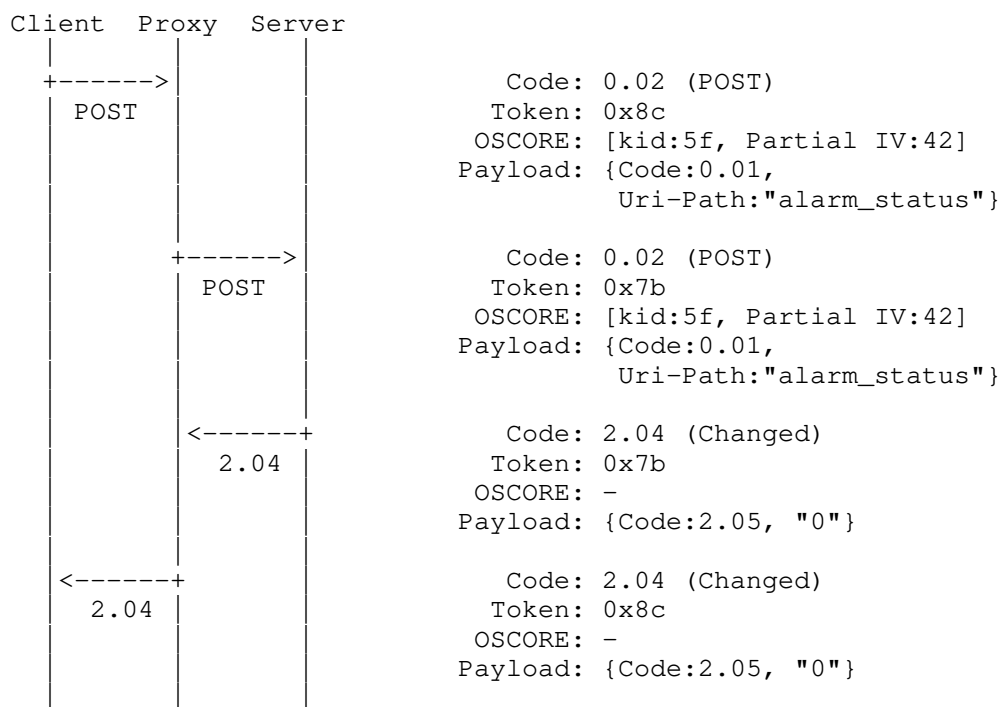


Figure 12: Secure Access to Sensor. Square brackets [...] indicate content of compressed COSE object. Curly brackets { ... } indicate encrypted data.

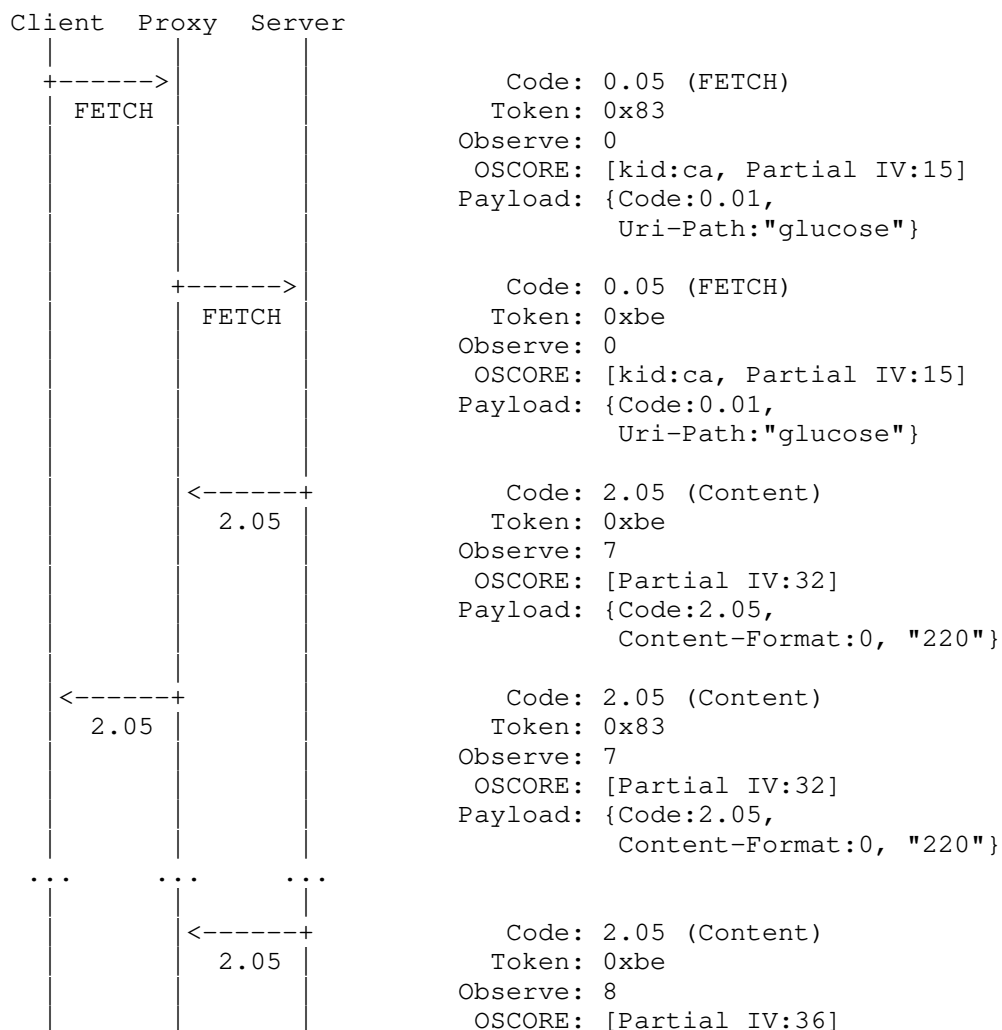
The request/response Codes are encrypted by OSCORE and only dummy Codes (POST/Changed) are visible in the header of the OSCORE message. The option Uri-Path ("alarm_status") and payload ("0") are encrypted.

The COSE header of the request contains an identifier (5f), indicating which security context was used to protect the message and a Partial IV (42).

The server verifies the request as specified in Section 8.2. The client verifies the response as specified in Section 8.4.

A.2. Secure Subscribe to Sensor

This example illustrates a client requesting subscription to a blood sugar measurement resource (GET /glucose), first receiving the value 220 mg/dl and then a second value 180 mg/dl.



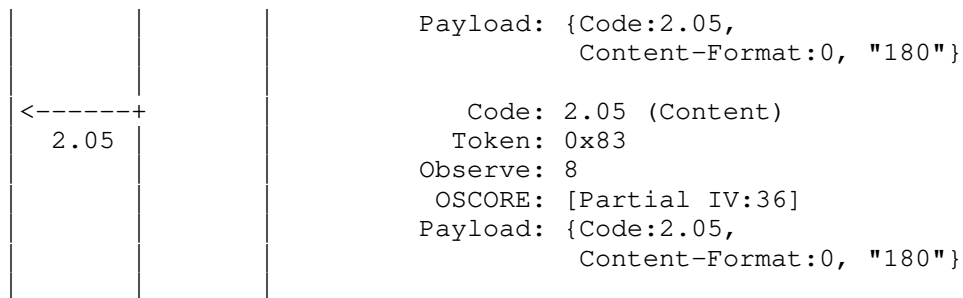


Figure 13: Secure Subscribe to Sensor. Square brackets [...] indicate content of compressed COSE object header. Curly brackets { ... } indicate encrypted data.

The dummy Codes (FETCH/Content) are used to allow forwarding of Observe messages. The options Content-Format (0) and the payload ("220" and "180"), are encrypted.

The COSE header of the request contains an identifier (ca), indicating the security context used to protect the message and a Partial IV (15). The COSE headers of the responses contains Partial IVs (32 and 36).

The server verifies that the Partial IV has not been received before. The client verifies that the responses are bound to the request and that the Partial IVs are greater than any Partial IV previously received in a response bound to the request.

Appendix B. Deployment Examples

For many IoT deployments, a 128 bit uniformly random Master Key is sufficient for encrypting all data exchanged with the IoT device throughout its lifetime. Two examples are given in this section. In the first example, the security context is only derived once from the Master Secret. In the second example, security contexts are derived multiple times using random inputs.

B.1. Security Context Derived Once

An application that only derives the security context once needs to handle the loss of mutable security context parameters, e.g. due to reboot.

B.1.1.1. Sender Sequence Number

In order to handle loss of Sender Sequence Numbers, the device may implement procedures for writing to non-volatile memory during normal operations and updating the security context after reboot, provided that the procedures comply with the requirements on the security context parameters (Section 3.3). This section gives an example of such a procedure.

There are known issues related to writing to non-volatile memory. For example, flash drives may have a limited number of erase operations during its life time. Also, the time for a write operation to non-volatile memory to be completed may be unpredictable, e.g. due to caching, which could result in important security context data not being stored at the time when the device reboots.

However, many devices have predictable limits for writing to non-volatile memory, are physically limited to only send a small amount of messages per minute, and may have no good source of randomness.

To prevent reuse of Sender Sequence Numbers (SSN), an endpoint may perform the following procedure during normal operations:

- o Before using a Sender Sequence Number that is evenly divisible by K, where K is a positive integer, store the Sender Sequence Number (SSN1) in non-volatile memory. After boot, the endpoint initiates the new Sender Sequence Number (SSN2) to the value stored in persistent memory plus K plus F: $SSN2 = SSN1 + K + F$, where F is a positive integer.
 - * Writing to non-volatile memory can be costly; the value K gives a trade-off between frequency of storage operations and efficient use of Sender Sequence Numbers.
 - * Writing to non-volatile memory may be subject to delays, or failure; F MUST be set so that the last Sender Sequence Number used before reboot is never larger than SSN2.

If F cannot be set so SSN2 is always larger than the last Sender Sequence Number used before reboot, the method described in this section MUST NOT be used.

B.1.1.2. Replay Window

In case of loss of security context on the server, to prevent accepting replay of previously received requests, the server may perform the following procedure after boot:

- o The server updates its Sender Sequence Number as specified in Appendix B.1.1, to be used as Partial IV in the response containing the Echo option (next bullet).
- o For each stored security context, the first time after boot the server receives an OSCORE request, the server responds with an OSCORE protected 4.01 (Unauthorized), containing only the Echo option [I-D.ietf-core-echo-request-tag] and no diagnostic payload. The server MUST use its Partial IV when generating the AEAD nonce and MUST include the Partial IV in the response (see Section 5). If the server with use of the Echo option can verify a second OSCORE request as fresh, then the Partial IV of the second request is set as the lower limit of the replay window of that security context.

B.1.3. Notifications

To prevent accepting replay of previously received notifications, the client may perform the following procedure after boot:

- o The client forgets about earlier registrations, removes all Notification Numbers and registers using Observe.

B.2. Security Context Derived Multiple Times

An application which does not require forward secrecy may allow multiple security contexts to be derived from one Master Secret. The requirements on the security context parameters MUST be fulfilled (Section 3.3) even if the client or server is rebooted, recommissioned or in error cases.

This section gives an example of a protocol which adds randomness to the ID Context parameter and uses that together with input parameters pre-established between client and server, in particular Master Secret, Master Salt, and Sender/Recipient ID (see Section 3.2), to derive new security contexts. The random input is transported between client and server in the 'kid context' parameter. This protocol MUST NOT be used unless both endpoints have good sources of randomness.

During normal requests the ID Context of an established security context may be sent in the 'kid context' which, together with 'kid', facilitates for the server to locate a security context. Alternatively, the 'kid context' may be omitted since the ID Context is expected to be known to both client and server, see Section 5.1.

The protocol described in this section may only be needed when the mutable part of security context is lost in the client or server,

e.g. when the endpoint has rebooted. The protocol may additionally be used whenever the client and server need to derive a new security context. For example, if a device is provisioned with one fixed set of input parameters (including Master Secret, Sender and Recipient Identifiers) then a randomized ID Context ensures that the security context is different for each deployment.

The protocol is described below with reference to Figure 14. The client or the server may initiate the protocol, in the latter case step 1 is omitted.

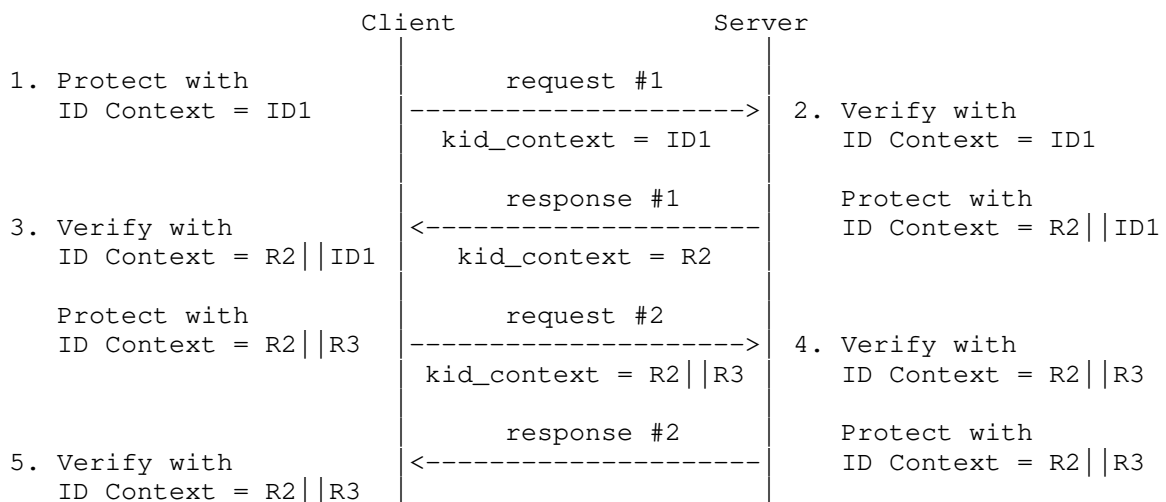


Figure 14: Protocol for establishing a new security context.

1. (Optional) If the client does not have a valid security context with the server, e.g. because of reboot or because this is the first time it contacts the server, then it generates a random string R1, and uses this as ID Context together with the input parameters shared with the server to derive a first security context. The client sends an OSCORE request to the server protected with the first security context, containing R1 wrapped in a CBOR bstr as 'kid context'. The request may target a special resource used for updating security contexts.
2. The server receives an OSCORE request for which it does not have a valid security context, either because the client has generated a new security context ID1 = R1, or because the server has lost part of its security context, e.g. ID Context, Sender Sequence Number or replay window. If the server is able to verify the request (see Section 8.2) with the new derived first security context using the received ID1 (transported in 'kid context') as

ID Context and the input parameters associated to the received 'kid', then the server generates a random string R2, and derives a second security context with ID Context = ID2 = R2 || ID1. The server sends a 4.01 (Unauthorized) response protected with the second security context, containing R2 wrapped in a CBOR bstr as 'kid context', and caches R2. R2 MUST NOT be reused as that may lead to reuse of key and nonce in response #1. Note that the server may receive several requests #1 associated with one security context, leading to multiple parallel protocol runs. Multiple instances of R2 may need to be cached until one of the protocol runs is completed, see Appendix B.2.1.

3. The client receives a response with 'kid context' containing a CBOR bstr wrapping R2 to an OSCORE request it made with ID Context = ID1. The client derives a second security context using ID Context = ID2 = R2 || ID1. If the client can verify the response (see Section 8.4) using the second security context, then the client makes a request protected with a third security context derived from ID Context = ID3 = R2 || R3, where R3 is a random byte string generated by the client. The request includes R2 || R3 wrapped in a CBOR bstr as 'kid context'.
4. If the server receives a request with 'kid context' containing a CBOR bstr wrapping ID3, where the first part of ID3 is identical to an R2 sent in a previous response #1 which it has not received before, then the server derives a third security context with ID Context = ID3. The server MUST NOT accept replayed request #2 messages. If the server can verify the request (see Section 8.2) with the third security context, then the server marks the third security context to be used with this client and removes all instances of R2 associated to this security context from the cache. This security context replaces the previous security context with the client, and the first and the second security contexts are deleted. The server responds using the same security context as in the request.
5. If the client receives a response to the request with the third security context and the response verifies (see Section 8.4), then the client marks the third security context to be used with this server. This security context replaces the previous security context with the server, and the first and second security contexts are deleted.

If verification fails in any step, the endpoint stops processing that message.

The length of the nonces R1, R2, and R3 is application specific. The application needs to set the length of each nonce such the

probability of its value being repeated is negligible; typically, at least 8 bytes long. Since R2 may be generated as the result of a replayed request #1, the probability for collision of R2s is impacted by the birthday paradox. For example, setting the length of R2 to 8 bytes results in an average collision after 2^{32} response #1 messages, which should not be an issue for a constrained server handling on the order of one request per second.

Request #2 can be an ordinary request. The server performs the action of the request and sends response #2 after having successfully completed the security context related operations in step 4. The client acts on response #2 after having successfully completed step 5.

When sending request #2, the client is assured that the Sender Key (derived with the random value R3) has never been used before. When receiving response #2, the client is assured that the response (protected with a key derived from the random value R3 and the Master Secret) was created by the server in response to request #2.

Similarly, when receiving request #2, the server is assured that the request (protected with a key derived from the random value R2 and the Master Secret) was created by the client in response to response #1. When sending response #2, the server is assured that the Sender Key (derived with the random value R2) has never been used before.

Implementation and denial-of-service considerations are made in Appendix B.2.1 and Appendix B.2.2.

B.2.1. Implementation Considerations

This section add some implemention considerations to the protocol described in the previous section.

The server may only have space for a few security contexts, or only be able to handle a few protocol runs in parallel. The server may legitimately receive multiple request #1 messages using the same non-mutable security context, e.g. due to packet loss. replays of old request #1 messages could be difficult for the server to distinguish from legitimate. The server needs to handle the case when the maximum number of cached R2s is reached. If the server receives a request #1 and is not capable of executing it then it may respond with an unprotected 5.03 (Service Unavailable). The server may clear up state from protocol runs which never complete, e.g. set a timer when caching R2, and remove R2 and the associated security contexts from the cache at timeout. Additionally, state information can be flushed at reboot.

As an alternative to caching R2, the server could generate R2 in such a way that it can be sent (in response #1) and verified (at reception of request #2) as the value of R2 it had generated. Such a procedure MUST NOT lead to the server accepting replayed request #2 messages. One construction described in the following is based on using a secret random HMAC key K_HMAC per set of non-mutable security context parameters associated to a client. This construction allows the server to handle verification of R2 in response #2 at the cost of storing the K_HMAC keys and a slightly larger message overhead in response #1. Steps below refer to modifications to Appendix B.2:

- o In step 2, R2 is generated in the following way. First, the server generates a random K_HMAC (unless it already has one associated with the security context), then it sets $R2 = S2 \parallel \text{HMAC}(K_HMAC, S2)$ where S2 is a random byte string, and the HMAC is truncated to 8 bytes. K_HMAC may have an expiration time, after which it is erased. Note that neither R2, S2 nor the derived first and second security contexts need to be cached.
- o In step 4, instead of verifying that R2 coincides with a cached value, the server looks up the associated K_HMAC and verifies the truncated HMAC, and the processing continues accordingly depending on verification success or failure. K_HMAC is used until a run of the protocol is completed (after verification of request #2), or until it expires (whatever comes first), after which K_HMAC is erased. (The latter corresponds to removing the cached values of R2 in step 4 of Appendix B.2, and makes the server reject replays of request #2.)

The length of S2 is application specific and the probability for collision of S2s is impacted by the birthday paradox. For example, setting the length of S2 to 8 bytes results in an average collision after 2^{32} response #1 messages, which should not be an issue for a constrained server handling on the order of one request per second.

Two endpoints sharing a security context may accidentally initiate two instances of the protocol at the same time, each in the role of client, e.g. after a power outage affecting both endpoints. Such a race condition could potentially lead to both protocols failing, and both endpoints repeatedly re-initiating the protocol without converging. Both endpoints can detect this situation and it can be handled in different ways. The requests could potentially be more spread out in time, for example by only initiating this protocol when the endpoint actually needs to make a request, potentially adding a random delay before requests immediately after reboot or if such parallel protocol runs are detected.

B.2.2. Attack Considerations

An on-path attacker may inject a message causing the endpoint to process verification of the message. A message crafted without access to the Master Secret will fail to verify.

Replaying an old request with a value of 'kid_context' which the server does not recognize could trigger the protocol. This causes the server to generate the first and second security context and send a response. But if the client did not expect a response it will be discarded. This may still result in a denial-of-service attack against the server e.g. because of not being able to manage the state associated with many parallel protocol runs, and it may prevent legitimate client requests. Implementation alternatives with less data caching per request #1 message are favorable in this respect, see Appendix B.2.1.

Replaying response #1 in response to some request other than request #1 will fail to verify, since response #1 is associated to request #1, through the dependencies of ID Contexts and the Partial IV of request #1 included in the external_aad of response #1.

If request #2 has already been well received, then the server has a valid security context, so a replay of request #2 is handled by the normal replay protection mechanism. Similarly if response #2 has already been received, a replay of response #2 to some other request from the client will fail by the normal verification of binding of response to request.

Appendix C. Test Vectors

This appendix includes the test vectors for different examples of CoAP messages using OSCORE. Given a set of inputs, OSCORE defines how to set up the Security Context in both the client and the server.

Note that in Appendix C.4 and all following test vectors the Token and the Message ID of the OSCORE-protected CoAP messages are set to the same value of the unprotected CoAP message, to help the reader with comparisons.

[NOTE: the following examples use option number = 9 (TBD1 assigned by IANA). If that differs, the RFC editor is asked to update the test vectors with data provided by the authors. Please remove this paragraph before publication.]

C.1. Test Vector 1: Key Derivation with Master Salt

In this test vector, a Master Salt of 8 bytes is used. The default values are used for AEAD Algorithm and HKDF.

C.1.1. Client

Inputs:

- o Master Secret: 0x0102030405060708090a0b0c0d0e0f10 (16 bytes)
- o Master Salt: 0x9e7ca92223786340 (8 bytes)
- o Sender ID: 0x (0 byte)
- o Recipient ID: 0x01 (1 byte)

From the previous parameters,

- o info (for Sender Key): 0x8540f60a634b657910 (9 bytes)
- o info (for Recipient Key): 0x854101f60a634b657910 (10 bytes)
- o info (for Common IV): 0x8540f60a6249560d (8 bytes)

Outputs:

- o Sender Key: 0xf0910ed7295e6ad4b54fc793154302ff (16 bytes)
- o Recipient Key: 0xffb14e093c94c9cac9471648b4f98710 (16 bytes)
- o Common IV: 0x4622d4dd6d944168eefb54987c (13 bytes)

From the previous parameters and a Partial IV equal to 0 (both for sender and recipient):

- o sender nonce: 0x4622d4dd6d944168eefb54987c (13 bytes)
- o recipient nonce: 0x4722d4dd6d944169eefb54987c (13 bytes)

C.1.2. Server

Inputs:

- o Master Secret: 0x0102030405060708090a0b0c0d0e0f10 (16 bytes)
- o Master Salt: 0x9e7ca92223786340 (8 bytes)

- o Sender ID: 0x01 (1 byte)
- o Recipient ID: 0x (0 byte)

From the previous parameters,

- o info (for Sender Key): 0x854101f60a634b657910 (10 bytes)
- o info (for Recipient Key): 0x8540f60a634b657910 (9 bytes)
- o info (for Common IV): 0x8540f60a6249560d (8 bytes)

Outputs:

- o Sender Key: 0xffb14e093c94c9cac9471648b4f98710 (16 bytes)
- o Recipient Key: 0xf0910ed7295e6ad4b54fc793154302ff (16 bytes)
- o Common IV: 0x4622d4dd6d944168eefb54987c (13 bytes)

From the previous parameters and a Partial IV equal to 0 (both for sender and recipient):

- o sender nonce: 0x4722d4dd6d944169eefb54987c (13 bytes)
- o recipient nonce: 0x4622d4dd6d944168eefb54987c (13 bytes)

C.2. Test Vector 2: Key Derivation without Master Salt

In this test vector, the default values are used for AEAD Algorithm, HKDF, and Master Salt.

C.2.1. Client

Inputs:

- o Master Secret: 0x0102030405060708090a0b0c0d0e0f10 (16 bytes)
- o Sender ID: 0x00 (1 byte)
- o Recipient ID: 0x01 (1 byte)

From the previous parameters,

- o info (for Sender Key): 0x854100f60a634b657910 (10 bytes)
- o info (for Recipient Key): 0x854101f60a634b657910 (10 bytes)

- o info (for Common IV): 0x8540f60a6249560d (8 bytes)

Outputs:

- o Sender Key: 0x321b26943253c7ffb6003b0b64d74041 (16 bytes)
- o Recipient Key: 0xe57b5635815177cd679ab4bcec9d7dda (16 bytes)
- o Common IV: 0xbe35ae297d2dace910c52e99f9 (13 bytes)

From the previous parameters and a Partial IV equal to 0 (both for sender and recipient):

- o sender nonce: 0xbf35ae297d2dace910c52e99f9 (13 bytes)
- o recipient nonce: 0xbf35ae297d2dace810c52e99f9 (13 bytes)

C.2.2. Server

Inputs:

- o Master Secret: 0x0102030405060708090a0b0c0d0e0f10 (16 bytes)
- o Sender ID: 0x01 (1 byte)
- o Recipient ID: 0x00 (1 byte)

From the previous parameters,

- o info (for Sender Key): 0x854101f60a634b657910 (10 bytes)
- o info (for Recipient Key): 0x854100f60a634b657910 (10 bytes)
- o info (for Common IV): 0x8540f60a6249560d (8 bytes)

Outputs:

- o Sender Key: 0xe57b5635815177cd679ab4bcec9d7dda (16 bytes)
- o Recipient Key: 0x321b26943253c7ffb6003b0b64d74041 (16 bytes)
- o Common IV: 0xbe35ae297d2dace910c52e99f9 (13 bytes)

From the previous parameters and a Partial IV equal to 0 (both for sender and recipient):

- o sender nonce: 0xbf35ae297d2dace810c52e99f9 (13 bytes)

- o recipient nonce: 0xbf35ae297d2dace910c52e99f9 (13 bytes)

C.3. Test Vector 3: Key Derivation with ID Context

In this test vector, a Master Salt of 8 bytes and a ID Context of 8 bytes are used. The default values are used for AEAD Algorithm and HKDF.

C.3.1. Client

Inputs:

- o Master Secret: 0x0102030405060708090a0b0c0d0e0f10 (16 bytes)
- o Master Salt: 0x9e7ca92223786340 (8 bytes)
- o Sender ID: 0x (0 byte)
- o Recipient ID: 0x01 (1 byte)
- o ID Context: 0x37cbf3210017a2d3 (8 bytes)

From the previous parameters,

- o info (for Sender Key): 0x85404837cbf3210017a2d30a634b657910 (17 bytes)
- o info (for Recipient Key): 0x8541014837cbf3210017a2d30a634b657910 (18 bytes)
- o info (for Common IV): 0x85404837cbf3210017a2d30a6249560d (16 bytes)

Outputs:

- o Sender Key: 0xaf2a1300a5e95788b356336eeecd2b92 (16 bytes)
- o Recipient Key: 0xe39a0c7c77b43f03b4b39ab9a268699f (16 bytes)
- o Common IV: 0x2ca58fb85ff1b81c0b7181b85e (13 bytes)

From the previous parameters and a Partial IV equal to 0 (both for sender and recipient):

- o sender nonce: 0x2ca58fb85ff1b81c0b7181b85e (13 bytes)
- o recipient nonce: 0x2da58fb85ff1b81d0b7181b85e (13 bytes)

C.3.2. Server

Inputs:

- o Master Secret: 0x0102030405060708090a0b0c0d0e0f10 (16 bytes)
- o Master Salt: 0x9e7ca92223786340 (8 bytes)
- o Sender ID: 0x01 (1 byte)
- o Recipient ID: 0x (0 byte)
- o ID Context: 0x37cbf3210017a2d3 (8 bytes)

From the previous parameters,

- o info (for Sender Key): 0x8541014837cbf3210017a2d30a634b657910 (18 bytes)
- o info (for Recipient Key): 0x85404837cbf3210017a2d30a634b657910 (17 bytes)
- o info (for Common IV): 0x85404837cbf3210017a2d30a6249560d (16 bytes)

Outputs:

- o Sender Key: 0xe39a0c7c77b43f03b4b39ab9a268699f (16 bytes)
- o Recipient Key: 0xaf2a1300a5e95788b356336eeecd2b92 (16 bytes)
- o Common IV: 0x2ca58fb85ff1b81c0b7181b85e (13 bytes)

From the previous parameters and a Partial IV equal to 0 (both for sender and recipient):

- o sender nonce: 0x2da58fb85ff1b81d0b7181b85e (13 bytes)
- o recipient nonce: 0x2ca58fb85ff1b81c0b7181b85e (13 bytes)

C.4. Test Vector 4: OSCORE Request, Client

This section contains a test vector for an OSCORE protected CoAP GET request using the security context derived in Appendix C.1. The unprotected request only contains the Uri-Path and Uri-Host options.

Unprotected CoAP request:

0x44015d1f00003974396c6f63616c686f737483747631 (22 bytes)

Common Context:

- o AEAD Algorithm: 10 (AES-CCM-16-64-128)
- o Key Derivation Function: HKDF SHA-256
- o Common IV: 0x4622d4dd6d944168eefb54987c (13 bytes)

Sender Context:

- o Sender ID: 0x (0 byte)
- o Sender Key: 0xf0910ed7295e6ad4b54fc793154302ff (16 bytes)
- o Sender Sequence Number: 20

The following COSE and cryptographic parameters are derived:

- o Partial IV: 0x14 (1 byte)
- o kid: 0x (0 byte)
- o external_aad: 0x8501810a40411440 (8 bytes)
- o AAD: 0x8368456e63727970743040488501810a40411440 (20 bytes)
- o plaintext: 0x01b3747631 (5 bytes)
- o encryption key: 0xf0910ed7295e6ad4b54fc793154302ff (16 bytes)
- o nonce: 0x4622d4dd6d944168eefb549868 (13 bytes)

From the previous parameter, the following is derived:

- o OSCORE option value: 0x0914 (2 bytes)
- o ciphertext: 0x612f1092f1776f1c1668b3825e (13 bytes)

From there:

- o Protected CoAP request (OSCORE message): 0x44025d1f00003974396c6f63616c686f7374620914ff612f1092f1776f1c1668b3825e (35 bytes)

C.5. Test Vector 5: OSCORE Request, Client

This section contains a test vector for an OSCORE protected CoAP GET request using the security context derived in Appendix C.2. The unprotected request only contains the Uri-Path and Uri-Host options.

Unprotected CoAP request:
0x440171c30000b932396c6f63616c686f737483747631 (22 bytes)

Common Context:

- o AEAD Algorithm: 10 (AES-CCM-16-64-128)
- o Key Derivation Function: HKDF SHA-256
- o Common IV: 0xbe35ae297d2dace910c52e99f9 (13 bytes)

Sender Context:

- o Sender ID: 0x00 (1 bytes)
- o Sender Key: 0x321b26943253c7ffb6003b0b64d74041 (16 bytes)
- o Sender Sequence Number: 20

The following COSE and cryptographic parameters are derived:

- o Partial IV: 0x14 (1 byte)
- o kid: 0x00 (1 byte)
- o external_aad: 0x8501810a4100411440 (9 bytes)
- o AAD: 0x8368456e63727970743040498501810a4100411440 (21 bytes)
- o plaintext: 0x01b3747631 (5 bytes)
- o encryption key: 0x321b26943253c7ffb6003b0b64d74041 (16 bytes)
- o nonce: 0xbf35ae297d2dace910c52e99ed (13 bytes)

From the previous parameter, the following is derived:

- o OSCORE option value: 0x091400 (3 bytes)
- o ciphertext: 0x4ed339a5a379b0b8bc731ffffb0 (13 bytes)

From there:

- o Protected CoAP request (OSCORE message): 0x440271c30000b932396c6f63616c686f737463091400ff4ed339a5a379b0b8bc731ffffb0 (36 bytes)

C.6. Test Vector 6: OSCORE Request, Client

This section contains a test vector for an OSCORE protected CoAP GET request for an application that sets the ID Context and requires it to be sent in the request, so 'kid context' is present in the protected message. This test vector uses the security context derived in Appendix C.3. The unprotected request only contains the Uri-Path and Uri-Host options.

Unprotected CoAP request:

0x44012f8eef9bbf7a396c6f63616c686f737483747631 (22 bytes)

Common Context:

- o AEAD Algorithm: 10 (AES-CCM-16-64-128)
- o Key Derivation Function: HKDF SHA-256
- o Common IV: 0x2ca58fb85ff1b81c0b7181b85e (13 bytes)
- o ID Context: 0x37cbf3210017a2d3 (8 bytes)

Sender Context:

- o Sender ID: 0x (0 bytes)
- o Sender Key: 0xaf2a1300a5e95788b356336eeecd2b92 (16 bytes)
- o Sender Sequence Number: 20

The following COSE and cryptographic parameters are derived:

- o Partial IV: 0x14 (1 byte)
- o kid: 0x (0 byte)
- o kid context: 0x37cbf3210017a2d3 (8 bytes)
- o external_aad: 0x8501810a40411440 (8 bytes)
- o AAD: 0x8368456e63727970743040488501810a40411440 (20 bytes)
- o plaintext: 0x01b3747631 (5 bytes)
- o encryption key: 0xaf2a1300a5e95788b356336eeecd2b92 (16 bytes)
- o nonce: 0x2ca58fb85ff1b81c0b7181b84a (13 bytes)

From the previous parameter, the following is derived:

- o OSCORE option value: 0x19140837cbf3210017a2d3 (11 bytes)
- o ciphertext: 0x72cd7273fd331ac45cffbe55c3 (13 bytes)

From there:

- o Protected CoAP request (OSCORE message):
0x44022f8eef9bbf7a396c6f63616c686f73746b19140837cbf3210017a2d3ff
72cd7273fd331ac45cffbe55c3 (44 bytes)

C.7. Test Vector 7: OSCORE Response, Server

This section contains a test vector for an OSCORE protected 2.05 (Content) response to the request in Appendix C.4. The unprotected response has payload "Hello World!" and no options. The protected response does not contain a 'kid' nor a Partial IV. Note that some parameters are derived from the request.

Unprotected CoAP response:
0x64455d1f00003974ff48656c6c6f20576f726c6421 (21 bytes)

Common Context:

- o AEAD Algorithm: 10 (AES-CCM-16-64-128)
- o Key Derivation Function: HKDF SHA-256
- o Common IV: 0x4622d4dd6d944168eefb54987c (13 bytes)

Sender Context:

- o Sender ID: 0x01 (1 byte)
- o Sender Key: 0xffb14e093c94c9cac9471648b4f98710 (16 bytes)
- o Sender Sequence Number: 0

The following COSE and cryptographic parameters are derived:

- o external_aad: 0x8501810a40411440 (8 bytes)
- o AAD: 0x8368456e63727970743040488501810a40411440 (20 bytes)
- o plaintext: 0x45ff48656c6c6f20576f726c6421 (14 bytes)
- o encryption key: 0xffb14e093c94c9cac9471648b4f98710 (16 bytes)

- o nonce: 0x4622d4dd6d944168eeffb549868 (13 bytes)

From the previous parameter, the following is derived:

- o OSCORE option value: 0x (0 bytes)
- o ciphertext: 0xdbaad1e9a7e7b2a813d3c31524378303cdafae119106 (22 bytes)

From there:

- o Protected CoAP response (OSCORE message):
0x64445d1f0000397490ffdbaad1e9a7e7b2a813d3c31524378303cdafae119106
(32 bytes)

C.8. Test Vector 8: OSCORE Response with Partial IV, Server

This section contains a test vector for an OSCORE protected 2.05 (Content) response to the request in Appendix C.4. The unprotected response has payload "Hello World!" and no options. The protected response does not contain a 'kid', but contains a Partial IV. Note that some parameters are derived from the request.

Unprotected CoAP response:
0x64455d1f00003974ff48656c6c6f20576f726c6421 (21 bytes)

Common Context:

- o AEAD Algorithm: 10 (AES-CCM-16-64-128)
- o Key Derivation Function: HKDF SHA-256
- o Common IV: 0x4622d4dd6d944168eeffb54987c (13 bytes)

Sender Context:

- o Sender ID: 0x01 (1 byte)
- o Sender Key: 0xffb14e093c94c9cac9471648b4f98710 (16 bytes)
- o Sender Sequence Number: 0

The following COSE and cryptographic parameters are derived:

- o Partial IV: 0x00 (1 byte)
- o external_aad: 0x8501810a40411440 (8 bytes)

- o AAD: 0x8368456e63727970743040488501810a40411440 (20 bytes)
- o plaintext: 0x45ff48656c6c6f20576f726c6421 (14 bytes)
- o encryption key: 0xffb14e093c94c9cac9471648b4f98710 (16 bytes)
- o nonce: 0x4722d4dd6d944169eefb54987c (13 bytes)

From the previous parameter, the following is derived:

- o OSCORE option value: 0x0100 (2 bytes)
- o ciphertext: 0x4d4c13669384b67354b2b6175ff4b8658c666a6cf88e (22 bytes)

From there:

- o Protected CoAP response (OSCORE message): 0x64445d1f00003974920100ff4d4c13669384b67354b2b6175ff4b8658c666a6cf88e (34 bytes)

Appendix D. Overview of Security Properties

D.1. Threat Model

This section describes the threat model using the terms of [RFC3552].

It is assumed that the endpoints running OSCORE have not themselves been compromised. The attacker is assumed to have control of the CoAP channel over which the endpoints communicate, including intermediary nodes. The attacker is capable of launching any passive or active, on-path or off-path attacks; including eavesdropping, traffic analysis, spoofing, insertion, modification, deletion, delay, replay, man-in-the-middle, and denial-of-service attacks. This means that the attacker can read any CoAP message on the network and undetectably remove, change, or inject forged messages onto the wire.

OSCORE targets the protection of the CoAP request/response layer (Section 2 of [RFC7252]) between the endpoints, including the CoAP Payload, Code, Uri-Path/Uri-Query, and the other Class E option instances (Section 4.1).

OSCORE does not protect the CoAP messaging layer (Section 2 of [RFC7252]) or other lower layers involved in routing and transporting the CoAP requests and responses.

Additionally, OSCORE does not protect Class U option instances (Section 4.1), as these are used to support CoAP forward proxy operations (see Section 5.7.2 of [RFC7252]). The supported proxies

(forwarding, cross-protocol e.g. CoAP to CoAP-mappable protocols such as HTTP) must be able to change certain Class U options (by instruction from the Client), resulting in the CoAP request being redirected to the server. Changes caused by the proxy may result in the request not reaching the server or reaching the wrong server. For cross-protocol proxies, mappings are done on the Outer part of the message so these protocols are essentially used as transport. Manipulation of these options may thus impact whether the protected message reaches or does not reach the destination endpoint.

Attacks on unprotected CoAP message fields generally causes denial-of-service attacks which are out of scope of this document, more details are given in Appendix D.5.

Attacks against the CoAP request-response layer are in scope. OSCORE is intended to protect against eavesdropping, spoofing, insertion, modification, deletion, replay, and man-in-the middle attacks.

OSCORE is susceptible to traffic analysis as discussed later in Appendix D.

D.2. Supporting Proxy Operations

CoAP is designed to work with intermediaries reading and/or changing CoAP message fields to perform supporting operations in constrained environments, e.g. forwarding and cross-protocol translations.

Securing CoAP on transport layer protects the entire message between the endpoints in which case CoAP proxy operations are not possible. In order to enable proxy operations, security on transport layer needs to be terminated at the proxy in which case the CoAP message in its entirety is unprotected in the proxy.

Requirements for CoAP end-to-end security are specified in [I-D.hartke-core-e2e-security-reqs], in particular forwarding is detailed in Section 2.2.1. The client and server are assumed to be honest, while proxies and gateways are only trusted to perform their intended operations.

By working at the CoAP layer, OSCORE enables different CoAP message fields to be protected differently, which allows message fields required for proxy operations to be available to the proxy while message fields intended for the other endpoint remain protected. In the remainder of this section we analyze how OSCORE protects the protected message fields and the consequences of message fields intended for proxy operation being unprotected.

D.3. Protected Message Fields

Protected message fields are included in the Plaintext (Section 5.3) and the Additional Authenticated Data (Section 5.4) of the COSE_Encrypt0 object and encrypted using an AEAD algorithm.

OSCORE depends on a pre-established random Master Secret (Section 12.3) used to derive encryption keys, and a construction for making (key, nonce) pairs unique (Appendix D.4). Assuming this is true, and the keys are used for no more data than indicated in Section 7.2.1, OSCORE should provide the following guarantees:

- o Confidentiality: An attacker should not be able to determine the plaintext contents of a given OSCORE message or determine that different plaintexts are related (Section 5.3).
- o Integrity: An attacker should not be able to craft a new OSCORE message with protected message fields different from an existing OSCORE message which will be accepted by the receiver.
- o Request-response binding: An attacker should not be able to make a client match a response to the wrong request.
- o Non-replayability: An attacker should not be able to cause the receiver to accept a message which it has previously received and accepted.

In the above, the attacker is anyone except the endpoints, e.g. a compromised intermediary. Informally, OSCORE provides these properties by AEAD-protecting the plaintext with a strong key and uniqueness of (key, nonce) pairs. AEAD encryption [RFC5116] provides confidentiality and integrity for the data. Response-request binding is provided by including the 'kid' and Partial IV of the request in the AAD of the response. Non-replayability of requests and notifications is provided by using unique (key, nonce) pairs and a replay protection mechanism (application dependent, see Section 7.4).

OSCORE is susceptible to a variety of traffic analysis attacks based on observing the length and timing of encrypted packets. OSCORE does not provide any specific defenses against this form of attack but the application may use a padding mechanism to prevent an attacker from directly determine the length of the padding. However, information about padding may still be revealed by side-channel attacks observing differences in timing.

D.4. Uniqueness of (key, nonce)

In this section we show that (key, nonce) pairs are unique as long as the requirements in Sections 3.3 and 7.2.1 are followed.

Fix a Common Context (Section 3.1) and an endpoint, called the encrypting endpoint. An endpoint may alternate between client and server roles, but each endpoint always encrypts with the Sender Key of its Sender Context. Sender Keys are (stochastically) unique since they are derived with HKDF using unique Sender IDs, so messages encrypted by different endpoints use different keys. It remains to prove that the nonces used by the fixed endpoint are unique.

Since the Common IV is fixed, the nonces are determined by a Partial IV (PIV) and the Sender ID of the endpoint generating that Partial IV (ID_PIV). The nonce construction (Section 5.2) with the size of the ID_PIV (S) creates unique nonces for different (ID_PIV, PIV) pairs. There are two cases:

A. For requests, and responses with Partial IV (e.g. Observe notifications):

- o ID_PIV = Sender ID of the encrypting endpoint
- o PIV = current Partial IV of the encrypting endpoint

Since the encrypting endpoint steps the Partial IV for each use, the nonces used in case A are all unique as long as the number of encrypted messages is kept within the required range (Section 7.2.1).

B. For responses without Partial IV (e.g. single response to a request):

- o ID_PIV = Sender ID of the endpoint generating the request
- o PIV = Partial IV of the request

Since the Sender IDs are unique, ID_PIV is different from the Sender ID of the encrypting endpoint. Therefore, the nonces in case B are different compared to nonces in case A, where the encrypting endpoint generated the Partial IV. Since the Partial IV of the request is verified for replay (Section 7.4) associated to this Recipient Context, PIV is unique for this ID_PIV, which makes all nonces in case B distinct.

D.5. Unprotected Message Fields

This sections analyses attacks on message fields which are not protected by OSCORE according to the threat model Appendix D.1.

D.5.1. CoAP Header Fields

- o Version. The CoAP version [RFC7252] is not expected to be sensitive to disclose. Currently there is only one CoAP version defined. A change of this parameter is potentially a denial-of-service attack. Future versions of CoAP need to analyze attacks to OSCORE protected messages due to an adversary changing the CoAP version.
- o Token/Token Length. The Token field is a client-local identifier for differentiating between concurrent requests [RFC7252]. CoAP proxies are allowed to read and change Token and Token Length between hops. An eavesdropper reading the Token can match requests to responses which can be used in traffic analysis. In particular this is true for notifications, where multiple responses are matched with one request. Modifications of Token and Token Length by an on-path attacker may become a denial-of-service attack, since it may prevent the client to identify to which request the response belongs or to find the correct information to verify integrity of the response.
- o Code. The Outer CoAP Code of an OSCORE message is POST or FETCH for requests with corresponding response codes. An endpoint receiving the message discards the Outer CoAP Code and uses the Inner CoAP Code instead (see Section 4.2). Hence, modifications from attackers to the Outer Code do not impact the receiving endpoint. However, changing the Outer Code from FETCH to a Code value for a method that does not work with Observe (such as POST) may, depending on proxy implementation since Observe is undefined for several Codes, cause the proxy to not forward notifications, which is a denial-of-service attack. The use of FETCH rather than POST reveals no more than what is revealed by the presence of the Outer Observe option.
- o Type/Message ID. The Type/Message ID fields [RFC7252] reveal information about the UDP transport binding, e.g. an eavesdropper reading the Type or Message ID gain information about how UDP messages are related to each other. CoAP proxies are allowed to change Type and Message ID. These message fields are not present in CoAP over TCP [RFC8323], and does not impact the request/response message. A change of these fields in a UDP hop is a denial-of-service attack. By sending an ACK, an attacker can make the endpoint believe that it does not need to retransmit the

previous message. By sending a RST, an attacker may be able to cancel an observation. By changing a NON to a CON, the attacker can cause the receiving endpoint to ACK messages for which no ACK was requested.

- o Length. This field contains the length of the message [RFC8323] which may be used for traffic analysis. These message fields are not present in CoAP over UDP, and does not impact the request/response message. A change of Length is a denial-of-service attack similar to changing TCP header fields.

D.5.2. CoAP Options

- o Max-Age. The Outer Max-Age is set to zero to avoid unnecessary caching of OSCORE error responses. Changing this value thus may cause unnecessary caching. No additional information is carried with this option.
- o Proxy-Uri/Proxy-Scheme. These options are used in CoAP forward proxy deployments. With OSCORE, the Proxy-Uri option does not contain the Uri-Path/Uri-Query parts of the URI. The other parts of Proxy-Uri cannot be protected because forward proxies need to change them in order to perform their functions. The server can verify what scheme is used in the last hop, but not what was requested by the client or what was used in previous hops.
- o Uri-Host/Uri-Port. In forward proxy deployments, the Uri-Host/Uri-Port may be changed by an adversary, and the application needs to handle the consequences of that (see Section 4.1.3.2). The Uri-Host may either be omitted, reveal information equivalent to that of the IP address or more privacy-sensitive information, which is discouraged.
- o Observe. The Outer Observe option is intended for a proxy to support forwarding of Observe messages, but is ignored by the endpoints since the Inner Observe determines the processing in the endpoints. Since the Partial IV provides absolute ordering of notifications it is not possible for an intermediary to spoof reordering (see Section 4.1.3.5). The absence of Partial IV, since only allowed for the first notification, does not prevent correct ordering of notifications. The size and distributions of notifications over time may reveal information about the content or nature of the notifications. Cancellations (Section 4.1.3.5.1) are not bound to the corresponding registrations in the same way responses are bound to requests in OSCORE (see Appendix D.3), but that does not open up for attacks based on mismatched cancellations, since for cancellations to be accepted, all options

in the decrypted message except for ETag Options MUST be the same (see Section 4.1.3.5).

- o Block1/Block2/Size1/Size2. The Outer Block options enables fragmentation of OSCORE messages in addition to segmentation performed by the Inner Block options. The presence of these options indicates a large message being sent and the message size can be estimated and used for traffic analysis. Manipulating these options is a potential denial-of-service attack, e.g. injection of alleged Block fragments. The specification of a maximum size of message, MAX_UNFRAGMENTED_SIZE (Section 4.1.3.4.2), above which messages will be dropped, is intended as one measure to mitigate this kind of attack.
- o No-Response. The Outer No-Response option is used to support proxy functionality, specifically to avoid error transmissions from proxies to clients, and to avoid bandwidth reduction to servers by proxies applying congestion control when not receiving responses. Modifying or introducing this option is a potential denial-of-service attack against the proxy operations, but since the option has an Inner value its use can be securely agreed between the endpoints. The presence of this option is not expected to reveal any sensitive information about the message exchange.
- o OSCORE. The OSCORE option contains information about the compressed COSE header. Changing this field may cause OSCORE verification to fail.

D.5.3. Error and Signaling Messages

Error messages occurring during CoAP processing are protected end-to-end. Error messages occurring during OSCORE processing are not always possible to protect, e.g. if the receiving endpoint cannot locate the right security context. For this setting, unprotected error messages are allowed as specified to prevent extensive retransmissions. Those error messages can be spoofed or manipulated, which is a potential denial-of-service attack.

This document specifies OPTIONAL error codes and specific diagnostic payloads for OSCORE processing error messages. Such messages might reveal information about how many and which security contexts exist on the server. Servers MAY want to omit the diagnostic payload of error messages, use the same error code for all errors, or avoid responding altogether in case of OSCORE processing errors, if that is a security concern for the application. Moreover, clients MUST NOT rely on the error code or the diagnostic payload to trigger specific

actions, as these errors are unprotected and can be spoofed or manipulated.

Signaling messages used in CoAP over TCP [RFC8323] are intended to be hop-by-hop; spoofing signaling messages can be used as a denial-of-service attack of a TCP connection.

D.5.4. HTTP Message Fields

In contrast to CoAP, where OSCORE does not protect header fields to enable CoAP-CoAP proxy operations, the use of OSCORE with HTTP is restricted to transporting a protected CoAP message over an HTTP hop. Any unprotected HTTP message fields may reveal information about the transport of the OSCORE message and enable various denial-of-service attacks. It is RECOMMENDED to additionally use TLS [RFC8446] for HTTP hops, which enables encryption and integrity protection of headers, but still leaves some information for traffic analysis.

Appendix E. CDDL Summary

Data structure definitions in the present specification employ the CDDL language for conciseness and precision. CDDL is defined in [I-D.ietf-cbor-cddl], which at the time of writing this appendix is in the process of completion. As the document is not yet available for a normative reference, the present appendix defines the small subset of CDDL that is being used in the present specification.

Within the subset being used here, a CDDL rule is of the form "name = type", where "name" is the name given to the "type". A "type" can be one of:

- o a reference to another named type, by giving its name. The predefined named types used in the present specification are: "uint", an unsigned integer (as represented in CBOR by major type 0); "int", an unsigned or negative integer (as represented in CBOR by major type 0 or 1); "bstr", a byte string (as represented in CBOR by major type 2); "tstr", a text string (as represented in CBOR by major type 3);
- o a choice between two types, by giving both types separated by a "/";
- o an array type (as represented in CBOR by major type 4), where the sequence of elements of the array is described by giving a sequence of entries separated by commas ",", and this sequence is enclosed by square brackets "[" and "]". Arrays described by an array description contain elements that correspond one-to-one to the sequence of entries given. Each entry of an array description

is of the form "name : type", where "name" is the name given to the entry and "type" is the type of the array element corresponding to this entry.

Acknowledgments

The following individuals provided input to this document: Christian Amsuess, Tobias Andersson, Carsten Bormann, Joakim Brorsson, Ben Campbell, Esko Dijk, Jaro Fietz, Thomas Fossati, Martin Gunnarsson, Klaus Hartke, Mirja Kuehlewind, Kathleen Moriarty, Eric Rescorla, Michael Richardson, Adam Roach, Jim Schaad, Peter van der Stok, Dave Thaler, Martin Thomson, Marco Tiloca, William Vignat, and Malisa Vucinic.

Ludwig Seitz and Goeran Selander worked on this document as part of the CelticPlus project CyberWI, with funding from Vinnova.

Authors' Addresses

Goeran Selander
Ericsson AB

Email: goran.selander@ericsson.com

John Mattsson
Ericsson AB

Email: john.mattsson@ericsson.com

Francesca Palombini
Ericsson AB

Email: francesca.palombini@ericsson.com

Ludwig Seitz
RISE SICS

Email: ludwig.seitz@ri.se

CoRE Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 6, 2020

M. Tiloca
RISE AB
G. Selander
F. Palombini
Ericsson AB
J. Park
Universitaet Duisburg-Essen
July 05, 2019

Group OSCORE - Secure Group Communication for CoAP
draft-ietf-core-oscore-groupcomm-05

Abstract

This document describes a mode for protecting group communication over the Constrained Application Protocol (CoAP). The proposed mode relies on Object Security for Constrained RESTful Environments (OSCORE) and the CBOR Object Signing and Encryption (COSE) format. In particular, it defines how OSCORE is used in a group communication setting, while fulfilling the same security requirements for group requests and responses. Source authentication of all messages exchanged within the group is provided by means of digital signatures produced by the sender and embedded in the protected CoAP messages.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on January 6, 2020.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

| | |
|---|----|
| 1. Introduction | 3 |
| 1.1. Terminology | 4 |
| 2. OSCORE Security Context | 5 |
| 2.1. Management of Group Keying Material | 8 |
| 2.2. Wrap-Around of Partial IVs | 9 |
| 3. The COSE Object | 10 |
| 3.1. Updated external_aad | 10 |
| 3.1.1. Updated external_aad for Encryption | 10 |
| 3.1.2. Updated external_aad for Signing | 11 |
| 3.2. Use of the 'kid' Parameter | 12 |
| 3.3. Updated 'unprotected' Field | 12 |
| 4. OSCORE Header Compression | 12 |
| 4.1. Encoding of the OSCORE Option Value | 12 |
| 4.2. Encoding of the OSCORE Payload | 13 |
| 4.3. Examples of Compressed COSE Objects | 14 |
| 5. Message Binding, Sequence Numbers, Freshness and Replay Protection | 15 |
| 5.1. Synchronization of Sender Sequence Numbers | 15 |
| 6. Message Processing | 16 |
| 6.1. Protecting the Request | 16 |
| 6.2. Verifying the Request | 17 |
| 6.3. Protecting the Response | 17 |
| 6.4. Verifying the Response | 18 |
| 7. Responsibilities of the Group Manager | 18 |
| 8. Security Considerations | 19 |
| 8.1. Group-level Security | 20 |
| 8.2. Uniqueness of (key, nonce) | 21 |
| 8.3. Management of Group Keying Material | 21 |
| 8.4. Update of Security Context and Key Rotation | 22 |
| 8.5. Collision of Group Identifiers | 22 |
| 8.6. Cross-group Message Injection | 23 |
| 8.7. End-to-end Protection | 24 |
| 8.8. Security Context Establishment | 24 |
| 8.9. Master Secret | 25 |
| 8.10. Replay Protection | 25 |
| 8.11. Client Aliveness | 26 |

| | |
|--|----|
| 8.12. Cryptographic Considerations | 26 |
| 8.13. Message Segmentation | 26 |
| 8.14. Privacy Considerations | 26 |
| 9. IANA Considerations | 27 |
| 9.1. Counter Signature Parameters Registry | 27 |
| 9.2. Counter Signature Key Parameters Registry | 29 |
| 9.3. Expert Review Instructions | 32 |
| 10. References | 33 |
| 10.1. Normative References | 33 |
| 10.2. Informative References | 34 |
| Appendix A. Assumptions and Security Objectives | 35 |
| A.1. Assumptions | 35 |
| A.2. Security Objectives | 37 |
| Appendix B. List of Use Cases | 37 |
| Appendix C. Example of Group Identifier Format | 40 |
| Appendix D. Set-up of New Endpoints | 41 |
| Appendix E. Examples of Synchronization Approaches | 41 |
| E.1. Best-Effort Synchronization | 41 |
| E.2. Baseline Synchronization | 42 |
| E.3. Challenge-Response Synchronization | 42 |
| Appendix F. No Verification of Signatures | 44 |
| Appendix G. Document Updates | 44 |
| G.1. Version -04 to -05 | 44 |
| G.2. Version -03 to -04 | 45 |
| G.3. Version -02 to -03 | 46 |
| G.4. Version -01 to -02 | 46 |
| G.5. Version -00 to -01 | 47 |
| Acknowledgments | 48 |
| Authors' Addresses | 48 |

1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] is a web transfer protocol specifically designed for constrained devices and networks [RFC7228].

Group communication for CoAP [RFC7390][I-D.dijk-core-groupcomm-bis] addresses use cases where deployed devices benefit from a group communication model, for example to reduce latencies, improve performance and reduce bandwidth utilisation. Use cases include lighting control, integrated building control, software and firmware updates, parameter and configuration updates, commissioning of constrained networks, and emergency multicast (see Appendix B). Furthermore, [RFC7390] recognizes the importance to introduce a secure mode for CoAP group communication. This specification defines such a mode.

Object Security for Constrained RESTful Environments

(OSCORE) [I-D.ietf-core-object-security] describes a security protocol based on the exchange of protected CoAP messages. OSCORE builds on CBOR Object Signing and Encryption (COSE) [RFC8152] and provides end-to-end encryption, integrity, replay protection and binding of response to request between a sender and a recipient, also in the presence of intermediaries. To this end, a CoAP message is protected by including its payload (if any), certain options, and header fields in a COSE object, which replaces the authenticated and encrypted fields in the protected message.

This document defines Group OSCORE, providing end-to-end security of CoAP messages exchanged between members of a group, and preserving independence of transport layer. In particular, the described approach defines how OSCORE should be used in a group communication setting, so that end-to-end security is assured in the same way as OSCORE for unicast communication. That is, end-to-end security is provided for CoAP multicast requests sent by a client to the group, and for related CoAP responses sent by multiple servers. Group OSCORE provides source authentication of all CoAP messages exchanged within the group, by means of digital signatures produced through private keys of sender devices and embedded in the protected CoAP messages.

As defined in the latest [I-D.dijk-core-groupcomm-bis], Group OSCORE is the security protocol to use for applications that rely on CoAP group communication. As in OSCORE, it is still possible to simultaneously rely on DTLS [RFC6347] to protect hop-by-hop communication between a sender and a proxy (and vice versa), and between a proxy and a recipient (and vice versa). Note that DTLS cannot be used to secure messages sent over multicast.

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 BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

Readers are expected to be familiar with the terms and concepts described in CoAP [RFC7252] including "endpoint", "client", "server", "sender" and "recipient"; group communication for CoAP [RFC7390] [I-D.dijk-core-groupcomm-bis]; COSE and counter signatures [RFC8152].

Readers are also expected to be familiar with the terms and concepts for protection and processing of CoAP messages through OSCORE, such

as "Security Context" and "Master Secret", defined in [I-D.ietf-core-object-security].

Terminology for constrained environments, such as "constrained device", "constrained-node network", is defined in [RFC7228].

This document refers also to the following terminology.

- o Keying material: data that is necessary to establish and maintain secure communication among endpoints. This includes, for instance, keys and IVs [RFC4949].
- o Group: a set of endpoints that share group keying material and security parameters (Common Context, see Section 2). The term group used in this specification refers thus to a "security group", not to be confused with network/multicast group or application group.
- o Group Manager: entity responsible for a group. Each endpoint in a group communicates securely with the respective Group Manager, which is neither required to be an actual group member nor to take part in the group communication. The full list of responsibilities of the Group Manager is provided in Section 7.
- o Silent server: member of a group that never responds to requests. Note that a silent server can act as a client, the two roles are independent.
- o Group Identifier (Gid): identifier assigned to the group. Group Identifiers should be unique within the set of groups of a given Group Manager, in order to avoid collisions. In case they are not, the considerations in Section 8.5 apply.
- o Group request: CoAP request message sent by a client in the group to all servers in that group.
- o Source authentication: evidence that a received message in the group originated from a specific identified group member. This also provides assurance that the message was not tampered with by anyone, be it a different legitimate group member or an endpoint which is not a group member.

2. OSCORE Security Context

To support group communication secured with OSCORE, each endpoint registered as member of a group maintains a Security Context as defined in Section 3 of [I-D.ietf-core-object-security], extended as defined below. Each endpoint in a group makes use of:

1. one Common Context, shared by all the endpoints in a given group. In particular:
 - * The ID Context parameter contains the Gid of the group, which is used to retrieve the Security Context for processing messages intended to the endpoints of the group (see Section 6). The choice of the Gid is application specific. An example of specific formatting of the Gid is given in Appendix C. The application needs to specify how to handle possible collisions between Gids, see Section 8.5.
 - * A new parameter Counter Signature Algorithm is included. Its value identifies the digital signature algorithm used to compute a counter signature on the COSE object (see Section 4.5 of [RFC8152]) which provides source authentication within the group. Its value is immutable once the Common Context is established. The used Counter Signature Algorithm MUST be selected among the signing ones defined in the COSE Algorithms Registry (see section 16.4 of [RFC8152]). The EdDSA signature algorithm ed25519 [RFC8032] is mandatory to implement. If Elliptic Curve Digital Signature Algorithm (ECDSA) is used, it is RECOMMENDED that implementations implement "deterministic ECDSA" as specified in [RFC6979].
 - * A new parameter Counter Signature Parameters is included. This parameter identifies the parameters associated to the digital signature algorithm specified in the Counter Signature Algorithm. This parameter MAY be empty and is immutable once the Common Context is established. The exact structure of this parameter depends on the value of Counter Signature Algorithm, and is defined in the Counter Signature Parameters Registry (see Section 9.1), where each entry indicates a specified structure of the Counter Signature Parameters.
 - * A new parameter Counter Signature Key Parameters is included. This parameter identifies the parameters associated to the keys used with the digital signature algorithm specified in the Counter Signature Algorithm. This parameter MAY be empty and is immutable once the Common Context is established. The exact structure of this parameter depends on the value of Counter Signature Algorithm, and is defined in the Counter Signature Key Parameters Registry (see Section 9.2), where each entry indicates a specified structure of the Counter Signature Key Parameters.
2. one Sender Context, unless the endpoint is configured exclusively as silent server. The Sender Context is used to secure outgoing messages and is initialized according to Section 3 of

[I-D.ietf-core-object-security], once the endpoint has joined the group. The Sender Context of a given endpoint matches the corresponding Recipient Context in all the endpoints receiving a protected message from that endpoint. Besides, in addition to what is defined in [I-D.ietf-core-object-security], the Sender Context stores also the endpoint's private key.

3. one Recipient Context for each distinct endpoint from which messages are received, used to process incoming messages. The recipient may generate the Recipient Context upon receiving an incoming message from another endpoint in the group for the first time (see Section 6.2 and Section 6.4). Each Recipient Context matches the Sender Context of the endpoint from which protected messages are received. Besides, in addition to what is defined in [I-D.ietf-core-object-security], each Recipient Context stores also the public key of the associated other endpoint from which messages are received. Note that each Recipient Context includes a Replay Window, unless the recipient acts only as client and hence processes only responses as incoming messages.

The table in Figure 1 overviews the new information included in the OSCORE Security Context, with respect to what defined in Section 3 of [I-D.ietf-core-object-security].

| Context portion | New information |
|------------------------|---|
| Common Context | Counter signature algorithm |
| Common Context | Counter signature parameters |
| Sender Context | Endpoint's own private key |
| Each Recipient Context | Public key of the associated other endpoint |

Figure 1: Additions to the OSCORE Security Context

Upon receiving a secure CoAP message, a recipient uses the sender's public key, in order to verify the counter signature of the COSE Object (see Section 3).

If not already stored in the Recipient Context associated to the sender, the recipient retrieves the sender's public key from the Group Manager, which collects public keys upon endpoints' joining the

group, acts as trusted key repository and ensures the correct association between the public key and the identifier of the sender, for instance by means of public key certificates.

Note that a group member can retrieve public keys from the Group Manager and generate the Recipient Context associated to another group member at any point in time, as long as this is done before verifying a received secure CoAP message. The exact configuration is application dependent. For example, an application can configure a group member to retrieve all the required information and to create the Recipient Context exactly upon receiving a message from another group member for the first time. As an alternative, the application can configure a group member to asynchronously retrieve the required information and update its list of Recipient Contexts well before receiving any message, e.g. by Observing [RFC7641] the Group Manager to get updates on the group membership.

It is RECOMMENDED that the Group Manager collects public keys and provides them to group members upon request as described in [I-D.ietf-ace-key-groupcomm-oscore], where the join process is based on the ACE framework for Authentication and Authorization in constrained environments [I-D.ietf-ace-oauth-authz]. Further details about how public keys can be handled and retrieved in the group is out of the scope of this document.

An endpoint receives its own Sender ID from the Group Manager upon joining the group. That Sender ID is valid only within that group, and is unique within the group. An endpoint uses its own Sender ID (together with other data) to generate unique AEAD nonces for outgoing messages, as in [I-D.ietf-core-object-security]. Endpoints which are configured only as silent servers do not have a Sender ID.

The Sender/Recipient Keys and the Common IV are derived according to the same scheme defined in Section 3.2 of [I-D.ietf-core-object-security]. The mandatory-to-implement HKDF and AEAD algorithms for Group OSCORE are the same as in [I-D.ietf-core-object-security].

2.1. Management of Group Keying Material

In order to establish a new Security Context in a group, a new Group Identifier (Gid) for that group and a new value for the Master Secret parameter MUST be distributed. An example of Gid format supporting this operation is provided in Appendix C. Then, each group member re-derives the keying material stored in its own Sender Context and Recipient Contexts as described in Section 2, using the updated Gid.

After a new Gid has been distributed, a same Recipient ID ('kid') should not be considered as a persistent and reliable indicator of the same group member. Such an indication can be actually achieved only by verifying countersignatures of received messages.

As a consequence, group members may end up retaining stale Recipient Contexts, that are no longer useful to verify incoming secure messages. Applications may define policies to delete (long-)unused Recipient Contexts and reduce the impact on storage space.

If the application requires so (see Appendix A.1), it is RECOMMENDED to adopt a group key management scheme, and securely distribute a new value for the Gid and for the Master Secret parameter of the group's Security Context, before a new joining endpoint is added to the group or after a currently present endpoint leaves the group. This is necessary to preserve backward security and forward security in the group, if the application requires it.

The specific approach used to distribute the new Gid and Master Secret parameter to the group is out of the scope of this document. However, it is RECOMMENDED that the Group Manager supports the distribution of the new Gid and Master Secret parameter to the group according to the Group Rekeying Process described in [I-D.ietf-ace-key-groupcomm-oscore].

2.2. Wrap-Around of Partial IVs

An endpoint can eventually experience a wrap-around of its own Sender Sequence Number, which is incremented after sending each new message including a Partial IV. This is the case for all group requests, all Observe notifications [RFC7641] and, optionally, any other response.

When a wrap-around happens, the endpoint MUST NOT transmit further messages including a Partial IV until it has derived a new Sender Context, in order to avoid reusing nonces with the same keys.

Furthermore, the endpoint SHOULD inform the Group Manager, that can take one of the following actions:

- o The Group Manager renews the OSCORE Security Context in the group (see Section 2.1).
- o The Group Manager provides a new Sender ID value to the endpoint that has experienced the wrap-around. Then, the endpoint derives a new Sender Context using the new Sender ID, as described in Section 3.2 of [I-D.ietf-core-object-security].

Either case, same considerations from Section 2.1 hold about possible retaining of stale Recipient Contexts.

3. The COSE Object

Building on Section 5 of [I-D.ietf-core-object-security], this section defines how to use COSE [RFC8152] to wrap and protect data in the original message. OSCORE uses the untagged COSE_Encrypt0 structure with an Authenticated Encryption with Additional Data (AEAD) algorithm. For Group OSCORE, the following modifications apply.

3.1. Updated external_aad

The external_aad in the Additional Authenticated Data (AAD) is extended as follows. In particular, it has one structure used for the encryption process producing the ciphertext, and one structure used for the signing process producing the counter signature.

3.1.1. Updated external_aad for Encryption

The first external_aad structure used for the encryption process producing the ciphertext (see Section 5.3 of [RFC8152]) includes also the counter signature algorithm and related parameters used to sign messages. In particular, compared with Section 5.4 of [I-D.ietf-core-object-security], the 'algorithms' array in the aad_array MUST also include:

- o 'alg_countersign', which contains the Counter Signature Algorithm from the Common Context (see Section 2). This parameter has the value specified in the "Value" field of the Counter Signature Parameters Registry (see Section 9.1) for this counter signature algorithm.

The 'algorithms' array in the aad_array MAY also include:

- o 'par_countersign', which contains the Counter Signature Parameters from the Common Context (see Section 2). This parameter contains the counter signature parameters encoded as specified in the "Parameters" field of the Counter Signature Parameters Registry (see Section 9.1), for the used counter signature algorithm. Note that if the Counter Signature Parameters in the Common Context is empty, 'par_countersign' is not present.
- o 'par_countersign_key', which contains the Counter Signature Key Parameters from the Common Context (see Section 2). This parameter contains the counter signature key parameters encoded as specified in the "Parameters" field of the Counter Signature Key

Parameters Registry (see Section 9.2), for the used counter signature algorithm. Note that if the Counter Signature Key Parameters in the Common Context is empty, 'par_countersign_key' is not present.

Thus, the following external_aad structure is used for the encryption process producing the ciphertext (see Section 5.3 of [RFC8152]).

```
external_aad = bstr .cbor aad_array

aad_array = [
  oscore_version : uint,
  algorithms : [alg_aead : int / tstr ,
                alg_countersign : int / tstr ,
                ? par_countersign : any ,
                ? par_countersign_key : any],
  request_kid : bstr,
  request_piv : bstr,
  options : bstr
]
```

3.1.2. Updated external_aad for Signing

The second external_aad structure used for the signing process producing the counter signature as defined below includes also:

- o the counter signature algorithm and related parameters used to sign messages, encoded as in the external_aad structure defined in Section 3.1.1;
- o the value of the OSCORE Option included in the OSCORE message, encoded as a binary string.

Thus, the following external_aad structure is used for the signing process producing the counter signature, as defined below.

```
external_aad = bstr .cbor aad_array

aad_array = [
  oscore_version : uint,
  algorithms : [alg_aead : int / tstr ,
                alg_countersign : int / tstr ,
                ? par_countersign : any ,
                ? par_countersign_key : any],
  request_kid : bstr,
  request_piv : bstr,
  OSCORE_option: bstr,
  options : bstr
]
```

Note for implementation: this requires the value of the OSCORE option to be fully ready, before starting the signing process.

3.2. Use of the 'kid' Parameter

The value of the 'kid' parameter in the 'unprotected' field of response messages MUST be set to the Sender ID of the endpoint transmitting the message. That is, unlike in [I-D.ietf-core-object-security], the 'kid' parameter is always present in all messages, i.e. both requests and responses.

3.3. Updated 'unprotected' Field

The 'unprotected' field MUST additionally include the following parameter:

- o CounterSignature0 : its value is set to the counter signature of the COSE object, computed by the sender using its own private key as described in Appendix A.2 of [RFC8152]. In particular, the Sig_structure contains the external_aad as defined in Section 3.1.2 and the ciphertext of the COSE_Encrypt0 object as payload.

4. OSCORE Header Compression

The OSCORE compression defined in Section 6 of [I-D.ietf-core-object-security] is used, with the following additions for the encoding of the OSCORE Option and the OSCORE Payload.

4.1. Encoding of the OSCORE Option Value

Analogously to [I-D.ietf-core-object-security], the value of the OSCORE option SHALL contain the OSCORE flag bits, the Partial IV

parameter, the kid context parameter (length and value), and the kid parameter, with the following modifications:

- o The first byte, containing the OSCORE flag bits, has the following encoding modifications:
 - * The fourth least significant bit MUST be set to 1 in every message, to indicate the presence of the 'kid' parameter for all group requests and responses. That is, unlike in [I-D.ietf-core-object-security], the 'kid' parameter is always present in all messages.
 - * The fifth least significant bit MUST be set to 1 for group requests, to indicate the presence of the 'kid context' parameter in the compressed COSE object. The 'kid context' MAY be present in responses if the application requires it. In such a case, the kid context flag MUST be set to 1.

The flag bits are registered in the OSCORE Flag Bits registry specified in Section 13.7 of [I-D.ietf-core-object-security].

- o The 'kid context' value encodes the Group Identifier value (Gid) of the group's Security Context.
- o The remaining bytes in the OSCORE Option value encode the value of the 'kid' parameter, which is always present both in group requests and in responses.

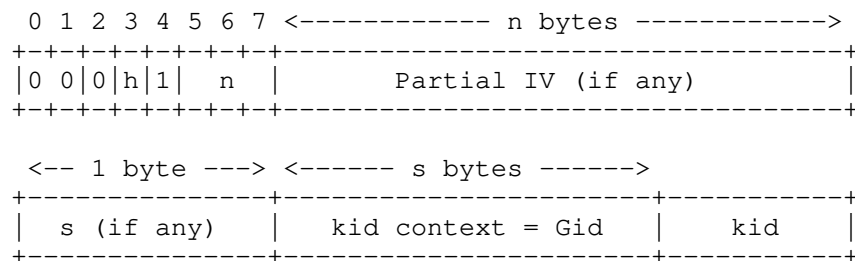


Figure 2: OSCORE Option Value

4.2. Encoding of the OSCORE Payload

The payload of the OSCORE message SHALL encode the ciphertext of the COSE object concatenated with the value of the CounterSignature0 of the COSE object, computed as in Appendix A.2 of [RFC8152] according to the Counter Signature Algorithm and Counter Signature Parameters in the Security Context.

4.3. Examples of Compressed COSE Objects

This section covers a list of OSCORE Header Compression examples for group requests and responses. The examples assume that the COSE_Encrypt0 object is set (which means the CoAP message and cryptographic material is known). Note that the examples do not include the full CoAP unprotected message or the full security context, but only the input necessary to the compression mechanism, i.e. the COSE_Encrypt0 object. The output is the compressed COSE object as defined in Section 4 and divided into two parts, since the object is transported in two CoAP fields: OSCORE option and payload.

The examples assume that the label for the new kid context defined in [I-D.ietf-core-object-security] has value 10. COUNTERSIGN is the CounterSignature0 byte string as described in Section 3 and is 64 bytes long.

1. Request with ciphertext = 0xaea0155667924dff8a24e4cb35b9, kid = 0x25, Partial IV = 5 and kid context = 0x44616c

Before compression (96 bytes):

```
[
  h'',
  { 4:h'25', 6:h'05', 10:h'44616c', 9:COUNTERSIGN },
  h'aea0155667924dff8a24e4cb35b9'
]
```

After compression (85 bytes):

Flag byte: 0b00011001 = 0x19

Option Value: 19 05 03 44 61 6c 25 (7 bytes)

Payload: ae a0 15 56 67 92 4d ff 8a 24 e4 cb 35 b9 COUNTERSIGN
(14 bytes + size of COUNTERSIGN)

1. Response with ciphertext = 60b035059d9ef5667c5a0710823b, kid = 0x52 and no Partial IV.

Before compression (88 bytes):

```
[
  h'',
  { 4:h'52', 9:COUNTERSIGN },
  h'60b035059d9ef5667c5a0710823b'
]
```

After compression (80 bytes):

Flag byte: 0b00001000 = 0x08

Option Value: 08 52 (2 bytes)

Payload: 60 b0 35 05 9d 9e f5 66 7c 5a 07 10 82 3b COUNTERSIGN
(14 bytes + size of COUNTERSIGN)

5. Message Binding, Sequence Numbers, Freshness and Replay Protection

The requirements and properties described in Section 7 of [I-D.ietf-core-object-security] also apply to OSCORE used in group communication. In particular, group OSCORE provides message binding of responses to requests, which provides relative freshness of responses, and replay protection of requests.

Besides, group OSCORE provides additional assurances on the client side, upon receiving responses bound to a same request. That is, as long as the client retains the CoAP Token used in a request (see Section 2.5 of [RFC7390]), group OSCORE ensures that: any possible response sent to that request is not a replay; and at most one response to that request from a given server is accepted, if required by the application.

More details about error processing for replay detection in group OSCORE are specified in Section 6 of this specification. The mechanisms describing replay protection and freshness of Observe notifications do not apply to group OSCORE, as Observe is not defined for group settings.

5.1. Synchronization of Sender Sequence Numbers

Upon joining the group, new servers are not aware of the Sender Sequence Number values currently used by different clients to transmit group requests. This means that, when such servers receive a secure group request from a given client for the first time, they are not able to verify if that request is fresh and has not been replayed or (purposely) delayed. The same holds when a server loses synchronization with Sender Sequence Numbers of clients, for instance after a device reboot.

The exact way to address this issue is application specific, and depends on the particular use case and its synchronization requirements. The list of methods to handle synchronization of Sender Sequence Numbers is part of the group communication policy, and different servers can use different methods.

Appendix E describes three possible approaches that can be considered for synchronization of sequence numbers.

6. Message Processing

Each request message and response message is protected and processed as specified in [I-D.ietf-core-object-security], with the modifications described in the following sections. The following security objectives are fulfilled, as further discussed in Appendix A.2: data replay protection, group-level data confidentiality, source authentication, message integrity.

As per [RFC7252][RFC7390][I-D.dijk-core-groupcomm-bis], group requests sent over multicast MUST be Non-Confirmable. Thus, senders should store their outgoing messages for an amount of time defined by the application and sufficient to correctly handle possible retransmissions. However, this does not prevent the acknowledgment of Confirmable group requests in non-multicast environments. Besides, according to Section 5.2.3 of [RFC7252], responses to Non-Confirmable group requests SHOULD be also Non-Confirmable. However, endpoints MUST be prepared to receive Confirmable responses in reply to a Non-Confirmable group request.

Furthermore, endpoints in the group locally perform error handling and processing of invalid messages according to the same principles adopted in [I-D.ietf-core-object-security]. However, a recipient MUST stop processing and silently reject any message which is malformed and does not follow the format specified in Section 3, or which is not cryptographically validated in a successful way. Either case, it is RECOMMENDED that the recipient does not send back any error message. This prevents servers from replying with multiple error messages to a client sending a group request, so avoiding the risk of flooding and possibly congesting the group.

6.1. Protecting the Request

A client transmits a secure group request as described in Section 8.1 of [I-D.ietf-core-object-security], with the following modifications.

- o In step 2, the 'algorithms' array in the Additional Authenticated Data is modified as described in Section 3.
- o In step 4, the encryption of the COSE object is modified as described in Section 3. The encoding of the compressed COSE object is modified as described in Section 4.
- o In step 5, the counter signature is computed and the format of the OSCORE message is modified as described in Section 4.2. In

particular, the payload of the OSCORE message includes also the counter signature.

6.2. Verifying the Request

Upon receiving a secure group request, a server proceeds as described in Section 8.2 of [I-D.ietf-core-object-security], with the following modifications.

- o In step 2, the decoding of the compressed COSE object follows Section 4. If the received Recipient ID ('kid') does not match with any Recipient Context for the retrieved Gid ('kid context'), then the server creates a new Recipient Context, initializes it according to Section 3 of [I-D.ietf-core-object-security], also retrieving the client's public key.
- o In step 4, the 'algorithms' array in the Additional Authenticated Data is modified as described in Section 3.
- o In step 6, the server also verifies the counter signature using the public key of the client from the associated Recipient Context.
- o Additionally, if the used Recipient Context was created upon receiving this group request and the message is not verified successfully, the server MAY delete that Recipient Context. Such a configuration, which is specified by the application, would prevent attackers from overloading the server's storage and creating processing overhead on the server.

6.3. Protecting the Response

A server that has received a secure group request may reply with a secure response, which is protected as described in Section 8.3 of [I-D.ietf-core-object-security], with the following modifications.

- o In step 2, the 'algorithms' array in the Additional Authenticated Data is modified as described in Section 3.
- o In step 4, the encryption of the COSE object is modified as described in Section 3. The encoding of the compressed COSE object is modified as described in Section 4.
- o In step 5, the counter signature is computed and the format of the OSCORE message is modified as described in Section 4.2. In particular, the payload of the OSCORE message includes also the counter signature.

6.4. Verifying the Response

Upon receiving a secure response message, the client proceeds as described in Section 8.4 of [I-D.ietf-core-object-security], with the following modifications.

- o In step 2, the decoding of the compressed COSE object is modified as described in Section 3. The client also checks whether it previously received a secure response to this request, such that it was successfully verified and it included the same Recipient ID ('kid') of the just received response. If the check yields a positive match and the response is not an Observe notification [RFC7641] (i.e., it does not include an Observe Option), the client SHALL stop processing the response. If the received Recipient ID ('kid') does not match with any Recipient Context for the retrieved Gid ('kid context'), then the client creates a new Recipient Context, initializes it according to Section 3 of [I-D.ietf-core-object-security], also retrieving the server's public key.
- o In step 3, the 'algorithms' array in the Additional Authenticated Data is modified as described in Section 3.
- o In step 5, the client also verifies the counter signature using the public key of the server from the associated Recipient Context. In case of success, the client also records the received Recipient ID ('kid') as included in a successfully verified response to the request.
- o Additionally, if the used Recipient Context was created upon receiving this response and the message is not verified successfully, the client MAY delete that Recipient Context. Such a configuration, which is specified by the application, would prevent attackers from overloading the client's storage and creating processing overhead on the client.

Upon freeing up the Token value of a secure group request for possible reuse [RFC7390][I-D.dijk-core-groupcomm-bis], the client MUST delete the list of recorded Recipient IDs associated to that request (see step 5 above).

7. Responsibilities of the Group Manager

The Group Manager is responsible for performing the following tasks:

1. Creating and managing OSCORE groups. This includes the assignment of a Gid to every newly created group, as well as ensuring uniqueness of Gids within the set of its OSCORE groups.

2. Defining policies for authorizing the joining of its OSCORE groups. Such policies can be enforced locally by the Group Manager, or by a third party in a trust relation with the Group Manager and entrusted to enforce join policies on behalf of the Group Manager.
 3. Driving the join process to add new endpoints as group members.
 4. Establishing Security Common Contexts and providing them to authorized group members during the join process, together with a corresponding Security Sender Context.
 5. Generating and managing Sender IDs within its OSCORE groups, as well as assigning and providing them to new endpoints during the join process. This includes ensuring uniqueness of Sender IDs within each of its OSCORE groups.
 6. Defining a communication policy for each of its OSCORE groups, and signalling it to new endpoints during the join process.
 7. Renewing the Security Context of an OSCORE group upon membership change, by revoking and renewing common security parameters and keying material (rekeying).
 8. Providing the management keying material that a new endpoint requires to participate in the rekeying process, consistent with the key management scheme used in the group joined by the new endpoint.
 9. Updating the Gid of its OSCORE groups, upon renewing the respective Security Context.
 10. Acting as key repository, in order to handle the public keys of the members of its OSCORE groups, and providing such public keys to other members of the same group upon request. The actual storage of public keys may be entrusted to a separate secure storage device.
8. Security Considerations

The same threat model discussed for OSCORE in Appendix D.1 of [I-D.ietf-core-object-security] holds for Group OSCORE. In addition, source authentication of messages is explicitly ensured by means of counter signatures, as further discussed in Section 8.1.

The same considerations on supporting Proxy operations discussed for OSCORE in Appendix D.2 of [I-D.ietf-core-object-security] hold for Group OSCORE.

The same considerations on protected message fields for OSCORE discussed in Appendix D.3 of [I-D.ietf-core-object-security] hold for Group OSCORE.

The same considerations on uniqueness of (key, nonce) pairs for OSCORE discussed in Appendix D.4 of [I-D.ietf-core-object-security] hold for Group OSCORE. This is further discussed in Section 8.2.

The same considerations on unprotected message fields for OSCORE discussed in Appendix D.5 of [I-D.ietf-core-object-security] hold for Group OSCORE, with the following difference. The countersignature included in a Group OSCORE message is computed also over the value of the OSCORE option, which is part of the Additional Authenticated Data used in the signing process. This is further discussed in Section 8.6.

As discussed in Section 6.2.3 of [I-D.dijk-core-groupcomm-bis], Group OSCORE addresses security attacks against CoAP listed in Sections 11.2-11.6 of [RFC7252], especially when mounted over IP multicast.

The rest of this section first discusses security aspects to be taken into account when using Group OSCORE. Then it goes through aspects covered in the security considerations of OSCORE (Section 12 of [I-D.ietf-core-object-security]), and discusses how they hold when Group OSCORE is used.

8.1. Group-level Security

The approach described in this document relies on commonly shared group keying material to protect communication within a group. This has the following implications.

- o Messages are encrypted at a group level (group-level data confidentiality), i.e. they can be decrypted by any member of the group, but not by an external adversary or other external entities.
- o The AEAD algorithm provides only group authentication, i.e. it ensures that a message sent to a group has been sent by a member of that group, but not by the alleged sender. This is why source authentication of messages sent to a group is ensured through a counter signature, which is computed by the sender using its own private key and then appended to the message payload.

Note that, even if an endpoint is authorized to be a group member and to take part in group communications, there is a risk that it behaves inappropriately. For instance, it can forward the content of messages in the group to unauthorized entities. However, in many use

cases, the devices in the group belong to a common authority and are configured by a commissioner (see Appendix B), which results in a practically limited risk and enables a prompt detection/reaction in case of misbehaving.

8.2. Uniqueness of (key, nonce)

The proof for uniqueness of (key, nonce) pairs in Appendix D.4 of [I-D.ietf-core-object-security] is also valid in group communication scenarios. That is, given an OSCORE group:

- o Uniqueness of Sender IDs within the group is enforced by the Group Manager.
- o The case A in Appendix D.4 of [I-D.ietf-core-object-security] concerns all group requests and responses including a Partial IV (e.g. Observe notifications). In this case, same considerations from [I-D.ietf-core-object-security] apply here as well.
- o The case B in Appendix D.4 of [I-D.ietf-core-object-security] concerns responses not including a Partial IV (e.g. single response to a group request). In this case, same considerations from [I-D.ietf-core-object-security] apply here as well.

As a consequence, each message encrypted/decrypted with the same Sender Key is processed by using a different (ID_PIV, PIV) pair. This means that nonces used by any fixed encrypting endpoint are unique. Thus, each message is processed with a different (key, nonce) pair.

8.3. Management of Group Keying Material

The approach described in this specification should take into account the risk of compromise of group members. In particular, this document specifies that a key management scheme for secure revocation and renewal of Security Contexts and group keying material should be adopted.

Especially in dynamic, large-scale, groups where endpoints can join and leave at any time, it is important that the considered group key management scheme is efficient and highly scalable with the group size, in order to limit the impact on performance due to the Security Context and keying material update.

8.4. Update of Security Context and Key Rotation

A group member can receive a message shortly after the group has been rekeyed, and new security parameters and keying material have been distributed by the Group Manager. In the following two cases, this may result in misaligned Security Contexts between the sender and the recipient.

In the first case, the sender protects a message using the old Security Context, i.e. before having installed the new Security Context. However, the recipient receives the message after having installed the new Security Context, hence not being able to correctly process it. A possible way to ameliorate this issue is to preserve the old, recent, Security Context for a maximum amount of time defined by the application. By doing so, the recipient can still try to process the received message using the old retained Security Context as second attempt. Note that a former (compromised) group member can take advantage of this by sending messages protected with the old retained Security Context. Therefore, a conservative application policy should not admit the storage of old Security Contexts.

In the second case, the sender protects a message using the new Security Context, but the recipient receives that request before having installed the new Security Context. Therefore, the recipient would not be able to correctly process the request and hence discards it. If the recipient receives the new Security Context shortly after that and the sender endpoint uses CoAP retransmissions, the former will still be able to receive and correctly process the message. In any case, the recipient should actively ask the Group Manager for an updated Security Context according to an application-defined policy, for instance after a given number of unsuccessfully decrypted incoming messages.

8.5. Collision of Group Identifiers

In case endpoints are deployed in multiple groups managed by different non-synchronized Group Managers, it is possible for Group Identifiers of different groups to coincide. That can also happen if the application can not guarantee unique Group Identifiers within a given Group Manager. However, this does not impair the security of the AEAD algorithm.

In fact, as long as the Master Secret is different for different groups and this condition holds over time, and as long as the Sender IDs within a group are unique, AEAD keys are different among different groups.

8.6. Cross-group Message Injection

A same endpoint is allowed to and would likely use the same signature key in multiple OSCORE groups, possibly administered by different Group Managers. Also, the same endpoint can register several times in the same group, getting multiple unique Sender IDs. This requires that, when a sender endpoint sends a message to an OSCORE group using a Sender ID, the countersignature included in the message is explicitly bound also to that group and to the used Sender ID.

To this end, the countersignature of each message protected with Group OSCORE is computed also over the value of the OSCORE option, which is part of the Additional Authenticated Data used in the signing process (see Section 3.1.2). That is, the countersignature is computed also over: the ID Context (Group ID) and the Partial IV, which are always present in group requests; as well as the Sender ID of the message originator, which is always present in all group requests and responses.

Since the signing process takes as input also the ciphertext of the COSE_Encrypt0 object, the countersignature is bound not only to the intended OSCORE group, hence to the triplet (Master Secret, Master Salt, ID Context), but also to a specific Sender ID in that group and to its specific symmetric key used for AEAD encryption, hence to the quartet (Master Secret, Master Salt, ID Context, Sender ID).

This makes it practically infeasible to perform the attack described below, where a malicious group member injects forged messages to a different OSCORE group than the originally intended one. Let us consider:

- o Two OSCORE groups G1 and G2, with ID Context (Group ID) Gid1 and Gid2, respectively. Both G1 and G2 use the AEAD cipher AES-CCM-16-64-128, i.e. the MAC of the ciphertext is 8 bytes in size.
- o A victim endpoint V which is member of both G1 and G2, and uses the same signature key in both groups. The endpoint V has Sender ID Sid1 in G1 and Sender ID Sid2 in G2. The pairs (Sid1, Gid1) and (Sid2, Gid2) identify the same public key of V in G1 and G2, respectively.
- o A malicious endpoint Z is also member of both G1 and G2. Hence, Z is able to derive the symmetric keys associated to V in G1 and G2.

If countersignatures were not computed also over the value of the OSCORE option as discussed above, Z can intercept a group message M1 sent by V to G1, and forge a valid signed message M2 to be injected in G2, making it appear as sent by V and valid to be accepted.

More in detail, Z first intercepts a message M1 sent by V in G1, and tries to forge a message M2, by changing the value of the OSCORE option from M1 as follows: the 'kid context' is changed from G1 to G2; and the 'kid' is changed from Sid1 to Sid2.

If M2 is used as a request message, there is a probability equal to 2^{-64} that the same unchanged MAC is successfully verified by using Sid2 as 'request_kid' and the symmetric key associated to V in G2. In such a case, the same unchanged signature would be also valid. Note that Z can check offline if a performed forgery is actually valid before sending the forged message to G2. That is, this attack has a complexity of 2^{64} offline calculations.

If M2 is used as a response, Z can also change the response Partial IV, until the same unchanged MAC is successfully verified by using Sid2 as 'request_kid' and the symmetric key associated to V in G2. In such a case, the same unchanged signature would be also valid. Since the Partial IV is 5 bytes in size, this requires 2^{40} operations to test all the Partial IVs, which can be done in real-time. Also, the probability that a single given message M1 can be used to forge a response M2 for a given request is equal to 2^{-24} , since there are more MAC values (8 bytes in size) than Partial IV values (5 bytes in size).

Note that, by changing the Partial IV as discussed above, any member of G1 would also be able to forge a valid signed response message M2 to be injected in G1.

8.7. End-to-end Protection

The same considerations from Section 12.1 of [I-D.ietf-core-object-security] hold for Group OSCORE.

Additionally, (D)TLS and Group OSCORE can be combined for protecting message exchanges occurring over unicast. Instead, it is not possible to combine DTLS and Group OSCORE for protecting message exchanges where messages are (also) sent over multicast.

8.8. Security Context Establishment

The use of COSE_Encrypt0 and AEAD to protect messages as specified in this document requires an endpoint to be a member of an OSCORE group.

That is, upon joining the group, the endpoint securely receives from the Group Manager the necessary input parameters, which are used to derive the Common Security Context and the Sender Context (see Section 2). The Group Manager ensures uniqueness of Sender IDs in the same group.

Each different Recipient Context for decrypting messages from a particular sender can be derived at runtime, at the latest upon receiving a message from that sender for the first time.

Countersignatures of group messages are verified by means of the public key of the respective sender endpoint. Upon nodes' joining, the Group Manager collects such public keys and MUST verify proof-of-possession of the respective private key. Later on, a group member can request from the Group Manager the public keys of other group members.

The joining process can occur, for instance, as defined in [I-D.ietf-ace-key-groupcomm-oscore].

8.9. Master Secret

Group OSCORE derives the Security Context using the same construction of OSCORE, and by using the Group Identifier of a group as the related ID Context. Hence, the same required properties of the Security Context parameters discussed in Section 3.3 of [I-D.ietf-core-object-security] hold for this document.

With particular reference to the OSCORE Master Secret, it has to be kept secret among the members of the respective OSCORE group and the Group Manager responsible for that group. Also, the Master Secret must have a good amount of randomness, and the Group Manager can generate it offline using a good random number generator. This includes the case where the Group Manager rekeys the group by generating and distributing a new Master Secret. Randomness requirements for security are described in [RFC4086].

8.10. Replay Protection

As in OSCORE, also Group OSCORE relies on sender sequence numbers included in the COSE message field 'Partial IV' and used to build AEAD nonces.

As discussed in Section 5.1, an endpoint that has just joined a group is exposed to replay attack, as it is not aware of the sender sequence numbers currently used by other group members. Appendix E describes how endpoints can synchronize with senders' sequence numbers.

Unless exchanges in a group rely only on unicast messages, Group OSCORE cannot be used with reliable transport. Thus, other than in such unlikely case, it cannot be defined that only messages with sequence number which are equal to previous sequence number + 1 are accepted.

The processing of response messages described in Section 6.4 also ensures that a client accepts a single valid response to a given request from each replying server, unless CoAP observation is used.

8.11. Client Aliveness

As discussed in Section 12.5 of [I-D.ietf-core-object-security], a server may use the Echo option [I-D.ietf-core-echo-request-tag] to verify the aliveness of the client that originated a received request. This would also allow the server to (re-)synchronize with the client's sequence number, as well as to ensure that the request is fresh and has not been replayed or (purposely) delayed, if it is the first one received from that client after having joined the group or rebooted (see Appendix E.3).

8.12. Cryptographic Considerations

The same considerations from Section 12.6 of [I-D.ietf-core-object-security] about the maximum Sender Sequence Number hold for Group OSCORE.

As discussed in Section 2.2, an endpoint that experiences a wrap-around of its own Sender Sequence Number MUST NOT transmit further messages including a Partial IV, until it has derived a new Sender Context. This prevents the endpoint to reuse the same AEAD nonces with the same Sender key.

In order to renew its own Sender Context, the endpoint SHOULD inform the Group Manager, which can either renew the whole OSCORE Security Context by means of group rekeying, or provide only that endpoint with a new Sender ID value. Either case, the endpoint derives a new Sender Context, and in particular a new Sender Key.

Additionally, the same considerations from Section 12.6 of [I-D.ietf-core-object-security] hold for Group OSCORE, about building the AEAD nonce and the secrecy of the Security Context parameters.

8.13. Message Segmentation

The same considerations from Section 12.7 of [I-D.ietf-core-object-security] hold for Group OSCORE.

8.14. Privacy Considerations

Group OSCORE ensures end-to-end integrity protection and encryption of the message payload and all options that are not used for proxy operations. In particular, options are processed according to the same class U/I/E that they have for OSCORE. Therefore, the same

privacy considerations from Section 12.8 of [I-D.ietf-core-object-security] hold for Group OSCORE.

Furthermore, the following privacy considerations hold, about the OSCORE option that may reveal information on the communicating endpoints.

- o The 'kid' parameter, which is intended to help a recipient endpoint to find the right Recipient Context, may reveal information about the Sender Endpoint. Since both requests and responses always include the 'kid' parameter, this may reveal information about both a client sending a group request and all the possibly replying servers sending their own individual response.
- o The 'kid context' parameter, which is intended to help a recipient endpoint to find the right Recipient Context, reveals information about the sender endpoint. In particular, it reveals that the sender endpoint is a member of a particular OSCORE group, whose current Group ID is indicated in the 'kid context' parameter. Moreover, this parameter explicitly relate two or more communicating endpoints, as members of the same OSCORE group.

Also, using the mechanisms described in Appendix E.3 to achieve sequence number synchronization with a client may reveal when a server device goes through a reboot. This can be mitigated by the server device storing the precise state of the replay window of each known client on a clean shutdown.

9. IANA Considerations

Note to RFC Editor: Please replace all occurrences of "[This Document]" with the RFC number of this specification and delete this paragraph.

This document has the following actions for IANA.

9.1. Counter Signature Parameters Registry

This specification establishes the IANA "Counter Signature Parameters" Registry. The Registry has been created to use the "Expert Review Required" registration procedure [RFC8126]. Expert review guidelines are provided in Section 9.3.

The columns of this table are:

- o Name: A value that can be used to identify an algorithm in documents for easier comprehension. Its value is taken from the 'Name' column of the "COSE Algorithms" Registry.
- o Value: The value to be used to identify this algorithm. Its content is taken from the 'Value' column of the "COSE Algorithms" Registry. The value MUST be the same one used in the "COSE Algorithms" Registry for the entry with the same 'Name' field.
- o Parameters: This indicates the CBOR encoding of the parameters (if any) for the counter signature algorithm indicated by the 'Value' field.
- o Description: A short description of the parameters encoded in the 'Parameters' field (if any).
- o Reference: This contains a pointer to the public specification for the field, if one exists.

Initial entries in the registry are as follows.

| Name | Value | Parameters | Description | Reference |
|-------|-------|------------|---|-----------------|
| EdDSA | -8 | crv : int | crv value taken from the COSE Elliptic Curve Registry | [This Document] |
| ES256 | -7 | crv : int | crv value taken from the COSE Elliptic Curve Registry | [This Document] |
| ES384 | -35 | crv : int | crv value taken from the COSE Elliptic Curve Registry | [This Document] |
| ES512 | -36 | crv : int | crv value taken from the COSE | [This Document] |

| | | | | |
|--|-----|--|-------------------------|-----------------|
| | | | Elliptic Curve Registry | |
| PS256 | -37 | | Parameters not present | [This Document] |
| PS384 | -38 | | Parameters not present | [This Document] |
| PS512 | -39 | | Parameters not present | [This Document] |
| RSAES-OAEP w/ RFC 8017 default parameters | -40 | | Parameters not present | [This Document] |
| RSAES-OAEP w/ SHA-256 | -41 | | Parameters not present | [This Document] |
| RSAES-OAEP w/ SHA-512 | -42 | | Parameters not present | [This Document] |

9.2. Counter Signature Key Parameters Registry

This specification establishes the IANA "Counter Signature Key Parameters" Registry. The Registry has been created to use the "Expert Review Required" registration procedure [RFC8126]. Expert review guidelines are provided in Section 9.3.

The columns of this table are:

- o **Name:** A value that can be used to identify an algorithm in documents for easier comprehension. Its value is taken from the 'Name' column of the "COSE Algorithms" Registry.
- o **Value:** The value to be used to identify this algorithm. Its content is taken from the 'Value' column of the "COSE Algorithms" Registry. The value **MUST** be the same one used in the "COSE Algorithms" Registry for the entry with the same 'Name' field.
- o **Parameters:** This indicates the CBOR encoding of the key parameters (if any) for the counter signature algorithm indicated by the 'Value' field.
- o **Description:** A short description of the parameters encoded in the 'Parameters' field (if any).
- o **Reference:** This contains a pointer to the public specification for the field, if one exists.

Initial entries in the registry are as follows.

| Name | Value | Parameters | Description | Reference |
|-------|-------|------------|--|-----------------|
| EdDSA | -8 | key : int | key value is 1, as Key Type "OKP" from the COSE Key Types Registry | [This Document] |
| | | crv : int | crv value taken from the COSE Elliptic Curve Registry | |
| ES256 | -7 | key : int | key value is 2, as Key Type "EC2" from the COSE Key Types Registry | [This Document] |
| | | crv : int | crv value taken from the COSE Elliptic Curve Registry | |

| | | | | |
|-------|-----|----------------------------|---|-----------------|
| ES384 | -35 | kty : int crv : int | kty value is 2, as Key Type "EC2" from the COSE Key Types Registry crv value taken from the COSE Elliptic Curve Registry | [This Document] |
| ES512 | -36 | kty : int crv : int | kty value is 2, as Key Type "EC2" from the COSE Key Types Registry crv value taken from the COSE Elliptic Curve Registry | [This Document] |
| PS256 | -37 | kty : int | kty value is 3, as Key Type "RSA" from the COSE Key Types Registry | [This Document] |
| PS384 | -38 | kty : int | kty value is 3, as Key Type "RSA" from the COSE Key Types Registry | [This Document] |
| PS512 | -39 | kty : int | kty value is 3, as Key Type "RSA" from the COSE Key Types Registry | [This Document] |
| | | | | |

| | | | | |
|--|-----|-----------|---|--------------------|
| RSAES-OAEP w/ RFC 8017 default parameters | -40 | key : int | key value is 3, as Key Type "RSA" from the COSE Key Types Registry | [This Document] |
| RSAES-OAEP w/ SHA-256 | -41 | key : int | key value is 3, as Key Type "RSA" from the COSE Key Types Registry | [This Document] |
| RSAES-OAEP w/ SHA-512 | -42 | key : int | key value is 3, as Key Type "RSA" from the COSE Key Types Registry | [This Document] |

9.3. Expert Review Instructions

The IANA Registry established in this document is defined as "Expert Review". This section gives some general guidelines for what the experts should be looking for, but they are being designated as experts for a reason so they should be given substantial latitude.

Expert reviewers should take into consideration the following points:

- o Clarity and correctness of registrations. Experts are expected to check the clarity of purpose and use of the requested entries. Experts should inspect the entry for the algorithm considered, to verify the conformity of the encoding proposed against the theoretical algorithm, including completeness of the 'Parameters' column. Expert needs to make sure values are taken from the right registry, when that's required. Expert should consider requesting an opinion on the correctness of registered parameters from the CBOR Object Signing and Encryption Working Group (COSE). Encodings that do not meet these objective of clarity and completeness should not be registered.
- o Duplicated registration and point squatting should be discouraged. Reviewers are encouraged to get sufficient information for registration requests to ensure that the usage is not going to duplicate one that is already registered and that the point is likely to be used in deployments.

- o Experts should take into account the expected usage of fields when approving point assignment. The length of the 'Parameters' encoding should be weighed against the usage of the entry, considering the size of device it will be used on. Additionally, the length of the encoded value should be weighed against how many code points of that length are left, the size of device it will be used on, and the number of code points left that encode to that size.
- o Specifications are recommended. When specifications are not provided, the description provided needs to have sufficient information to verify the points above.

10. References

10.1. Normative References

- [I-D.dijk-core-groupcomm-bis]
Dijk, E., Wang, C., and M. Tiloca, "Group Communication for the Constrained Application Protocol (CoAP)", draft-dijk-core-groupcomm-bis-00 (work in progress), March 2019.
- [I-D.ietf-core-object-security]
Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security for Constrained RESTful Environments (OSCORE)", draft-ietf-core-object-security-16 (work in progress), March 2019.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC4086] Eastlake 3rd, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, DOI 10.17487/RFC4086, June 2005, <<https://www.rfc-editor.org/info/rfc4086>>.
- [RFC6979] Pornin, T., "Deterministic Usage of the Digital Signature Algorithm (DSA) and Elliptic Curve Digital Signature Algorithm (ECDSA)", RFC 6979, DOI 10.17487/RFC6979, August 2013, <<https://www.rfc-editor.org/info/rfc6979>>.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<https://www.rfc-editor.org/info/rfc7252>>.

- [RFC8032] Josefsson, S. and I. Liusvaara, "Edwards-Curve Digital Signature Algorithm (EdDSA)", RFC 8032, DOI 10.17487/RFC8032, January 2017, <<https://www.rfc-editor.org/info/rfc8032>>.
- [RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <<https://www.rfc-editor.org/info/rfc8126>>.
- [RFC8152] Schaad, J., "CBOR Object Signing and Encryption (COSE)", RFC 8152, DOI 10.17487/RFC8152, July 2017, <<https://www.rfc-editor.org/info/rfc8152>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.

10.2. Informative References

- [I-D.ietf-ace-key-groupcomm-oscore]
Tiloca, M., Park, J., and F. Palombini, "Key Management for OSCORE Groups in ACE", draft-ietf-ace-key-groupcomm-oscore-01 (work in progress), March 2019.
- [I-D.ietf-ace-oauth-authz]
Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "Authentication and Authorization for Constrained Environments (ACE) using the OAuth 2.0 Framework (ACE-OAuth)", draft-ietf-ace-oauth-authz-24 (work in progress), March 2019.
- [I-D.ietf-core-echo-request-tag]
Amsuess, C., Mattsson, J., and G. Selander, "CoAP: Echo, Request-Tag, and Token Processing", draft-ietf-core-echo-request-tag-05 (work in progress), May 2019.
- [I-D.somaraju-ace-multicast]
Somaraju, A., Kumar, S., Tschofenig, H., and W. Werner, "Security for Low-Latency Group Communication", draft-somaraju-ace-multicast-02 (work in progress), October 2016.
- [RFC4944] Montenegro, G., Kushalnagar, N., Hui, J., and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks", RFC 4944, DOI 10.17487/RFC4944, September 2007, <<https://www.rfc-editor.org/info/rfc4944>>.

- [RFC4949] Shirey, R., "Internet Security Glossary, Version 2", FYI 36, RFC 4949, DOI 10.17487/RFC4949, August 2007, <<https://www.rfc-editor.org/info/rfc4949>>.
- [RFC6282] Hui, J., Ed. and P. Thubert, "Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks", RFC 6282, DOI 10.17487/RFC6282, September 2011, <<https://www.rfc-editor.org/info/rfc6282>>.
- [RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <<https://www.rfc-editor.org/info/rfc6347>>.
- [RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, DOI 10.17487/RFC7228, May 2014, <<https://www.rfc-editor.org/info/rfc7228>>.
- [RFC7390] Rahman, A., Ed. and E. Dijk, Ed., "Group Communication for the Constrained Application Protocol (CoAP)", RFC 7390, DOI 10.17487/RFC7390, October 2014, <<https://www.rfc-editor.org/info/rfc7390>>.
- [RFC7641] Hartke, K., "Observing Resources in the Constrained Application Protocol (CoAP)", RFC 7641, DOI 10.17487/RFC7641, September 2015, <<https://www.rfc-editor.org/info/rfc7641>>.

Appendix A. Assumptions and Security Objectives

This section presents a set of assumptions and security objectives for the approach described in this document.

A.1. Assumptions

The following assumptions are assumed to be already addressed and are out of the scope of this document.

- o Multicast communication topology: this document considers both 1-to-N (one sender and multiple recipients) and M-to-N (multiple senders and multiple recipients) communication topologies. The 1-to-N communication topology is the simplest group communication scenario that would serve the needs of a typical low-power and lossy network (LLN). Examples of use cases that benefit from secure group communication are provided in Appendix B.

In a 1-to-N communication model, only a single client transmits data to the group, in the form of request messages; in an M-to-N

communication model (where M and N do not necessarily have the same value), M group members are clients. According to [RFC7390], any possible proxy entity is supposed to know about the clients in the group and to not perform aggregation of response messages from multiple servers. Also, every client expects and is able to handle multiple response messages associated to a same request sent to the group.

- o Group size: security solutions for group communication should be able to adequately support different and possibly large groups. The group size is the current number of members in a group. In the use cases mentioned in this document, the number of clients (normally the controlling devices) is expected to be much smaller than the number of servers (i.e. the controlled devices). A security solution for group communication that supports 1 to 50 clients would be able to properly cover the group sizes required for most use cases that are relevant for this document. The maximum group size is expected to be in the range of 2 to 100 devices. Groups larger than that should be divided into smaller independent groups.
- o Communication with the Group Manager: an endpoint must use a secure dedicated channel when communicating with the Group Manager, also when not registered as group member.
- o Provisioning and management of Security Contexts: an OSCORE Security Context must be established among the group members. A secure mechanism must be used to generate, revoke and (re-)distribute keying material, multicast security policies and security parameters in the group. The actual provisioning and management of the Security Context is out of the scope of this document.
- o Multicast data security ciphersuite: all group members must agree on a ciphersuite to provide authenticity, integrity and confidentiality of messages in the group. The ciphersuite is specified as part of the Security Context.
- o Backward security: a new device joining the group should not have access to any old Security Contexts used before its joining. This ensures that a new group member is not able to decrypt confidential data sent before it has joined the group. The adopted key management scheme should ensure that the Security Context is updated to ensure backward confidentiality. The actual mechanism to update the Security Context and renew the group keying material upon a group member's joining has to be defined as part of the group key management scheme.

- o Forward security: entities that leave the group should not have access to any future Security Contexts or message exchanged within the group after their leaving. This ensures that a former group member is not able to decrypt confidential data sent within the group anymore. Also, it ensures that a former member is not able to send encrypted and/or integrity protected messages to the group anymore. The actual mechanism to update the Security Context and renew the group keying material upon a group member's leaving has to be defined as part of the group key management scheme.

A.2. Security Objectives

The approach described in this document aims at fulfilling the following security objectives:

- o Data replay protection: replayed group request messages or response messages must be detected.
- o Group-level data confidentiality: messages sent within the group shall be encrypted if privacy sensitive data is exchanged within the group. This document considers group-level data confidentiality since messages are encrypted at a group level, i.e. in such a way that they can be decrypted by any member of the group, but not by an external adversary or other external entities.
- o Source authentication: messages sent within the group shall be authenticated. That is, it is essential to ensure that a message is originated by a member of the group in the first place, and in particular by a specific member of the group.
- o Message integrity: messages sent within the group shall be integrity protected. That is, it is essential to ensure that a message has not been tampered with by an external adversary or other external entities which are not group members.
- o Message ordering: it must be possible to determine the ordering of messages coming from a single sender. In accordance with OSCORE [I-D.ietf-core-object-security], this results in providing relative freshness of group requests and absolute freshness of responses. It is not required to determine ordering of messages from different senders.

Appendix B. List of Use Cases

Group Communication for CoAP [RFC7390][I-D.dijk-core-groupcomm-bis] provides the necessary background for multicast-based CoAP communication, with particular reference to low-power and lossy

networks (LLNs) and resource constrained environments. The interested reader is encouraged to first read [RFC7390][I-D.dijk-core-groupcomm-bis] to understand the non-security related details. This section discusses a number of use cases that benefit from secure group communication. Specific security requirements for these use cases are discussed in Appendix A.

- o **Lighting control:** consider a building equipped with IP-connected lighting devices, switches, and border routers. The devices are organized into groups according to their physical location in the building. For instance, lighting devices and switches in a room or corridor can be configured as members of a single group. Switches are then used to control the lighting devices by sending on/off/dimming commands to all lighting devices in a group, while border routers connected to an IP network backbone (which is also multicast-enabled) can be used to interconnect routers in the building. Consequently, this would also enable logical groups to be formed even if devices in the lighting group may be physically in different subnets (e.g. on wired and wireless networks). Connectivity between lighting devices may be realized, for instance, by means of IPv6 and (border) routers supporting 6LoWPAN [RFC4944][RFC6282]. Group communication enables synchronous operation of a group of connected lights, ensuring that the light preset (e.g. dimming level or color) of a large group of luminaires are changed at the same perceived time. This is especially useful for providing a visual synchronicity of light effects to the user. As a practical guideline, events within a 200 ms interval are perceived as simultaneous by humans, which is necessary to ensure in many setups. Devices may reply back to the switches that issue on/off/dimming commands, in order to report about the execution of the requested operation (e.g. OK, failure, error) and their current operational status. In a typical lighting control scenario, a single switch is the only entity responsible for sending commands to a group of lighting devices. In more advanced lighting control use cases, a M-to-N communication topology would be required, for instance in case multiple sensors (presence or day-light) are responsible to trigger events to a group of lighting devices. Especially in professional lighting scenarios, the roles of client and server are configured by the lighting commissioner, and devices strictly follow those roles.
- o **Integrated building control:** enabling Building Automation and Control Systems (BACSs) to control multiple heating, ventilation and air-conditioning units to pre-defined presets. Controlled units can be organized into groups in order to reflect their physical position in the building, e.g. devices in the same room can be configured as members of a single group. As a practical

guideline, events within intervals of seconds are typically acceptable. Controlled units are expected to possibly reply back to the BACS issuing control commands, in order to report about the execution of the requested operation (e.g. OK, failure, error) and their current operational status.

- o Software and firmware updates: software and firmware updates often comprise quite a large amount of data. This can overload a LLN that is otherwise typically used to deal with only small amounts of data, on an infrequent base. Rather than sending software and firmware updates as unicast messages to each individual device, multicasting such updated data to a larger group of devices at once displays a number of benefits. For instance, it can significantly reduce the network load and decrease the overall time latency for propagating this data to all devices. Even if the complete whole update process itself is secured, securing the individual messages is important, in case updates consist of relatively large amounts of data. In fact, checking individual received data piecemeal for tampering avoids that devices store large amounts of partially corrupted data and that they detect tampering hereof only after all data has been received. Devices receiving software and firmware updates are expected to possibly reply back, in order to provide a feedback about the execution of the update operation (e.g. OK, failure, error) and their current operational status.
- o Parameter and configuration update: by means of multicast communication, it is possible to update the settings of a group of similar devices, both simultaneously and efficiently. Possible parameters are related, for instance, to network load management or network access controls. Devices receiving parameter and configuration updates are expected to possibly reply back, to provide a feedback about the execution of the update operation (e.g. OK, failure, error) and their current operational status.
- o Commissioning of LLNs systems: a commissioning device is responsible for querying all devices in the local network or a selected subset of them, in order to discover their presence, and be aware of their capabilities, default configuration, and operating conditions. Queried devices displaying similarities in their capabilities and features, or sharing a common physical location can be configured as members of a single group. Queried devices are expected to reply back to the commissioning device, in order to notify their presence, and provide the requested information and their current operational status.
- o Emergency multicast: a particular emergency related information (e.g. natural disaster) is generated and multicast by an emergency

notifier, and relayed to multiple devices. The latter may reply back to the emergency notifier, in order to provide their feedback and local information related to the ongoing emergency. This kind of setups should additionally rely on a fault tolerance multicast algorithm, such as MPL.

Appendix C. Example of Group Identifier Format

This section provides an example of how the Group Identifier (Gid) can be specifically formatted. That is, the Gid can be composed of two parts, namely a Group Prefix and a Group Epoch.

For each group, the Group Prefix is constant over time and is uniquely defined in the set of all the groups associated to the same Group Manager. The choice of the Group Prefix for a given group's Security Context is application specific. The size of the Group Prefix directly impact on the maximum number of distinct groups under the same Group Manager.

The Group Epoch is set to 0 upon the group's initialization, and is incremented by 1 upon completing each renewal of the Security Context and keying material in the group (see Section 2.1). In particular, once a new Master Secret has been distributed to the group, all the group members increment by 1 the Group Epoch in the Group Identifier of that group.

As an example, a 3-byte Group Identifier can be composed of: i) a 1-byte Group Prefix '0xb1' interpreted as a raw byte string; and ii) a 2-byte Group Epoch interpreted as an unsigned integer ranging from 0 to 65535. Then, after having established the Security Common Context 61532 times in the group, its Group Identifier will assume value '0xb1f05c'.

Using an immutable Group Prefix for a group assumes that enough time elapses between two consecutive usages of the same Group Epoch value in that group. This ensures that the Gid value is temporally unique during the lifetime of a given message. Thus, the expected highest rate for addition/removal of group members and consequent group rekeying should be taken into account for a proper dimensioning of the Group Epoch size.

As discussed in Section 8.5, if endpoints are deployed in multiple groups managed by different non-synchronized Group Managers, it is possible that Group Identifiers of different groups coincide at some point in time. In this case, a recipient has to handle coinciding Group Identifiers, and has to try using different OSCORE Security Contexts to process an incoming message, until the right one is found and the message is correctly verified. Therefore, it is favourable

that Group Identifiers from different Group Managers have a size that result in a small probability of collision. How small this probability should be is up to system designers.

Appendix D. Set-up of New Endpoints

An endpoint joins a group by explicitly interacting with the responsible Group Manager. When becoming members of a group, endpoints are not required to know how many and what endpoints are in the same group.

Communications between a joining endpoint and the Group Manager rely on the CoAP protocol and must be secured. Specific details on how to secure communications between joining endpoints and a Group Manager are out of the scope of this document.

The Group Manager must verify that the joining endpoint is authorized to join the group. To this end, the Group Manager can directly authorize the joining endpoint, or expect it to provide authorization evidence previously obtained from a trusted entity. Further details about the authorization of joining endpoints are out of scope.

In case of successful authorization check, the Group Manager generates a Sender ID assigned to the joining endpoint, before proceeding with the rest of the join process. That is, the Group Manager provides the joining endpoint with the keying material and parameters to initialize the OSCORE Security Context (see Section 2). The actual provisioning of keying material and parameters to the joining endpoint is out of the scope of this document.

It is RECOMMENDED that the join process adopts the approach described in [I-D.ietf-ace-key-groupcomm-oscore] and based on the ACE framework for Authentication and Authorization in constrained environments [I-D.ietf-ace-oauth-authz].

Appendix E. Examples of Synchronization Approaches

This section describes three possible approaches that can be considered by server endpoints to synchronize with sender sequence numbers of client endpoints sending group requests.

E.1. Best-Effort Synchronization

Upon receiving a group request from a client, a server does not take any action to synchronize with the sender sequence number of that client. This provides no assurance at all as to message freshness, which can be acceptable in non-critical use cases.

E.2. Baseline Synchronization

Upon receiving a group request from a given client for the first time, a server initializes its last-seen sender sequence number in its Recipient Context associated to that client. However, the server drops the group request without delivering it to the application layer. This provides a reference point to identify if future group requests from the same client are fresher than the last one received.

A replay time interval exists, between when a possibly replayed or delayed message is originally transmitted by a given client and the first authentic fresh message from that same client is received. This can be acceptable for use cases where servers admit such a trade-off between performance and assurance of message freshness.

E.3. Challenge-Response Synchronization

A server performs a challenge-response exchange with a client, by using the Echo Option for CoAP described in Section 2 of [I-D.ietf-core-echo-request-tag] and according to Section 7.5.2 of [I-D.ietf-core-object-security].

That is, upon receiving a group request from a particular client for the first time, the server processes the message as described in Section 6.2 of this specification, but, even if valid, does not deliver it to the application. Instead, the server replies to the client with a 4.03 Forbidden response message including an Echo Option, and stores the option value included therein.

Upon receiving a 4.03 Forbidden response that includes an Echo Option and originates from a verified group member, a client sends a request as a unicast message addressed to the same server, echoing the Echo Option value. In particular, the client does not necessarily resend the same group request, but can instead send a more recent one, if the application permits it. This makes it possible for the client to not retain previously sent group requests for full retransmission, unless the application explicitly requires otherwise. In either case, the client uses the sender sequence number value currently stored in its own Sender Context. If the client stores group requests for possible retransmission with the Echo Option, it should not store a given request for longer than a pre-configured time interval. Note that the unicast request echoing the Echo Option is correctly treated and processed as a message, since the 'kid context' field including the Group Identifier of the OSCORE group is still present in the OSCORE Option as part of the COSE object (see Section 3).

Upon receiving the unicast request including the Echo Option, the server verifies that the option value equals the stored and previously sent value; otherwise, the request is silently discarded. Then, the server verifies that the unicast request has been received within a pre-configured time interval, as described in [I-D.ietf-core-echo-request-tag]. In such a case, the request is further processed and verified; otherwise, it is silently discarded. Finally, the server updates the Recipient Context associated to that client, by setting the Replay Window according to the Sequence Number from the unicast request conveying the Echo Option. The server either delivers the request to the application if it is an actual retransmission of the original one, or discards it otherwise. Mechanisms to signal whether the resent request is a full retransmission of the original one are out of the scope of this specification.

In case it does not receive a valid unicast request including the Echo Option within the configured time interval, the server endpoint should perform the same challenge-response upon receiving the next group request from that same client.

A server should not deliver group requests from a given client to the application until one valid request from that same client has been verified as fresh, as conveying an echoed Echo Option [I-D.ietf-core-echo-request-tag]. Also, a server may perform the challenge-response described above at any time, if synchronization with sender sequence numbers of clients is (believed to be) lost, for instance after a device reboot. It is the role of the application to define under what circumstances sender sequence numbers lose synchronization. This can include a minimum gap between the sender sequence number of the latest accepted group request from a client and the sender sequence number of a group request just received from the same client. A client has to be always ready to perform the challenge-response based on the Echo Option in case a server starts it.

Note that endpoints configured as silent servers are not able to perform the challenge-response described above, as they do not store a Sender Context to secure the 4.03 Forbidden response to the client. Therefore, silent servers should adopt alternative approaches to achieve and maintain synchronization with sender sequence numbers of clients.

This approach provides an assurance of absolute message freshness. However, it can result in an impact on performance which is undesirable or unbearable, especially in large groups where many endpoints at the same time might join as new members or lose synchronization.

Appendix F. No Verification of Signatures

There are some application scenarios using group communication that have particularly strict requirements. One example of this is the requirement of low message latency in non-emergency lighting applications [I-D.somaraju-ace-multicast]. For those applications which have tight performance constraints and relaxed security requirements, it can be inconvenient for some endpoints to verify digital signatures in order to assert source authenticity of received messages. In other cases, the signature verification can be deferred or only checked for specific actions. For instance, a command to turn a bulb on where the bulb is already on does not need the signature to be checked. In such situations, the counter signature needs to be included anyway as part of the message, so that an endpoint that needs to validate the signature for any reason has the ability to do so.

In this specification, it is NOT RECOMMENDED that endpoints do not verify the counter signature of received messages. However, it is recognized that there may be situations where it is not always required. The consequence of not doing the signature validation is that security in the group is based only on the group-authenticity of the shared keying material used for encryption. That is, endpoints in the group have evidence that a received message has been originated by a group member, although not specifically identifiable in a secure way. This can violate a number of security requirements, as the compromise of any element in the group means that the attacker has the ability to control the entire group. Even worse, the group may not be limited in scope, and hence the same keying material might be used not only for light bulbs but for locks as well. Therefore, extreme care must be taken in situations where the security requirements are relaxed, so that deployment of the system will always be done safely.

Appendix G. Document Updates

RFC EDITOR: PLEASE REMOVE THIS SECTION.

G.1. Version -04 to -05

- o Added references to draft-dijk-core-groupcomm-bis.
- o New parameter Counter Signature Key Parameters (Section 2).
- o Clarification about Recipient COntexts (Section 2).
- o Two different external_aad for encrypting and signing (Section 3.1).

- o Updated response verification to handle Observe notifications (Section 6.4).
- o Extended Security Considerations (Section 8).
- o New "Counter Signature Key Parameters" IANA Registry (Section 9.2).

G.2. Version -03 to -04

- o Added the new "Counter Signature Parameters" in the Security Common Context (see Section 2).
- o Added recommendation on using "deterministic ECDSA" if ECDSA is used as counter signature algorithm (see Section 2).
- o Clarified possible asynchronous retrieval of key material from the Group Manager, in order to process incoming messages (see Section 2).
- o Structured Section 3 into subsections.
- o Added the new 'par_countersign' to the aad_array of the external_aad (see Section 3.1).
- o Clarified non reliability of 'kid' as identity indicator for a group member (see Section 2.1).
- o Described possible provisioning of new Sender ID in case of Partial IV wrap-around (see Section 2.2).
- o The former signature bit in the Flag Byte of the OSCORE option value is reverted to reserved (see Section 4.1).
- o Updated examples of compressed COSE object, now with the sixth less significant bit in the Flag Byte of the OSCORE option value set to 0 (see Section 4.3).
- o Relaxed statements on sending error messages (see Section 6).
- o Added explicit step on computing the counter signature for outgoing messages (see Sections 6.1 and 6.3).
- o Handling of just created Recipient Contexts in case of unsuccessful message verification (see Sections 6.2 and 6.4).
- o Handling of replied/repeated responses on the client (see Section 6.4).

- o New IANA Registry "Counter Signature Parameters" (see Section 9.1).

G.3. Version -02 to -03

- o Revised structure and phrasing for improved readability and better alignment with draft-ietf-core-object-security.
- o Added discussion on wrap-Around of Partial IVs (see Section 2.2).
- o Separate sections for the COSE Object (Section 3) and the OSCORE Header Compression (Section 4).
- o The countersignature is now appended to the encrypted payload of the OSCORE message, rather than included in the OSCORE Option (see Section 4).
- o Extended scope of Section 5, now titled " Message Binding, Sequence Numbers, Freshness and Replay Protection".
- o Clarifications about Non-Confirmable messages in Section 5.1 "Synchronization of Sender Sequence Numbers".
- o Clarifications about error handling in Section 6 "Message Processing".
- o Compacted list of responsibilities of the Group Manager in Section 7.
- o Revised and extended security considerations in Section 8.
- o Added IANA considerations for the OSCORE Flag Bits Registry in Section 9.
- o Revised Appendix D, now giving a short high-level description of a new endpoint set-up.

G.4. Version -01 to -02

- o Terminology has been made more aligned with RFC7252 and draft-ietf-core-object-security: i) "client" and "server" replace the old "multicaster" and "listener", respectively; ii) "silent server" replaces the old "pure listener".
- o Section 2 has been updated to have the Group Identifier stored in the 'ID Context' parameter defined in draft-ietf-core-object-security.

- o Section 3 has been updated with the new format of the Additional Authenticated Data.
- o Major rewriting of Section 4 to better highlight the differences with the message processing in draft-ietf-core-object-security.
- o Added Sections 7.2 and 7.3 discussing security considerations about uniqueness of (key, nonce) and collision of group identifiers, respectively.
- o Minor updates to Appendix A.1 about assumptions on multicast communication topology and group size.
- o Updated Appendix C on format of group identifiers, with practical implications of possible collisions of group identifiers.
- o Updated Appendix D.2, adding a pointer to draft-palombini-ace-key-groupcomm about retrieval of nodes' public keys through the Group Manager.
- o Minor updates to Appendix E.3 about Challenge-Response synchronization of sequence numbers based on the Echo option from draft-ietf-core-echo-request-tag.

G.5. Version -00 to -01

- o Section 1.1 has been updated with the definition of group as "security group".
- o Section 2 has been updated with:
 - * Clarifications on establishment/derivation of security contexts.
 - * A table summarizing the the additional context elements compared to OSCORE.
- o Section 3 has been updated with:
 - * Examples of request and response messages.
 - * Use of CounterSignature0 rather than CounterSignature.
 - * Additional Authenticated Data including also the signature algorithm, while not including the Group Identifier any longer.
- o Added Section 6, listing the responsibilities of the Group Manager.

- o Added Appendix A (former section), including assumptions and security objectives.
- o Appendix B has been updated with more details on the use cases.
- o Added Appendix C, providing an example of Group Identifier format.
- o Appendix D has been updated to be aligned with draft-palombini-ace-key-groupcomm.

Acknowledgments

The authors sincerely thank Stefan Beck, Rolf Blom, Carsten Bormann, Esko Dijk, Klaus Hartke, Rikard Hoeglund, Richard Kelsey, John Mattsson, Jim Schaad, Ludwig Seitz and Peter van der Stok for their feedback and comments.

The work on this document has been partly supported by VINNOVA and the Celtic-Next project CRITISEC; and by the EIT-Digital High Impact Initiative ACTIVE.

Authors' Addresses

Marco Tiloca
RISE AB
Isafjordsgatan 22
Kista SE-16440 Stockholm
Sweden

Email: marco.tiloca@ri.se

Goeran Selander
Ericsson AB
Torshamnsgatan 23
Kista SE-16440 Stockholm
Sweden

Email: goran.selander@ericsson.com

Francesca Palombini
Ericsson AB
Torshamnsgatan 23
Kista SE-16440 Stockholm
Sweden

Email: francesca.palombini@ericsson.com

Jiye Park
Universitaet Duisburg-Essen
Schuetzenbahn 70
Essen 45127
Germany

Email: ji-ye.park@uni-due.de

CoRE
Internet-Draft
Intended status: Standards Track
Expires: January 8, 2020

P. van der Stok
Consultant
M. Koster
SmartThings
C. Amsuess
Energy Harvesting Solutions
July 07, 2019

CoRE Resource Directory: DNS-SD mapping
draft-ietf-core-rd-dns-sd-05

Abstract

Resource and service discovery are complementary. Resource discovery provides fine-grained detail about the content of a web server, while service discovery can provide a scalable method to locate servers in large networks. This document defines a method for mapping between CoRE Link Format attributes and DNS-Based Service Discovery records to facilitate the use of either method to locate RESTful service interfaces (APIs) in heterogeneous HTTP/CoAP environments.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on January 8, 2020.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents

carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

| | |
|---|----|
| 1. Introduction and Background | 2 |
| 1.1. Terminology | 3 |
| 1.2. CoRE Resource Discovery | 4 |
| 1.3. CoRE Resource Directories | 5 |
| 1.4. DNS-Based Service Discovery | 5 |
| 2. New Link-Format Attributes | 6 |
| 2.1. Export attribute "exp" | 7 |
| 2.2. Resource Instance attribute "ins=" | 7 |
| 2.3. Service Type attribute "st=" | 7 |
| 3. Mapping CoRE Link Attributes to DNS-SD Record Fields | 7 |
| 3.1. Mapping Resource Instance attribute "ins=" to <Instance> | 7 |
| 3.2. Mapping Service Type attribute "st=" to <ServiceType> | 8 |
| 3.3. <Domain> Mapping | 8 |
| 3.4. TXT Record key=value strings | 9 |
| 3.5. Exporting resource links into DNS-SD | 9 |
| 4. Exporting Resource Directory Service to DNS | 10 |
| 5. IANA considerations | 10 |
| 5.1. RD Parameters Registry | 10 |
| 5.2. Service Name and Transport Protocol Port Number Registry | 11 |
| 6. Security considerations | 11 |
| 7. Contributors | 11 |
| 8. Acknowledgments | 11 |
| 9. References | 11 |
| 9.1. Normative References | 11 |
| 9.2. Informative References | 13 |
| Authors' Addresses | 14 |

1. Introduction and Background

The Constrained RESTful Environments (CoRE) working group aims at realizing the [REST] architecture in a suitable form for the most constrained devices (e.g. 8-bit microcontrollers with limited RAM and ROM) and networks (e.g. 6LoWPAN [RFC4944]). CoRE is aimed at machine-to-machine (M2M) applications such as smart energy and building automation. The main deliverable of CoRE is the Constrained Application Protocol (CoAP) specification [RFC7252].

CoRE Link Format [RFC6690] is intended to support fine-grained discovery of hosted resources, their attributes, and possibly other related resources. Automated dynamic discovery of resources hosted

by a constrained server is critical in M2M applications, where human intervention is minimal and static configurations result in brittleness.

DNS-Based Service Discovery (DNS-SD) [RFC6763] supports wide-area search for instances of a given service type (i.e. servers that support a particular application protocol stack). A service instance consists of a server's name, IP address, and port number plus additional meta-data about the server. This data may extend to support multi-function devices, where multiple services are available at the same endpoint. The result of the discovery process may include a path to a resource representing the entry point to each function's RESTful service interface and possibly a link to a formal description of that interface (e.g. a JSON Hyper-Schema document [I-D.handrews-json-schema-hyperschema]).

Resource and service discovery are complementary in the case of large networks, where the latter can facilitate scaling. This document defines a mapping between CoRE Link Format attributes and DNS-Based Service Discovery records that permits discovery of CoAP services by either method. It also addresses the CoRE charter goal to interoperate with DNS-SD.

The primary use case for mapping between resource and service discovery is to support heterogeneous HTTP/CoAP environments where, for example, HTTP clients may discover and communicate with CoAP servers that are behind a "cross proxy" [RFC8075].

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 [RFC2119]. The term "byte" is used in its now conventional sense as a synonym for "octet".

This specification requires readers to be familiar with all the terms and concepts that are discussed in [RFC6690] and [RFC8288]. Readers should also be familiar with the terms and concepts discussed in [RFC7252].

This specification also incorporates the terminology of [I-D.ietf-core-resource-directory].

In particular, the following terms are used frequently:

Endpoint: a web server associated with a specific IP address and port; thus a physical device may host one or more endpoints. Endpoints may also act as clients.

Link: Web Linking [RFC8288] defines a Web Link (link) as a typed connection between two resources, comprised of:

- o a link context,
- o a link relation type (see Section 2.1 of [RFC8288],
- o a link target, and
- o optionally, target attributes (see Section 2.2 of [RFC8288]).

A link can be viewed as a statement of the form "link context has a link relation type resource at link target, which (optionally) has target attributes", where link target and context are typically Universal Resource Identifiers (URIs) [RFC3986]. For example, "https://www.example.com/" has a "canonical" resource at "https://example.com", which has a "type" of "text/html".

1.2. CoRE Resource Discovery

The main function of Resource Discovery is to return links to the resources hosted by a server, complemented by attributes about those resources and additional link relations. In CoRE this collection of links and attributes is itself a resource (in contrast to HTTP, where headers delivered with a specific resource describe its attributes).

Resource Discovery can be performed either unicast or multicast. When a server's IP address is already known, either a priori or resolved via the Domain Name System (DNS) [RFC1034][RFC1035], unicast discovery is performed in order to locate the entry point to the resource of interest. This is performed using a GET to `"/.well-known/core"` on the server, which returns a payload in the CoRE Link Format [RFC6690]. A client would then match the appropriate Resource Type, Interface Description, and possible media type [RFC2045] for its application. These attributes may also be included in the query string in order to filter the number of links returned in a response.

Multicast Resource Discovery is useful when a client needs to locate a resource within a limited scope, and that scope supports IP multicast. A GET request to the appropriate multicast address is made for `"/.well-known/core"`. In order to limit the number and size of responses, a query string is recommended with the known attributes. Typically, a resource would be discovered based on its

Resource Type and/or Interface Description, along with possible application-specific attributes.

1.3. CoRE Resource Directories

In many M2M scenarios, direct discovery of resources is not practical due to sleeping nodes, limited bandwidth, or networks where multicast traffic is inefficient. These problems can be solved by deploying a network element called a Resource Directory (RD), which hosts descriptions of resources that originate on other endpoints and allows indirect lookups to be performed for those resources.

The Resource Directory implements a set of REST interfaces for endpoints to register and maintain collections of links, called Resource Directory registrations. [I-D.ietf-core-resource-directory] specifies the web interfaces that an RD supports for endpoints to discover the RD and to register, maintain, lookup and remove resource descriptions; for the RD to validate entries; and for clients to lookup resources from the RD.

1.4. DNS-Based Service Discovery

DNS-Based Service Discovery (DNS-SD) defines a conventional method of naming and configuring DNS PTR, SRV, and TXT resource records to facilitate discovery of services (such as CoAP servers in a subdomain) using the existing DNS infrastructure. This section gives a brief overview of DNS-SD; for a detailed specification see [RFC6763].

DNS-SD Service Names are limited to 255 bytes and are of the form:

Service Name = <Instance>.<ServiceType>.<Domain>

The Service Name identifies a SRV/TXT Resource Record (RR) pair. The SRV RR specifies the hostname and port of an endpoint. The TXT RR provides additional information in the form of key/value pairs. DNS-Based Service Discovery is accomplished by sending a DNS request for PTR records with the name <ServiceType>.<Domain>, which will return a list of zero or more Service Names.

The <Domain> part of the Service Name is identical to the global (DNS subdomain) part of the authority in URIs [RFC3986] that identify the resources on an individual server or group of servers.

The <ServiceType> part is generally composed of two labels. The first label of the pair is the application protocol name [RFC6335] preceded by an underscore character. For example, an organization such as the Open Connectivity Foundation [OCF] that specifies

Resource Types [RFC6335] might register application protocol names beginning with "oic", which all servers that advertise OCF resources would use as part of their ServiceType. The second label indicates the transport protocol binding and is typically "_udp" for CoAP services.

The default <Instance> part of the Service Name SHOULD be set to a default value at the factory and MAY be modified during the commissioning process. It MUST uniquely identify an instance of <ServiceType> within a <Domain>. Taken together, these three elements comprise a unique name for an SRV/TXT record pair within the DNS subdomain.

The granularity of a Service Name MAY be that of a host or group, or it might represent a particular resource within a CoAP server. The SRV record contains the host name (AAAA record name) and port of the endpoint, while protocol is part of the Service Name. In the case where a Service Name identifies a particular resource, the path part of the URI must be carried in a corresponding TXT record.

A DNS TXT record is in practice limited to a few hundred bytes in length, which is indicated in the resource record header in the DNS response message (See section 6 of [RFC6763]). The data consist of one or more strings comprising a key/value pair. By convention, the first pair is txtver=<number> (to support different versions of a service description). Each string is formatted as a single length byte followed by 0-255 bytes of text. An example string is:

```
-----
| 0x08 | t | x | t | v | e | r | = | 1 |
-----
```

2. New Link-Format Attributes

When using the CoRE Link Format to describe resources being discovered by or posted to a resource directory service, additional information about those resources is often useful. This specification defines the following new attributes for use in the CoRE Link Format [RFC6690] to enable the data-driven mappings described in Section 3:

```
link-extension = ( "exp" )
link-extension = ( "ins" "=" (ptoken | quoted-string) )
                  ; The token or string is max 63 bytes
link-extension = ( "st" "=" (ptoken | quoted-string) )
                  ; The token or string is max 15 bytes
```

2.1. Export attribute "exp"

The Export "exp" attribute is used as a flag to indicate that a link description MAY be exported from a resource directory to external directories.

The CoRE Link Format is used for many purposes between CoAP endpoints. Some are useful mainly locally; for example checking the observability of a resource before accessing it, determining the size of a resource, or traversing dynamic resource structures. However, other links are very useful to be exported to other directories, for example the entry point resource to a functional service. This attribute MAY be used as a query parameter in the RD Lookup Function Set defined in Section 6 of [I-D.ietf-core-resource-directory].

2.2. Resource Instance attribute "ins="

The Resource Instance "ins=" attribute is an identifier for this resource, which makes it possible to distinguish it from other similar resources in a Resource Directory. This attribute specifies the value to be used for the <Instance> portion of an exported DNS-SD Service Name (see Section 1.4), and SHOULD be unique across resources with the same Resource Type "rt=" attribute in the domain in which it is used.

A Resource Instance SHOULD be a descriptive human readable string like "Ceiling Light, Room 3". This attribute MUST NOT be more than 63 bytes in length. The resource identifier attribute MUST NOT appear more than once in a link description. This attribute MAY be used as a query parameter in the RD Lookup Function Set defined in Section 7 of [I-D.ietf-core-resource-directory].

2.3. Service Type attribute "st="

The Service Type instance "st=" attribute specifies the value to be used for the <ServiceType> portion of an exported DNS-SD Service Name (see Section 1.4). This attribute MUST NOT be more than 15 bytes in length (see [RFC6335], Section 5.1) and MUST be present in the IANA Service Name registry [st].

3. Mapping CoRE Link Attributes to DNS-SD Record Fields

3.1. Mapping Resource Instance attribute "ins=" to <Instance>

The Resource Instance "ins=" attribute maps directly to the <Instance> part of a DNS-SD Service Name. It is stored directly in the DNS as a single DNS label of canonical precomposed UTF-8 [RFC3629] "Net-Unicode" (Unicode Normalization Form C) [RFC5198]

text. However, if the "ins=" attribute is chosen to match the DNS host name of a service, it SHOULD use the syntax defined in Section 3.5 of [RFC1034] and Section 2.1 of [RFC1123].

The <Instance> part of the name of a service being offered on the network SHOULD be configurable by the user setting up the service, so that he or she may give it an informative name. However, the device or service SHOULD NOT require the user to configure a name before it can be used. A sensible choice of default name can allow the device or service to be accessed in many cases without any manual configuration at all (see Appendix D of [RFC6763]).

DNS labels are limited to 63 bytes in length and the entire Service Name may not exceed 255 bytes.

3.2. Mapping Service Type attribute "st=" to <ServiceType>

The Service Type "st=" attribute maps directly to the <ServiceType> part of a DNS-SD Service Name.

In practice, the ServiceType should unambiguously identify interoperable devices. It is up to individual SDOs to specify how to represent their registered Resource Type "rt=" values as registered application protocol names according to [RFC6335]. The application name is then used as the value of the resource "st=" attribute.

The resulting application protocol name MUST be composed of at least a single Net-Unicode text string, without underscore '_' or period '.' and limited to 15 bytes in length (see Section 5.1 of [RFC6335]). This string is mapped to the DNS-SD <ServiceType> by prepending an underscore and appending a period followed by the "_udp" label. For example, rt="oic.d.light" might correspond to the registered application protocol name st="oic-d-light" and would be mapped into Service Type "_oic-d-light._udp".

The resulting string is used to form labels for DNS-SD records which are stored directly in the DNS.

3.3. <Domain> Mapping

TBD: A method must be specified to determine which DNS zone the CoAP service description should be exported to. See, for example, Section 11 in [RFC6763] and Section 2 in [I-D.sctl-service-registration].

3.4. TXT Record key=value strings

DNS-SD key/value pairs may be derived from CoRE Link Format information and exported as key=value strings in a DNS-SD TXT record (See Section 6.3 of [RFC6763]).

The resource <URI> is exported as key/value pair "path=<URI>".

The Interface Description "if=" attribute is exported as key/value pair "if=<Interface Description>".

The DNS TXT record can be further populated by importing any other resource description attributes as they share the same key=value format specified in Section 6 of [RFC6763].

3.5. Exporting resource links into DNS-SD

Assuming the ability to query a Resource Directory or multicast a GET (?exp) over the local link, CoAP resource discovery may be used to populate the DNS-SD database in an automated fashion. CoAP resource descriptions (links) can be exported to DNS-SD for exposure to service discovery by using the Resource Instance attribute as the basis for a unique Service Name, composed with the Service Type attribute as the <ServiceType>, and registered in the appropriate <Domain>. The agent responsible for exporting records to the DNS zone file SHOULD be authenticated to the DNS server. The following example, using the example lookup location /rd-lookup, shows an agent discovering a resource to be exported:

```
Req: GET /rd-lookup/res?exp

Res: 2.05 Content
<coap://[FDFD::1234]:5683/light/1>;
  exp;st=oic-d-light;rt="oic.d.light";ins="Spot";
  d="sector";ep="node1"
```

The agent subsequently registers the following DNS-SD RRs, assuming a derived DNS zone name "office.example.com":

```
_oic-d-light._udp.office.example.com      IN PTR
  Spot._oic-d-light._udp.office.example.com
Spot._oic-d-light._udp.office.example.com IN TXT
  txtver=1;path=/light/1;rt=oic.d.light;d=sector
Spot._oic-d-light._udp.office.example.com IN SRV
  0 0 5683 node1.office.example.com.
node1.office.example.com.                  IN AAAA FDFD::1234
```


4. Exporting Resource Directory Service to DNS

In some cases it is required that one (or more) Resource Directories (RD) in a given DNS domain can be discoverable from DNS. The `/.well-known/core` resource of the RD should reflect this by specifying the "ins", "exp", and the "st" attributes in the the link of the RD service. This document specifies in Section 5 two servicetypes: `_rd-lookup-res._udp` and `_rd-lookup-ep._udp` for resource types `rt = core.rd-lookup-res` and `rt = core.rd-lookup-ep` respectively. The default coap and coaps ports are respectively: 5683 and 5684.

The value of the instance MAY be specified by the manager of the resource directories. In case of an unmanaged RD (for example in a home network) it is recommended that the ins parameter takes a value provided by an Authorization Server during the acceptance of the RD to the network (see for example section 7 of [I-D.ietf-core-resource-directory]).

With the assumption that the "ins" value is attributed by Authorization Server, and `[FDFD::1234]` is IPaddress of RD, Example links for RD are:

```
Req: GET coap://[FDFD::1234]/.well-known/core?exp
```

```
Res: 2.05 Content
<rd-lookup/res>;
  exp;st=rd-lookup-res;rt="core.rd-lookup-res";
  ins="505567",
<rd-lookup/ep>;
  exp;st=rd-lookup-ep;rt="core.rd-lookup-ep";
  ins="505572"
```

The link attributes can be exported to RR by the mapping process described in Section 3.

5. IANA considerations

Two registries are affected by this document: (1) "RD Parameters" registry under "Core Parameters" registry, and (2) Service Name and Transport Protocol Port Number Registry

5.1. RD Parameters Registry

This specification defines new parameters for the registry "RD Parameters" provided under "CoRE Parameters" (TBD).

| Full name | Short | Validity | Use | Description |
|-------------------|-------|----------|-----|--|
| ServiceType | st | | RLA | Name of the Service Type, max 63 bytes |
| Resource Instance | ins | | RLA | Instance identifier of the resource |
| Export | exp | | RLA | flag to indicate exportation |

5.2. Service Name and Transport Protocol Port Number Registry

This specification defines new parameters for the Service Name and Transport Protocol Port Number Registry:

- * `_rd-lookup-res._udp` at ports 5683 and 5684
- * `_rd-lookup-ep._udp` at ports 5683 and 5684

6. Security considerations

Malicious nodes can export fake link attributes to DNS. It is recommended that the RD can be authenticated, and is authorized to both join the network and export its link attributes. Authentication is specified in [I-D.ietf-ace-oauth-authz].

7. Contributors

Kerry Lynn was the initiator of, and major contributor to this document. This document was split out from [I-D.ietf-core-resource-directory]. Zach Shelby was a co-author of the original version of this draft.

8. Acknowledgments

The authors wish to thank Stuart Cheshire, Ted Lemon, and David Thaler for their thorough reviews and clarifying suggestions.

9. References

9.1. Normative References

- [RFC1034] Mockapetris, P., "Domain names - concepts and facilities", STD 13, RFC 1034, DOI 10.17487/RFC1034, November 1987, <<https://www.rfc-editor.org/info/rfc1034>>.

- [RFC1035] Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, DOI 10.17487/RFC1035, November 1987, <<https://www.rfc-editor.org/info/rfc1035>>.
- [RFC1123] Braden, R., Ed., "Requirements for Internet Hosts - Application and Support", STD 3, RFC 1123, DOI 10.17487/RFC1123, October 1989, <<https://www.rfc-editor.org/info/rfc1123>>.
- [RFC2045] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, DOI 10.17487/RFC2045, November 1996, <<https://www.rfc-editor.org/info/rfc2045>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, DOI 10.17487/RFC3629, November 2003, <<https://www.rfc-editor.org/info/rfc3629>>.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <<https://www.rfc-editor.org/info/rfc3986>>.
- [RFC4944] Montenegro, G., Kushalnagar, N., Hui, J., and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks", RFC 4944, DOI 10.17487/RFC4944, September 2007, <<https://www.rfc-editor.org/info/rfc4944>>.
- [RFC5198] Klensin, J. and M. Padlipsky, "Unicode Format for Network Interchange", RFC 5198, DOI 10.17487/RFC5198, March 2008, <<https://www.rfc-editor.org/info/rfc5198>>.
- [RFC6335] Cotton, M., Eggert, L., Touch, J., Westerlund, M., and S. Cheshire, "Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry", BCP 165, RFC 6335, DOI 10.17487/RFC6335, August 2011, <<https://www.rfc-editor.org/info/rfc6335>>.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <<https://www.rfc-editor.org/info/rfc6690>>.

- [RFC6763] Cheshire, S. and M. Krochmal, "DNS-Based Service Discovery", RFC 6763, DOI 10.17487/RFC6763, February 2013, <<https://www.rfc-editor.org/info/rfc6763>>.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<https://www.rfc-editor.org/info/rfc7252>>.
- [RFC8075] Castellani, A., Loreto, S., Rahman, A., Fossati, T., and E. Dijk, "Guidelines for Mapping Implementations: HTTP to the Constrained Application Protocol (CoAP)", RFC 8075, DOI 10.17487/RFC8075, February 2017, <<https://www.rfc-editor.org/info/rfc8075>>.
- [RFC8288] Nottingham, M., "Web Linking", RFC 8288, DOI 10.17487/RFC8288, October 2017, <<https://www.rfc-editor.org/info/rfc8288>>.

9.2. Informative References

- [I-D.handrews-json-schema-hyperschema]
Andrews, H. and A. Wright, "JSON Hyper-Schema: A Vocabulary for Hypermedia Annotation of JSON", draft-handrews-json-schema-hyperschema-01 (work in progress), January 2018.
- [I-D.ietf-ace-oauth-authz]
Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "Authentication and Authorization for Constrained Environments (ACE) using the OAuth 2.0 Framework (ACE-OAuth)", draft-ietf-ace-oauth-authz-24 (work in progress), March 2019.
- [I-D.ietf-core-resource-directory]
Shelby, Z., Koster, M., Bormann, C., Stok, P., and C. Amsuess, "CoRE Resource Directory", draft-ietf-core-resource-directory-22 (work in progress), July 2019.
- [I-D.sctl-service-registration]
Cheshire, S. and T. Lemon, "Service Registration Protocol for DNS-Based Service Discovery", draft-sctl-service-registration-02 (work in progress), July 2018.
- [OCF] Foundation, O., "OCF Specification 2.0", 2018, <<https://openconnectivity.org/developer/specifications>>.

- [REST] Fielding, R., "Architectural Styles and the Design of Network-based Software Architectures", 2000, <http://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf>.
- [rt] IANA, ., "Resource Type (rt=) Link Target Attribute Values", 2012, <<https://www.iana.org/assignments/core-parameters/core-parameters.xhtml#rt-link-target-att-value>>.
- [st] IANA, ., "Service Name and Transport Protocol Port Number Registry", 2018, <<https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xml>>.

Authors' Addresses

Peter van der Stok
Consultant

Phone: +31 492474673 (Netherlands), +33 966015248 (France)
Email: consultancy@vanderstok.org
URI: www.vanderstok.org

Michael Koster
SmartThings
665 Clyde Avenue
Mountain View, CA 94043
USA

Phone: +1 707-502-5136
Email: Michael.Koster@smartthings.com

Christian Amsuess
Energy Harvesting Solutions
Hollandstr. 12/4
1020
Austria

Phone: +43 664-9790639
Email: c.amsuess@energyharvesting.at

CoRE
Internet-Draft
Intended status: Standards Track
Expires: January 9, 2020

Z. Shelby
ARM
M. Koster
SmartThings
C. Bormann
Universitaet Bremen TZI
P. van der Stok
consultant
C. Amsuess, Ed.
July 08, 2019

CoRE Resource Directory
draft-ietf-core-resource-directory-23

Abstract

In many IoT applications, direct discovery of resources is not practical due to sleeping nodes, disperse networks, or networks where multicast traffic is inefficient. These problems can be solved by employing an entity called a Resource Directory (RD), which contains information about resources held on other servers, allowing lookups to be performed for those resources. The input to an RD is composed of links and the output is composed of links constructed from the information stored in the RD. This document specifies the web interfaces that a Resource Directory supports for web servers to discover the RD and to register, maintain, lookup and remove information on resources. Furthermore, new target attributes useful in conjunction with an RD are defined.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on January 9, 2020.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

| | |
|--|----|
| 1. Introduction | 3 |
| 2. Terminology | 4 |
| 3. Architecture and Use Cases | 6 |
| 3.1. Principles | 6 |
| 3.2. Architecture | 7 |
| 3.3. RD Content Model | 8 |
| 3.4. Link-local addresses and zone identifiers | 12 |
| 3.5. Use Case: Cellular M2M | 12 |
| 3.6. Use Case: Home and Building Automation | 13 |
| 3.7. Use Case: Link Catalogues | 14 |
| 4. RD discovery and other interface-independent components | 14 |
| 4.1. Finding a Resource Directory | 15 |
| 4.1.1. Resource Directory Address Option (RDAO) | 17 |
| 4.2. Payload Content Formats | 18 |
| 4.3. URI Discovery | 19 |
| 5. Registration | 21 |
| 5.1. Simple Registration | 26 |
| 5.2. Third-party registration | 28 |
| 5.3. Operations on the Registration Resource | 29 |
| 5.3.1. Registration Update | 29 |
| 5.3.2. Registration Removal | 32 |
| 5.3.3. Further operations | 33 |
| 6. RD Lookup | 33 |
| 6.1. Resource lookup | 34 |
| 6.2. Lookup filtering | 34 |
| 6.3. Resource lookup examples | 36 |
| 6.4. Endpoint lookup | 39 |
| 7. Security policies | 40 |
| 7.1. Secure RD discovery | 41 |
| 7.2. Secure RD filtering | 42 |
| 7.3. Secure endpoint Name assignment | 42 |

| | | |
|--------------------|---|----|
| 8. | Security Considerations | 42 |
| 8.1. | Endpoint Identification and Authentication | 42 |
| 8.2. | Access Control | 43 |
| 8.3. | Denial of Service Attacks | 43 |
| 9. | IANA Considerations | 44 |
| 9.1. | Resource Types | 44 |
| 9.2. | IPv6 ND Resource Directory Address Option | 44 |
| 9.3. | RD Parameter Registry | 44 |
| 9.3.1. | Full description of the "Endpoint Type" Registration Parameter | 46 |
| 9.4. | "Endpoint Type" (et=) RD Parameter values | 46 |
| 9.5. | Multicast Address Registration | 47 |
| 10. | Examples | 47 |
| 10.1. | Lighting Installation | 48 |
| 10.1.1. | Installation Characteristics | 48 |
| 10.1.2. | RD entries | 49 |
| 10.2. | OMA Lightweight M2M (LWM2M) Example | 52 |
| 10.2.1. | The LWM2M Object Model | 52 |
| 10.2.2. | LWM2M Register Endpoint | 54 |
| 10.2.3. | LWM2M Update Endpoint Registration | 55 |
| 10.2.4. | LWM2M De-Register Endpoint | 56 |
| 11. | Acknowledgments | 56 |
| 12. | Changelog | 56 |
| 13. | References | 66 |
| 13.1. | Normative References | 66 |
| 13.2. | Informative References | 66 |
| Appendix A. | Groups Registration and Lookup | 68 |
| Appendix B. | Web links and the Resource Directory | 70 |
| B.1. | A simple example | 70 |
| B.1.1. | Resolving the URIs | 71 |
| B.1.2. | Interpreting attributes and relations | 71 |
| B.2. | A slightly more complex example | 71 |
| B.3. | Enter the Resource Directory | 72 |
| B.4. | A note on differences between link-format and Link headers | 74 |
| Appendix C. | Limited Link Format | 75 |
| Authors' Addresses | | 75 |

1. Introduction

In the work on Constrained RESTful Environments (CoRE), a REST architecture suitable for constrained nodes (e.g. with limited RAM and ROM [RFC7228]) and networks (e.g. 6LoWPAN [RFC4944]) has been established and is used in Internet-of-Things (IoT) or machine-to-machine (M2M) applications such as smart energy and building automation.

The discovery of resources offered by a constrained server is very important in machine-to-machine applications where there are no humans in the loop and static interfaces result in fragility. The discovery of resources provided by an HTTP Web Server is typically called Web Linking [RFC8288]. The use of Web Linking for the description and discovery of resources hosted by constrained web servers is specified by the CoRE Link Format [RFC6690]. However, [RFC6690] only describes how to discover resources from the web server that hosts them by querying `"/.well-known/core"`. In many constrained scenarios, direct discovery of resources is not practical due to sleeping nodes, disperse networks, or networks where multicast traffic is inefficient. These problems can be solved by employing an entity called a Resource Directory (RD), which contains information about resources held on other servers, allowing lookups to be performed for those resources.

This document specifies the web interfaces that a Resource Directory supports for web servers to discover the RD and to register, maintain, lookup and remove information on resources. Furthermore, new target attributes useful in conjunction with a Resource Directory are defined. Although the examples in this document show the use of these interfaces with CoAP [RFC7252], they can be applied in an equivalent manner to HTTP [RFC7230].

2. 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 [RFC2119]. The term "byte" is used in its now customary sense as a synonym for "octet".

This specification requires readers to be familiar with all the terms and concepts that are discussed in [RFC3986], [RFC8288] and [RFC6690]. Readers should also be familiar with the terms and concepts discussed in [RFC7252]. To describe the REST interfaces defined in this specification, the URI Template format is used [RFC6570].

This specification makes use of the following additional terminology:

resolve against

The expression "a URI-reference is `_resolved against_` a base URI" is used to describe the process of [RFC3986] Section 5.2.

Noteworthy corner cases are that if the URI-reference is a (full) URI and resolved against any base URI, that gives the original full URI, and that resolving an empty URI reference gives the base URI without any fragment identifier.

Resource Directory

A web entity that stores information about web resources and implements the REST interfaces defined in this specification for registration and lookup of those resources.

Sector

In the context of a Resource Directory, a sector is a logical grouping of endpoints.

The abbreviation "d=" is used for the sector in query parameters for compatibility with deployed implementations.

Endpoint

Endpoint (EP) is a term used to describe a web server or client in [RFC7252]. In the context of this specification an endpoint is used to describe a web server that registers resources to the Resource Directory. An endpoint is identified by its endpoint name, which is included during registration, and has a unique name within the associated sector of the registration.

Registration Base URI

The Base URI of a Registration is a URI that typically gives scheme and authority information about an Endpoint. The Registration Base URI is provided at registration time, and is used by the Resource Directory to resolve relative references of the registration into URIs.

Target

The target of a link is the destination address (URI) of the link. It is sometimes identified with "href=", or displayed as "<target>". Relative targets need resolving with respect to the Base URI (section 5.2 of [RFC3986]).

This use of the term Target is consistent with [RFC8288]'s use of the term.

Context

The context of a link is the source address (URI) of the link, and describes which resource is linked to the target. A link's context is made explicit in serialized links as the "anchor=" attribute.

This use of the term Context is consistent with [RFC8288]'s use of the term.

Directory Resource

A resource in the Resource Directory (RD) containing registration resources.

Registration Resource

A resource in the RD that contains information about an Endpoint and its links.

Commissioning Tool

Commissioning Tool (CT) is a device that assists during the installation of the network by assigning values to parameters, naming endpoints and groups, or adapting the installation to the needs of the applications.

Registrant-ep

Registrant-ep is the endpoint that is registered into the RD. The registrant-ep can register itself, or a CT registers the registrant-ep.

RDAO

Resource Directory Address Option. A new IPv6 Neighbor Discovery option defined for announcing a Resource Directory's address.

3. Architecture and Use Cases

3.1. Principles

The Resource Directory is primarily a tool to make discovery operations more efficient than querying /.well-known/core on all connected devices, or across boundaries that would be limiting those operations.

It provides information about resources hosted by other devices that could otherwise only be obtained by directly querying the /.well-known/core resource on these other devices, either by a unicast request or a multicast request.

Information SHOULD only be stored in the resource directory if it can be obtained by querying the described device's /.well-known/core resource directly.

Data in the resource directory can only be provided by the device which hosts those data or a dedicated Commissioning Tool (CT). These CTs are thought to act on behalf of endpoints too constrained, or generally unable, to present that information themselves. No other client can modify data in the resource directory. Changes to the information in the Resource Directory do not propagate automatically back to the web servers from where the information originated.

3.2. Architecture

The resource directory architecture is illustrated in Figure 1. A Resource Directory (RD) is used as a repository of registrations describing resources hosted on other web servers, also called endpoints (EP). An endpoint is a web server associated with a scheme, IP address and port. A physical node may host one or more endpoints. The RD implements a set of REST interfaces for endpoints to register and maintain resource directory registrations, and for endpoints to lookup resources from the RD. An RD can be logically segmented by the use of Sectors.

A mechanism to discover an RD using CoRE Link Format [RFC6690] is defined.

Registrations in the RD are soft state and need to be periodically refreshed.

An endpoint uses specific interfaces to register, update and remove a registration. It is also possible for an RD to fetch Web Links from endpoints and add their contents to resource directory registrations.

At the first registration of an endpoint, a "registration resource" is created, the location of which is returned to the registering endpoint. The registering endpoint uses this registration resource to manage the contents of registrations.

A lookup interface for discovering any of the Web Links stored in the RD is provided using the CoRE Link Format.

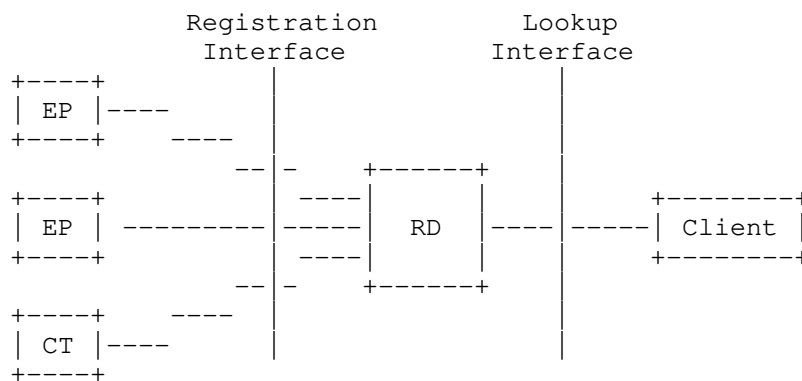


Figure 1: The resource directory architecture.

A Registrant-EP MAY keep concurrent registrations to more than one RD at the same time if explicitly configured to do so, but that is not expected to be supported by typical EP implementations. Any such registrations are independent of each other. The usual expectation when multiple discovery mechanisms or addresses are configured is that they constitute a fall-back path for a single registration.

3.3. RD Content Model

The Entity-Relationship (ER) models shown in Figure 2 and Figure 3 model the contents of /.well-known/core and the resource directory respectively, with entity-relationship diagrams [ER]. Entities (rectangles) are used for concepts that exist independently. Attributes (ovals) are used for concepts that exist only in connection with a related entity. Relations (diamonds) give a semantic meaning to the relation between entities. Numbers specify the cardinality of the relations.

Some of the attribute values are URIs. Those values are always full URIs and never relative references in the information model. They can, however, be expressed as relative references in serializations, and often are.

These models provide an abstract view of the information expressed in link-format documents and a Resource Directory. They cover the concepts, but not necessarily all details of an RD's operation; they are meant to give an overview, and not be a template for implementations.

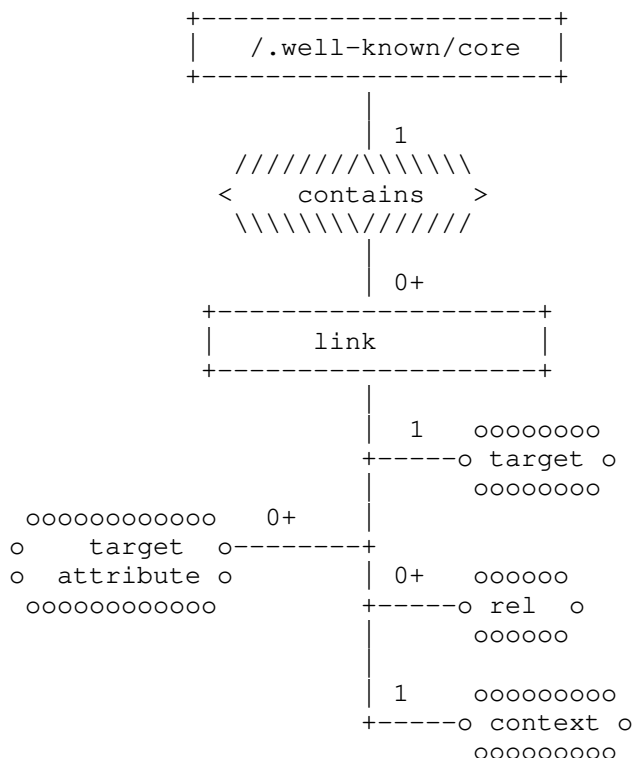


Figure 2: E-R Model of the content of `/.well-known/core`

The model shown in Figure 2 models the contents of `/.well-known/core` which contains:

- o a set of links belonging to the hosting web server

The web server is free to choose links it deems appropriate to be exposed in its ".well-known/core". Typically, the links describe resources that are served by the host, but the set can also contain links to resources on other servers (see examples in [RFC6690] page 14). The set does not necessarily contain links to all resources served by the host.

A link has the following attributes (see [RFC8288]):

- o Zero or more link relations: They describe relations between the link context and the link target.

In link-format serialization, they are expressed as space-separated values in the "rel" attribute, and default to "hosts".

- o A link context URI: It defines the source of the relation, e.g. `_who_ "hosts" something`.

In link-format serialization, it is expressed in the "anchor" attribute. It defaults to that document's URI.

- o A link target URI: It defines the destination of the relation (e.g. `_what_ is hosted`), and is the topic of all target attributes.

In link-format serialization, it is expressed between angular brackets, and sometimes called the "href".

- o Other target attributes (e.g. resource type (rt), interface (if), or content format (ct)). These provide additional information about the target URI.

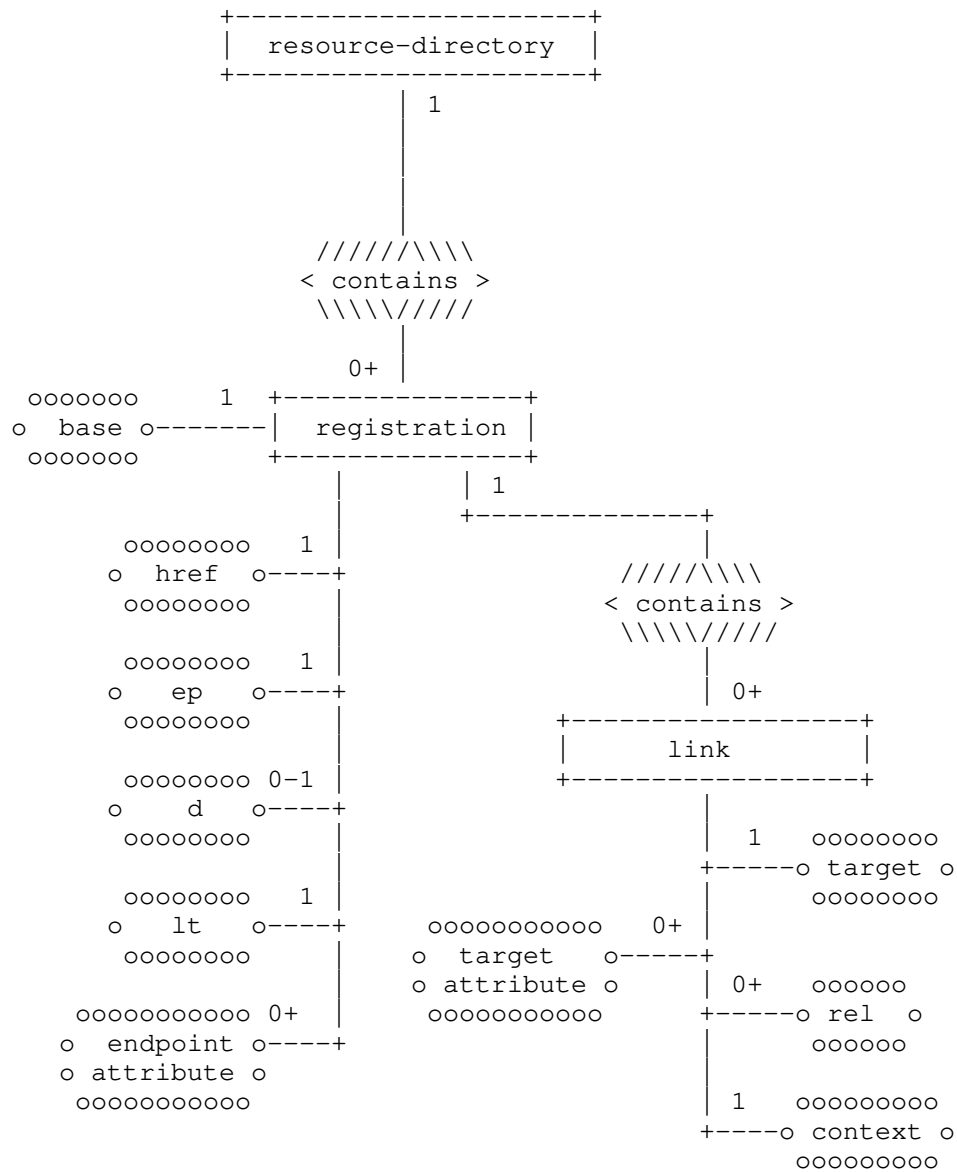


Figure 3: E-R Model of the content of the Resource Directory

The model shown in Figure 3 models the contents of the resource directory which contains in addition to /.well-known/core:

- o 0 to n Registrations of endpoints,

A registration is associated with one endpoint. A registration defines a set of links as defined for `/.well-known/core`. A Registration has six types of attributes:

- o an endpoint name ("ep", a Unicode string) unique within a sector
- o a Registration Base URI ("base", a URI typically describing the `scheme://authority` part)
- o a lifetime ("lt"),
- o a registration resource location inside the RD ("href"),
- o optionally a sector ("d", a Unicode string)
- o optional additional endpoint attributes (from Section 9.3)

The cardinality of "base" is currently 1; future documents are invited to extend the RD specification to support multiple values (e.g. [I-D.silverajan-core-coap-protocol-negotiation]). Its value is used as a Base URI when resolving URIs in the links contained in the endpoint.

Links are modelled as they are in Figure 2.

3.4. Link-local addresses and zone identifiers

Registration Base URIs can contain link-local IP addresses. To be usable across hosts, those can not be serialized to contain zone identifiers (see [RFC6874] Section 1).

Link-local addresses can only be used on a single link (therefore RD servers can not announce them when queried on a different link), and lookup clients using them need to keep track of which interface they got them from.

Therefore, it is advisable in many scenarios to use addresses with larger scope if available.

3.5. Use Case: Cellular M2M

Over the last few years, mobile operators around the world have focused on development of M2M solutions in order to expand the business to the new type of users: machines. The machines are connected directly to a mobile network using an appropriate embedded wireless interface (GSM/GPRS, WCDMA, LTE) or via a gateway providing short and wide range wireless interfaces. From the system design point of view, the ambition is to design horizontal solutions that

can enable utilization of machines in different applications depending on their current availability and capabilities as well as application requirements, thus avoiding silo like solutions. One of the crucial enablers of such design is the ability to discover resources (machines -- endpoints) capable of providing required information at a given time or acting on instructions from the end users.

Imagine a scenario where endpoints installed on vehicles enable tracking of the position of these vehicles for fleet management purposes and allow monitoring of environment parameters. During the boot-up process endpoints register with a Resource Directory, which is hosted by the mobile operator or somewhere in the cloud. Periodically, these endpoints update their registration and may modify resources they offer.

When endpoints are not always connected, for example because they enter a sleep mode, a remote server is usually used to provide proxy access to the endpoints. Mobile apps or web applications for environment monitoring contact the RD, look up the endpoints capable of providing information about the environment using an appropriate set of link parameters, obtain information on how to contact them (URLs of the proxy server), and then initiate interaction to obtain information that is finally processed, displayed on the screen and usually stored in a database. Similarly, fleet management systems provide the appropriate link parameters to the RD to look up for EPs deployed on the vehicles the application is responsible for.

3.6. Use Case: Home and Building Automation

Home and commercial building automation systems can benefit from the use of M2M web services. The discovery requirements of these applications are demanding. Home automation usually relies on run-time discovery to commission the system, whereas in building automation a combination of professional commissioning and run-time discovery is used. Both home and building automation involve peer-to-peer interactions between endpoints, and involve battery-powered sleeping devices.

Two phases can be discerned for a network servicing the system: (1) installation and (2) operation. During the operational phase, the network is connected to the Internet with a Border router (6LBR) and the nodes connected to the network can use the Internet services that are provided by the Internet Provider or the network administrator. During the installation phase, the network is completely stand-alone, no 6LBR is connected, and the network only supports the IP communication between the connected nodes. The installation phase is usually followed by the operational phase.

3.7. Use Case: Link Catalogues

Resources may be shared through data brokers that have no knowledge beforehand of who is going to consume the data. Resource Directory can be used to hold links about resources and services hosted anywhere to make them discoverable by a general class of applications.

For example, environmental and weather sensors that generate data for public consumption may provide data to an intermediary server, or broker. Sensor data are published to the intermediary upon changes or at regular intervals. Descriptions of the sensors that resolve to links to sensor data may be published to a Resource Directory. Applications wishing to consume the data can use RD Lookup to discover and resolve links to the desired resources and endpoints. The Resource Directory service need not be coupled with the data intermediary service. Mapping of Resource Directories to data intermediaries may be many-to-many.

Metadata in web link formats like [RFC6690] which may be internally stored as triples, or relation/attribute pairs providing metadata about resource links, need to be supported by Resource Directories. External catalogues that are represented in other formats may be converted to common web linking formats for storage and access by Resource Directories. Since it is common practice for these to be URN encoded, simple and lossless structural transforms should generally be sufficient to store external metadata in Resource Directories.

The additional features of Resource Directory allow sectors to be defined to enable access to a particular set of resources from particular applications. This provides isolation and protection of sensitive data when needed. Application groups with multicast addresses may be defined to support efficient data transport.

4. RD discovery and other interface-independent components

This and the following sections define the required set of REST interfaces between a Resource Directory (RD), endpoints and lookup clients. Although the examples throughout these sections assume the use of CoAP [RFC7252], these REST interfaces can also be realized using HTTP [RFC7230]. Only multicast discovery operations are not possible on HTTP, and Simple Registration can not be executed as base attribute (which is mandatory for HTTP) can not be used there. In all definitions in these sections, both CoAP response codes (with dot notation) and HTTP response codes (without dot notation) are shown. An RD implementing this specification MUST support the discovery, registration, update, lookup, and removal interfaces.

All operations on the contents of the Resource Directory MUST be atomic and idempotent.

For several operations, interface templates are given in list form; those describe the operation participants, request codes, URIs, content formats and outcomes. Sections of those templates contain normative content about Interaction, Method, URI Template and URI Template Variables as well as the details of the Success condition. The additional sections on options like Content-Format and on Failure codes give typical cases that an implementation of the RD should deal with. Those serve to illustrate the typical responses to readers who are not yet familiar with all the details of CoAP based interfaces; they do not limit what a server may respond under atypical circumstances.

REST clients (registrant-EPs / CTs, lookup clients, RD servers during simple registrations) MUST be prepared to receive any unsuccessful code and act upon it according to its definition, options and/or payload to the best of their capabilities, falling back to failing the operation if recovery is not possible. In particular, they should retry the request upon 5.03 (Service Unavailable; 503 in HTTP) according to the Max-Age (Retry-After in HTTP) option, and fall back to link-format when receiving 4.15 (Unsupported Content Format; 415 in HTTP).

A resource directory MAY make the information submitted to it available to further directories, if it can ensure that a loop does not form. The protocol used between directories to ensure loop-free operation is outside the scope of this document.

4.1. Finding a Resource Directory

A (re-)starting device may want to find one or more resource directories for discovery purposes. Dependent on the operational conditions, one or more of the techniques below apply. The use of DNS-SD [RFC6763] is described in [I-D.ietf-core-rd-dns-sd].

The device may be pre-configured to exercise specific mechanisms for finding the resource directory:

1. It may be configured with a specific IP address for the RD. That IP address may also be an anycast address, allowing the network to forward RD requests to an RD that is topologically close; each target network environment in which some of these preconfigured nodes are to be brought up is then configured with a route for this anycast address that leads to an appropriate RD. (Instead of using an anycast address, a multicast address can also be

preconfigured. The RD servers then need to configure one of their interfaces with this multicast address.)

2. It may be configured with a DNS name for the RD and use DNS to return the IP address of the RD; it can find a DNS server to perform the lookup using the usual mechanisms for finding DNS servers.

For cases where the device is not specifically configured with a way to find a resource directory, the network may want to provide a suitable default.

1. If the address configuration of the network is performed via SLAAC, this is provided by the RDAO option Section 4.1.1.
2. If the address configuration of the network is performed via DHCP, this could be provided via a DHCP option (no such option is defined at the time of writing).

Finally, if neither the device nor the network offers any specific configuration, the device may want to employ heuristics to find a suitable resource directory.

The present specification does not fully define these heuristics, but suggests a number of candidates:

1. In a 6LoWPAN, just assume the Border Router (6LBR) can act as a resource directory (using the ABRO option to find that [RFC6775]). Confirmation can be obtained by sending a Unicast to "coap://[6LBR]/.well-known/core?rt=core.rd*".
2. In a network that supports multicast well, discovering the RD using a multicast query for /.well-known/core as specified in CoRE Link Format [RFC6690]: Sending a Multicast GET to "coap://[MCD1]/.well-known/core?rt=core.rd*". RDs within the multicast scope will answer the query.

When answering a multicast request directed at a link-local address, the RD may want to respond from a routable address; this makes it easier for registrants to use one of their own routable addresses for registration.

As some of the RD addresses obtained by the methods listed here are just (more or less educated) guesses, endpoints MUST make use of any error messages to very strictly rate-limit requests to candidate IP addresses that don't work out. For example, an ICMP Destination Unreachable message (and, in particular, the port unreachable code for this message) may indicate the lack of a CoAP server on the

candidate host, or a CoAP error response code such as 4.05 "Method Not Allowed" may indicate unwillingness of a CoAP server to act as a directory server.

The following RD discovery mechanisms are recommended:

- o In managed networks with border routers that need stand-alone operation, the RDAO option is recommended (e.g. operational phase described in Section 3.6).
- o In managed networks without border router (no Internet services available), the use of a preconfigured anycast address is recommended (e.g. installation phase described in Section 3.6).
- o The use of DNS facilities is described in [I-D.ietf-core-rd-dns-sd].

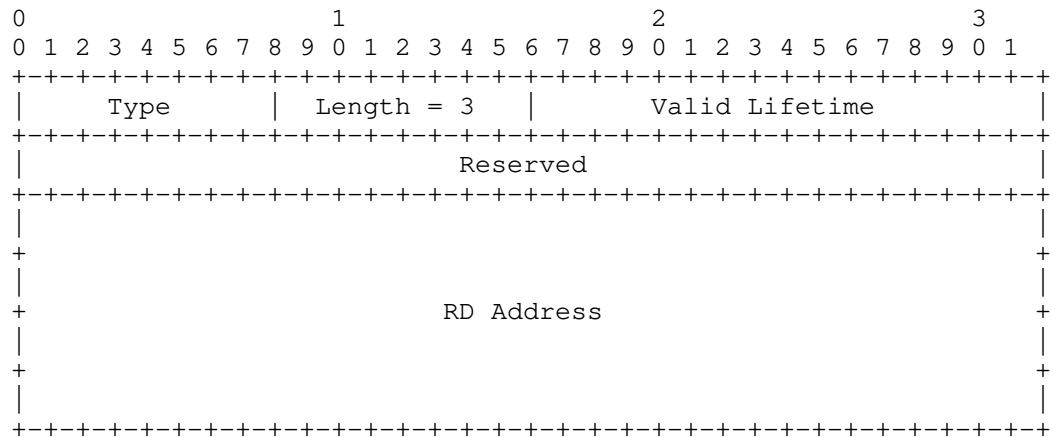
The use of multicast discovery in mesh networks is NOT recommended.

4.1.1. Resource Directory Address Option (RDAO)

The Resource Directory Address Option (RDAO) using IPv6 Neighbor Discovery (ND) carries information about the address of the Resource Directory (RD). This information is needed when endpoints cannot discover the Resource Directory with a link-local or realm-local scope multicast address, for instance because the endpoint and the RD are separated by a Border Router (6LBR). In many circumstances the availability of DHCP cannot be guaranteed either during commissioning of the network. The presence and the use of the RD is essential during commissioning.

It is possible to send multiple RDAO options in one message, indicating as many resource directory addresses.

The RDAO format is:



Fields:

| | |
|-----------------|---|
| Type: | 38 |
| Length: | 8-bit unsigned integer. The length of the option in units of 8 bytes. Always 3. |
| Valid Lifetime: | 16-bit unsigned integer. The length of time in units of 60 seconds (relative to the time the packet is received) that this Resource Directory address is valid. A value of all zero bits (0x0) indicates that this Resource Directory address is not valid anymore. |
| Reserved: | This field is unused. It MUST be initialized to zero by the sender and MUST be ignored by the receiver. |
| RD Address: | IPv6 address of the RD. |

Figure 4: Resource Directory Address Option

4.2. Payload Content Formats

Resource Directory implementations using this specification MUST support the application/link-format content format (ct=40).

Resource Directories implementing this specification MAY support additional content formats.

Any additional content format supported by a Resource Directory implementing this specification SHOULD be able to express all the information expressible in link-format. It MAY be able to express information that is inexpressible in link-format, but those expressions SHOULD be avoided where possible.

4.3. URI Discovery

Before an endpoint can make use of an RD, it must first know the RD's address and port, and the URI path information for its REST APIs. This section defines discovery of the RD and its URIs using the well-known interface of the CoRE Link Format [RFC6690]. A complete set of RD discovery methods is described in Section 4.1.

Discovery of the RD registration URI path is performed by sending either a multicast or unicast GET request to `"/.well-known/core"` and including a Resource Type (rt) parameter [RFC6690] with the value `"core.rd"` in the query string. Likewise, a Resource Type parameter value of `"core.rd-lookup*"` is used to discover the URIs for RD Lookup operations, `core.rd*` is used to discover all URI paths for RD operations. Upon success, the response will contain a payload with a link format entry for each RD function discovered, indicating the URI of the RD function returned and the corresponding Resource Type. When performing multicast discovery, the multicast IP address used will depend on the scope required and the multicast capabilities of the network (see Section 9.5).

A Resource Directory MAY provide hints about the content-formats it supports in the links it exposes or registers, using the `"ct"` target attribute, as shown in the example below. Clients MAY use these hints to select alternate content-formats for interaction with the Resource Directory.

HTTP does not support multicast and consequently only unicast discovery can be supported using HTTP. The well-known entry points SHOULD be provided to enable unicast discovery.

An implementation of this resource directory specification MUST support query filtering for the `rt` parameter as defined in [RFC6690].

While the link targets in this discovery step are often expressed in path-absolute form, this is not a requirement. Clients of the RD SHOULD therefore accept URIs of all schemes they support, both as URIs and relative references, and not limit the set of discovered URIs to those hosted at the address used for URI discovery.

The URI Discovery operation can yield multiple URIs of a given resource type. The client of the RD can use any of the discovered addresses initially.

The discovery request interface is specified as follows (this is exactly the Well-Known Interface of [RFC6690] Section 4, with the additional requirement that the server MUST support query filtering):

Interaction: EP and Client -> RD

Method: GET

URI Template: /.well-known/core{?rt}

URI Template Variables:

rt := Resource Type. SHOULD contain one of the values "core.rd", "core.rd-lookup*", "core.rd-lookup-res", "core.rd-lookup-ep", or "core.rd*"

Accept: absent, application/link-format or any other media type representing web links

The following response is expected on this interface:

Success: 2.05 "Content" or 200 "OK" with an application/link-format or other web link payload containing one or more matching entries for the RD resource.

The following example shows an endpoint discovering an RD using this interface, thus learning that the directory resource location, in this example, is /rd, and that the content-format delivered by the server hosting the resource is application/link-format (ct=40). Note that it is up to the RD to choose its RD locations.

Req: GET coap://[MCD1]/.well-known/core?rt=core.rd*

Res: 2.05 Content
</rd>;rt="core.rd";ct=40,
</rd-lookup/ep>;rt="core.rd-lookup-ep";ct=40,
</rd-lookup/res>;rt="core.rd-lookup-res";ct=40,

Figure 5: Example discovery exchange

The following example shows the way of indicating that a client may request alternate content-formats. The Content-Format code attribute "ct" MAY include a space-separated sequence of Content-Format codes as specified in Section 7.2.1 of [RFC7252], indicating that multiple

content-formats are available. The example below shows the required Content-Format 40 (application/link-format) indicated as well as a CBOR and JSON representation from [I-D.ietf-core-links-json] (which have no numeric values assigned yet, so they are shown as TBD64 and TBD504 as in that draft). The RD resource locations /rd, and /rd-lookup are example values. The server in this example also indicates that it is capable of providing observation on resource lookups.

```
Req: GET coap://[MCD1]/.well-known/core?rt=core.rd*
```

```
Res: 2.05 Content
```

```
</rd>;rt="core.rd";ct="40 65225",  
</rd-lookup/res>;rt="core.rd-lookup-res";ct="40 TBD64 TBD504";obs,  
</rd-lookup/ep>;rt="core.rd-lookup-ep";ct="40 TBD64 TBD504",
```

Figure 6: Example discovery exchange indicating additional content-formats

From a management and maintenance perspective, it is necessary to identify the components that constitute the RD server. The identification refers to information about for example client-server incompatibilities, supported features, required updates and other aspects. The URI discovery address, as described in section 4 of [RFC6690] can be used to find the identification.

It would typically be stored in an implementation information link (as described in [I-D.bormann-t2trg-rel-impl]):

```
Req: GET /.well-known/core?rel=impl-info
```

```
Res: 2.05 Content
```

```
<http://software.example.com/shiny-resource-directory/1.0beta1>;  
rel="impl-info"
```

Figure 7: Example exchange of obtaining implementation information

Note that depending on the particular server's architecture, such a link could be anchored at the RD server's root, at the discovery site (as in this example) or at individual RD components. The latter is to be expected when different applications are run on the same server.

5. Registration

After discovering the location of an RD, a registrant-ep or CT MAY register the resources of the registrant-ep using the registration interface. This interface accepts a POST from an endpoint containing the list of resources to be added to the directory as the message

payload in the CoRE Link Format [RFC6690] or other representations of web links, along with query parameters indicating the name of the endpoint, and optionally the sector, lifetime and base URI of the registration. It is expected that other specifications will define further parameters (see Section 9.3). The RD then creates a new registration resource in the RD and returns its location. The receiving endpoint MUST use that location when refreshing registrations using this interface. Registration resources in the RD are kept active for the period indicated by the lifetime parameter. The creating endpoint is responsible for refreshing the registration resource within this period using either the registration or update interface. The registration interface MUST be implemented to be idempotent, so that registering twice with the same endpoint parameters ep and d (sector) does not create multiple registration resources.

The following rules apply for a registration request targeting a given (ep, d) value pair:

- o When the (ep, d) value pair of the registration-request is different from any existing registration, a new registration is generated.
- o When the (ep, d) value pair of the registration-request is equal to an existing registration, the content and parameters of the existing registration are replaced with the content of the registration request.

The posted link-format document can (and typically does) contain relative references both in its link targets and in its anchors, or contain empty anchors. The RD server needs to resolve these references in order to faithfully represent them in lookups. They are resolved against the base URI of the registration, which is provided either explicitly in the "base" parameter or constructed implicitly from the requester's URI as constructed from its network address and scheme.

For media types to which Appendix C applies (i.e. documents in application/link-format), the RD only needs to accept representations in Limited Link Format as described there. Its behavior with representations outside that subset is implementation defined.

The registration request interface is specified as follows:

Interaction: EP -> RD

Method: POST

URI Template: `{+rd}{?ep,d,lt,base,extra-attrs*}`

URI Template Variables:

`rd` := RD registration URI (mandatory). This is the location of the RD, as obtained from discovery.

`ep` := Endpoint name (mostly mandatory). The endpoint name is an identifier that MUST be unique within a sector. As the endpoint name is a Unicode string, it is encoded in UTF-8 (and possibly pct-encoding) during variable expansion (see [RFC6570] Section 3.2.1). The endpoint name MUST NOT contain any character in the inclusive ranges 0-31 or 127-159. The maximum length of this parameter is 63 UTF-8 encoded bytes. If the RD is configured to recognize the endpoint (e.g. based on its security context), the RD assigns an endpoint name based on a set of configuration parameter values.

`d` := Sector (optional). The sector to which this endpoint belongs. When this parameter is not present, the RD MAY associate the endpoint with a configured default sector or leave it empty. The sector is encoded like the `ep` parameter, and is limited to 63 UTF-8 encoded bytes as well. The endpoint name and sector name are not set when one or both are set in an accompanying authorization token.

`lt` := Lifetime (optional). Lifetime of the registration in seconds. Range of 60-4294967295. If no lifetime is included in the initial registration, a default value of 90000 (25 hours) SHOULD be assumed.

`base` := Base URI (optional). This parameter sets the base URI of the registration, under which the relative links in the payload are to be interpreted. The specified URI typically does not have a path component of its own, and MUST be suitable as a base URI to resolve any relative references given in the registration. The parameter is therefore usually of the shape "scheme://authority" for HTTP and CoAP URIs. The URI SHOULD NOT have a query or fragment component as any non-empty relative part in a reference would remove those parts from the resulting URI.

In the absence of this parameter the scheme of the protocol, source address and source port of the registration request are assumed. The Base URI is consecutively constructed by concatenating the used protocol's scheme with the characters "://", the requester's source address as an address literal and

":" followed by its port (if it was not the protocol's default one) in analogy to [RFC7252] Section 6.5.

This parameter is mandatory when the directory is filled by a third party such as an commissioning tool.

If the registrant-ep uses an ephemeral port to register with, it MUST include the base parameter in the registration to provide a valid network path.

A registrant that can not be reached by potential lookup clients at the address it registers from (e.g. because it is behind some form of Network Address Translation (NAT)) MUST provide a reachable base address with its registration.

If the Base URI contains a link-local IP literal, it MUST NOT contain a Zone Identifier, and MUST be local to the link on which the registration request is received.

Endpoints that register with a base that contains a path component can not meaningfully use [RFC6690] Link Format due to its prevalence of the Origin concept in relative reference resolution. Those applications should use different representations of links to which Appendix C is not applicable (e.g. [I-D.hartke-t2trg-coral]).

extra-attrs := Additional registration attributes (optional).
The endpoint can pass any parameter registered at Section 9.3 to the directory. If the RD is aware of the parameter's specified semantics, it processes it accordingly. Otherwise, it MUST store the unknown key and its value(s) as an endpoint attribute for further lookup.

Content-Format: application/link-format or any other indicated media type representing web links

The following response is expected on this interface:

Success: 2.01 "Created" or 201 "Created". The Location-Path option or Location header MUST be included in the response. This location MUST be a stable identifier generated by the RD as it is used for all subsequent operations on this registration resource. The registration resource location thus returned is for the purpose of updating the lifetime of the registration and for maintaining the content of the registered links, including updating and deleting links.

A registration with an already registered ep and d value pair responds with the same success code and location as the original registration; the set of links registered with the endpoint is replaced with the links from the payload.

The location MUST NOT have a query or fragment component, as that could conflict with query parameters during the Registration Update operation. Therefore, the Location-Query option MUST NOT be present in a successful response.

If the registration fails, including request timeouts, or if delays from Service Unavailable responses with Max-Age or Retry-After accumulate to exceed the registrant's configured timeouts, it SHOULD pick another registration URI from the "URI Discovery" step and if there is only one or the list is exhausted, pick other choices from the "Finding a Resource Directory" step. Care has to be taken to consider the freshness of results obtained earlier, e.g. of the result of a "/.well-known/core" response, the lifetime of an RDAO option and of DNS responses. Any rate limits and persistent errors from the "Finding a Resource Directory" step must be considered for the whole registration time, not only for a single operation.

The following example shows a registrant-ep with the name "node1" registering two resources to an RD using this interface. The location "/rd" is an example RD location discovered in a request similar to Figure 5.

```
Req: POST coap://rd.example.com/rd?ep=node1
Content-Format: 40
Payload:
</sensors/temp>;ct=41;rt="temperature-c";if="sensor",
<http://www.example.com/sensors/temp>;
  anchor="/sensors/temp";rel="describedby"

Res: 2.01 Created
Location-Path: /rd/4521
```

Figure 8: Example registration payload

A Resource Directory may optionally support HTTP. Here is an example of almost the same registration operation above, when done using HTTP.

```
Req: POST /rd?ep=nodel&base=http://[2001:db8:1::1] HTTP/1.1
Host: example.com
Content-Type: application/link-format
Payload:
</sensors/temp>;ct=41;rt="temperature-c";if="sensor",
<http://www.example.com/sensors/temp>;
  anchor="/sensors/temp";rel="describedby"

Res: 201 Created
Location: /rd/4521
```

Figure 9: Example registration payload as expressed using HTTP

5.1. Simple Registration

Not all endpoints hosting resources are expected to know how to upload links to an RD as described in Section 5. Instead, simple endpoints can implement the Simple Registration approach described in this section. An RD implementing this specification **MUST** implement Simple Registration. However, there may be security reasons why this form of directory discovery would be disabled.

This approach requires that the registrant-ep makes available the hosted resources that it wants to be discovered, as links on its `"/.well-known/core"` interface as specified in [RFC6690]. The links in that document are subject to the same limitations as the payload of a registration (with respect to Appendix C).

- o The registrant-ep finds one or more addresses of the directory server as described in Section 4.1.
- o The registrant-ep sends (and regularly refreshes with) a POST request to the `"/.well-known/core"` URI of the directory server of choice. The body of the POST request is empty, and triggers the resource directory server to perform GET requests at the requesting registrant-ep's `/.well-known/core` to obtain the link-format payload to register.

The registrant-ep includes the same registration parameters in the POST request as it would per Section 5. The registration base URI of the registration is taken from the registrant-ep's network address (as is default with regular registrations).

Example request from registrant-EP to RD (unanswered until the next step):

Req: POST /.well-known/core?lt=6000&ep=node1
(No payload)

Figure 10: First half example exchange of a simple registration

- o The Resource Directory queries the registrant-ep's discovery resource to determine the success of the operation. It SHOULD keep a cache of the discovery resource and not query it again as long as it is fresh.

Example request from the RD to the registrant-EP:

Req: GET /.well-known/core
Accept: 40

Res: 2.05 Content
Content-Format: 40
Payload:
</sen/temp>

Figure 11: Example exchange of the RD querying the simple endpoint

With this response, the RD would answer the previous step's request:

Res: 2.04 Changed

Figure 12: Second half example exchange of a simple registration

The sequence of fetching the registration content before sending a successful response was chosen to make responses reliable, and the caching item was chosen to still allow very constrained registrants. Registrants MUST be able to serve a GET request to "/.well-known/core" after having requested registration. Constrained devices MAY regard the initial request as temporarily failed when they need RAM occupied by their own request to serve the RD's GET, and retry later when the RD already has a cached representation of their discovery resources. Then, the RD can reply immediately and the registrant can receive the response.

The simple registration request interface is specified as follows:

Interaction: EP -> RD

Method: POST

URI Template: /.well-known/core{?ep,d,lt,extra-attrs*}

URI Template Variables are as they are for registration in Section 5. The base attribute is not accepted to keep the registration interface simple; that rules out registration over CoAP-over-TCP or HTTP that would need to specify one.

The following response is expected on this interface:

Success: 2.04 "Changed".

For the second interaction triggered by the above, the registrant-ep takes the role of server and the RD the role of client. (Note that this is exactly the Well-Known Interface of [RFC6690] Section 4):

Interaction: RD -> EP

Method: GET

URI Template: /.well-known/core

The following response is expected on this interface:

Success: 2.05 "Content".

The RD MUST delete registrations created by simple registration after the expiration of their lifetime. Additional operations on the registration resource cannot be executed because no registration location is returned.

5.2. Third-party registration

For some applications, even Simple Registration may be too taxing for some very constrained devices, in particular if the security requirements become too onerous.

In a controlled environment (e.g. building control), the Resource Directory can be filled by a third party device, called a Commissioning Tool (CT). The commissioning tool can fill the Resource Directory from a database or other means. For that purpose scheme, IP address and port of the URI of the registered device is the value of the "base" parameter of the registration described in Section 5.

It should be noted that the value of the "base" parameter applies to all the links of the registration and has consequences for the anchor value of the individual links as exemplified in Appendix B. An eventual (currently non-existing) "base" attribute of the link is not affected by the value of "base" parameter in the registration.

5.3. Operations on the Registration Resource

This section describes how the registering endpoint can maintain the registrations that it created. The registering endpoint can be the registrant-ep or the CT. An endpoint SHOULD NOT use this interface for registrations that it did not create. The registrations are resources of the RD.

After the initial registration, the registering endpoint retains the returned location of the Registration Resource for further operations, including refreshing the registration in order to extend the lifetime and "keep-alive" the registration. When the lifetime of the registration has expired, the RD SHOULD NOT respond to discovery queries concerning this endpoint. The RD SHOULD continue to provide access to the Registration Resource after a registration time-out occurs in order to enable the registering endpoint to eventually refresh the registration. The RD MAY eventually remove the registration resource for the purpose of garbage collection. If the Registration Resource is removed, the corresponding endpoint will need to be re-registered.

The Registration Resource may also be used cancel the registration using DELETE, and to perform further operations beyond the scope of this specification.

These operations are described below.

5.3.1. Registration Update

The update interface is used by the registering endpoint to refresh or update its registration with an RD. To use the interface, the registering endpoint sends a POST request to the registration resource returned by the initial registration operation.

An update MAY update the lifetime or the base URI registration parameters "lt", "base" as in Section 5. Parameters that are not being changed SHOULD NOT be included in an update. Adding parameters that have not changed increases the size of the message but does not have any other implications. Parameters MUST be included as query parameters in an update operation as in Section 5.

A registration update resets the timeout of the registration to the (possibly updated) lifetime of the registration, independent of whether a "lt" parameter was given.

If the base URI of the registration is changed in an update, relative references submitted in the original registration or later updates are resolved anew against the new base.

The registration update operation only describes the use of POST with an empty payload. Future standards might describe the semantics of using content formats and payloads with the POST method to update the links of a registration (see Section 5.3.3).

The update registration request interface is specified as follows:

Interaction: EP -> RD

Method: POST

URI Template: {+location}{?lt,base,extra-attrs*}

URI Template Variables:

location := This is the Location returned by the RD as a result of a successful earlier registration.

lt := Lifetime (optional). Lifetime of the registration in seconds. Range of 60-4294967295. If no lifetime is included, the previous last lifetime set on a previous update or the original registration (falling back to 90000) SHOULD be used.

base := Base URI (optional). This parameter updates the Base URI established in the original registration to a new value. If the parameter is set in an update, it is stored by the RD as the new Base URI under which to interpret the relative links present in the payload of the original registration, following the same restrictions as in the registration. If the parameter is not set in the request but was set before, the previous Base URI value is kept unmodified. If the parameter is not set in the request and was not set before either, the source address and source port of the update request are stored as the Base URI.

extra-attrs := Additional registration attributes (optional). As with the registration, the RD processes them if it knows their semantics. Otherwise, unknown attributes are stored as endpoint attributes, overriding any previously stored endpoint attributes of the same key.

Note that this default behavior does not allow removing an endpoint attribute in an update. For attributes whose functionality depends on the endpoints' ability to remove them in an update, it can make sense to define a value whose presence is equivalent to the absence of a value. As an alternative, an extension can define different updating rules for their attributes. That necessitates either discovery of

whether the RD is aware of that extension, or tolerating the default behavior.

Content-Format: none (no payload)

The following responses are expected on this interface:

Success: 2.04 "Changed" or 204 "No Content" if the update was successfully processed.

Failure: 4.04 "Not Found" or 404 "Not Found". Registration does not exist (e.g. may have been removed).

If the registration fails in any way, including "Not Found" and request timeouts, or if the time indicated in a Service Unavailable Max-Age/Retry-After exceeds the remaining lifetime, the registering endpoint SHOULD attempt registration again.

The following example shows how the registering endpoint updates its registration resource at an RD using this interface with the example location value: /rd/4521.

Req: POST /rd/4521

Res: 2.04 Changed

Figure 13: Example update of a registration

The following example shows the registering endpoint updating its registration resource at an RD using this interface with the example location value: /rd/4521. The initial registration by the registering endpoint set the following values:

- o endpoint name (ep)=endpoint1
- o lifetime (lt)=500
- o Base URI (base)=coap://local-proxy-old.example.com:5683
- o payload of Figure 8

The initial state of the Resource Directory is reflected in the following request:

```
Req: GET /rd-lookup/res?ep=endpoint1

Res: 2.01 Content
Payload:
<coap://local-proxy-old.example.com:5683/sensors/temp>;ct=41;
  rt="temperature-c";if="sensor";
  anchor="coap://local-proxy-old.example.com:5683/",
<http://www.example.com/sensors/temp>;
  anchor="coap://local-proxy-old.example.com:5683/sensors/temp";rel="described
by"
```

Figure 14: Example lookup before a change to the base address

The following example shows the registering endpoint changing the Base URI to "coaps://new.example.com:5684":

```
Req: POST /rd/4521?base=coaps://new.example.com:5684

Res: 2.04 Changed
```

Figure 15: Example registration update that changes the base address

The consecutive query returns:

```
Req: GET /rd-lookup/res?ep=endpoint1

Res: 2.01 Content
Payload:
<coap://new.example.com:5684/sensors/temp>;ct=41;
  rt="temperature-c";if="sensor";
  anchor="coap://new.example.com:5684/",
<http://www.example.com/sensors/temp>;
  anchor="coap://new.example.com:5684/sensors/temp";rel="describedby"
```

Figure 16: Example lookup after a change to the base address

5.3.2. Registration Removal

Although RD registrations have soft state and will eventually timeout after their lifetime, the registering endpoint SHOULD explicitly remove an entry from the RD if it knows it will no longer be available (for example on shut-down). This is accomplished using a removal interface on the RD by performing a DELETE on the endpoint resource.

The removal request interface is specified as follows:

Interaction: EP -> RD

Method: DELETE

URI Template: {+location}

URI Template Variables:

location := This is the Location returned by the RD as a result of a successful earlier registration.

The following responses are expected on this interface:

Success: 2.02 "Deleted" or 204 "No Content" upon successful deletion

Failure: 4.04 "Not Found" or 404 "Not Found". Registration does not exist (e.g. may already have been removed).

The following examples shows successful removal of the endpoint from the RD with example location value /rd/4521.

Req: DELETE /rd/4521

Res: 2.02 Deleted

Figure 17: Example of a registration removal

5.3.3. Further operations

Additional operations on the registration can be specified in future documents, for example:

- o Send iPATCH (or PATCH) updates ([RFC8132]) to add, remove or change the links of a registration.
- o Use GET to read the currently stored set of links in a registration resource.

Those operations are out of scope of this document, and will require media types suitable for modifying sets of links.

6. RD Lookup

To discover the resources registered with the RD, a lookup interface must be provided. This lookup interface is defined as a default, and it is assumed that RDs may also support lookups to return resource descriptions in alternative formats (e.g. JSON or CBOR link format [I-D.ietf-core-links-json]) or using more advanced interfaces (e.g. supporting context or semantic based lookup) on different resources that are discovered independently.

RD Lookup allows lookups for endpoints and resources using attributes defined in this document and for use with the CoRE Link Format. The result of a lookup request is the list of links (if any) corresponding to the type of lookup. Thus, an endpoint lookup **MUST** return a list of endpoints and a resource lookup **MUST** return a list of links to resources.

The lookup type is selected by a URI endpoint, which is indicated by a Resource Type as per Table 1 below:

| Lookup Type | Resource Type | Mandatory |
|-------------|--------------------|-----------|
| Resource | core.rd-lookup-res | Mandatory |
| Endpoint | core.rd-lookup-ep | Mandatory |

Table 1: Lookup Types

6.1. Resource lookup

Resource lookup results in links that are semantically equivalent to the links submitted to the RD. The links and link parameters returned by the lookup are equal to the submitted ones, except that the target and anchor references are fully resolved.

Links that did not have an anchor attribute are therefore returned with the base URI of the registration as the anchor. Links of which href or anchor was submitted as a (full) URI are returned with these attributes unmodified.

Above rules allow the client to interpret the response as links without any further knowledge of the storage conventions of the RD. The Resource Directory **MAY** replace the registration base URIs with a configured intermediate proxy, e.g. in the case of an HTTP lookup interface for CoAP endpoints.

If the base URI of a registration contains a link-local address, the RD **MUST NOT** show its links unless the lookup was made from the same link. The RD **MUST NOT** include zone identifiers in the resolved URIs.

6.2. Lookup filtering

Using the Accept Option, the requester can control whether the returned list is returned in CoRE Link Format ("application/link-format", default) or in alternate content-formats (e.g. from [I-D.ietf-core-links-json]).

The page and count parameters are used to obtain lookup results in specified increments using pagination, where count specifies how many links to return and page specifies which subset of links organized in sequential pages, each containing 'count' links, starting with link zero and page zero. Thus, specifying count of 10 and page of 0 will return the first 10 links in the result set (links 0-9). Count = 10 and page = 1 will return the next 'page' containing links 10-19, and so on.

Multiple search criteria MAY be included in a lookup. All included criteria MUST match for a link to be returned. The Resource Directory MUST support matching with multiple search criteria.

A link matches a search criterion if it has an attribute of the same name and the same value, allowing for a trailing "*" wildcard operator as in Section 4.1 of [RFC6690]. Attributes that are defined as "link-type" match if the search value matches any of their values (see Section 4.1 of [RFC6690]; e.g. "?if=core.s" matches ";if="abc core.s";"). A resource link also matches a search criterion if its endpoint would match the criterion, and viceversa, an endpoint link matches a search criterion if any of its resource links matches it.

Note that "href" is a valid search criterion and matches target references. Like all search criteria, on a resource lookup it can match the target reference of the resource link itself, but also the registration resource of the endpoint that registered it. Queries for resource link targets MUST be in URI form (i.e. not relative references) and are matched against a resolved link target. Queries for endpoints SHOULD be expressed in path-absolute form if possible and MUST be expressed in URI form otherwise; the RD SHOULD recognize either.

Endpoints that are interested in a lookup result repeatedly or continuously can use mechanisms like ETag caching, resource observation ([RFC7641]), or any future mechanism that might allow more efficient observations of collections. These are advertised, detected and used according to their own specifications and can be used with the lookup interface as with any other resource.

When resource observation is used, every time the set of matching links changes, or the content of a matching link changes, the RD sends a notification with the matching link set. The notification contains the successful current response to the given request, especially with respect to representing zero matching links (see "Success" item below).

The lookup interface is specified as follows:

Interaction: Client -> RD

Method: GET

URI Template: {+type-lookup-location}{?page,count,search*}

URI Template Variables:

type-lookup-location := RD Lookup URI for a given lookup type (mandatory). The address is discovered as described in Section 4.3.

search := Search criteria for limiting the number of results (optional).

page := Page (optional). Parameter cannot be used without the count parameter. Results are returned from result set in pages that contain 'count' links starting from index (page * count). Page numbering starts with zero.

count := Count (optional). Number of results is limited to this parameter value. If the page parameter is also present, the response MUST only include 'count' links starting with the (page * count) link in the result set from the query. If the count parameter is not present, then the response MUST return all matching links in the result set. Link numbering starts with zero.

Accept: absent, application/link-format or any other indicated media type representing web links

The following responses codes are defined for this interface:

Success: 2.05 "Content" or 200 "OK" with an "application/link-format" or other web link payload containing matching entries for the lookup. The payload can contain zero links (which is an empty payload in [RFC6690] link format, but could also be "[]" in JSON based formats), indicating that no entities matched the request.

6.3. Resource lookup examples

The examples in this section assume the existence of CoAP hosts with a default CoAP port 61616. HTTP hosts are possible and do not change the nature of the examples.

The following example shows a client performing a resource lookup with the example resource look-up locations discovered in Figure 5:

```
Req: GET /rd-lookup/res?rt=temperature

Res: 2.05 Content
<coap://[2001:db8:3::123]:61616/temp>;rt="temperature";
    anchor="coap://[2001:db8:3::123]:61616"
```

Figure 18: Example a resource lookup

A client that wants to be notified of new resources as they show up can use observation:

```
Req: GET /rd-lookup/res?rt=light
Observe: 0

Res: 2.05 Content
Observe: 23
Payload: empty

(at a later point in time)

Res: 2.05 Content
Observe: 24
Payload:
<coap://[2001:db8:3::124]/west>;rt="light";
    anchor="coap://[2001:db8:3::124] ",
<coap://[2001:db8:3::124]/south>;rt="light";
    anchor="coap://[2001:db8:3::124] ",
<coap://[2001:db8:3::124]/east>;rt="light";
    anchor="coap://[2001:db8:3::124] "
```

Figure 19: Example an observing resource lookup

The following example shows a client performing a paginated resource lookup

Req: GET /rd-lookup/res?page=0&count=5

Res: 2.05 Content

```
<coap://[2001:db8:3::123]:61616/res/0>;rt=sensor;ct=60;
  anchor="coap://[2001:db8:3::123]:61616",
<coap://[2001:db8:3::123]:61616/res/1>;rt=sensor;ct=60;
  anchor="coap://[2001:db8:3::123]:61616",
<coap://[2001:db8:3::123]:61616/res/2>;rt=sensor;ct=60;
  anchor="coap://[2001:db8:3::123]:61616",
<coap://[2001:db8:3::123]:61616/res/3>;rt=sensor;ct=60;
  anchor="coap://[2001:db8:3::123]:61616",
<coap://[2001:db8:3::123]:61616/res/4>;rt=sensor;ct=60;
  anchor="coap://[2001:db8:3::123]:61616"
```

Req: GET /rd-lookup/res?page=1&count=5

Res: 2.05 Content

```
<coap://[2001:db8:3::123]:61616/res/5>;rt=sensor;ct=60;
  anchor="coap://[2001:db8:3::123]:61616",
<coap://[2001:db8:3::123]:61616/res/6>;rt=sensor;ct=60;
  anchor="coap://[2001:db8:3::123]:61616",
<coap://[2001:db8:3::123]:61616/res/7>;rt=sensor;ct=60;
  anchor="coap://[2001:db8:3::123]:61616",
<coap://[2001:db8:3::123]:61616/res/8>;rt=sensor;ct=60;
  anchor="coap://[2001:db8:3::123]:61616",
<coap://[2001:db8:3::123]:61616/res/9>;rt=sensor;ct=60;
  anchor="coap://[2001:db8:3::123]:61616"
```

Figure 20: Examples of paginated resource lookup

The following example shows a client performing a lookup of all resources of all endpoints of a given endpoint type. It assumes that two endpoints (with endpoint names "sensor1" and "sensor2") have previously registered with their respective addresses "coap://sensor1.example.com" and "coap://sensor2.example.com", and posted the very payload of the 6th request of section 5 of [RFC6690].

It demonstrates how absolute link targets stay unmodified, while relative ones are resolved:

```

Req: GET /rd-lookup/res?et=oic.d.sensor

<coap://sensor1.example.com/sensors>;ct=40;title="Sensor Index";
  anchor="coap://sensor1.example.com",
<coap://sensor1.example.com/sensors/temp>;rt="temperature-c";
  if="sensor"; anchor="coap://sensor1.example.com",
<coap://sensor1.example.com/sensors/light>;rt="light-lux";
  if="sensor"; anchor="coap://sensor1.example.com",
<http://www.example.com/sensors/t123>;rel="describedby";
  anchor="coap://sensor1.example.com/sensors/temp",
<coap://sensor1.example.com/t>;rel="alternate";
  anchor="coap://sensor1.example.com/sensors/temp",
<coap://sensor2.example.com/sensors>;ct=40;title="Sensor Index";
  anchor="coap://sensor2.example.com",
<coap://sensor2.example.com/sensors/temp>;rt="temperature-c";
  if="sensor"; anchor="coap://sensor2.example.com",
<coap://sensor2.example.com/sensors/light>;rt="light-lux";
  if="sensor"; anchor="coap://sensor2.example.com",
<http://www.example.com/sensors/t123>;rel="describedby";
  anchor="coap://sensor2.example.com/sensors/temp",
<coap://sensor2.example.com/t>;rel="alternate";
  anchor="coap://sensor2.example.com/sensors/temp"

```

Figure 21: Example of resource lookup from multiple endpoints

6.4. Endpoint lookup

The endpoint lookup returns registration resources which can only be manipulated by the registering endpoint.

Endpoint registration resources are annotated with their endpoint names (ep), sectors (d, if present) and registration base URI (base; reports the registrant-ep's address if no explicit base was given) as well as a constant resource type (rt="core.rd-ep"); the lifetime (lt) is not reported. Additional endpoint attributes are added as target attributes to their endpoint link unless their specification says otherwise.

Links to endpoints SHOULD be presented in path-absolute form or, if required, as absolute references. (This avoids the RFC6690 ambiguities.)

Base addresses that contain link-local addresses MUST NOT include zone identifiers, and such registrations MUST NOT be shown unless the lookup was made from the same link from which the registration was made.

While Endpoint Lookup does expose the registration resources, the RD does not need to make them accessible to clients. Clients SHOULD NOT attempt to dereference or manipulate them.

A Resource Directory can report endpoints in lookup that are not hosted at the same address. Lookup clients MUST be prepared to see arbitrary URIs as registration resources in the results and treat them as opaque identifiers; the precise semantics of such links are left to future specifications.

The following example shows a client performing an endpoint type (et) lookup with the value oic.d.sensor (which is currently a registered rt value):

```
Req: GET /rd-lookup/ep?et=oic.d.sensor
```

```
Res: 2.05 Content
```

```
</rd/1234>;base="coap://[2001:db8:3::127]:61616";ep="node5";  
et="oic.d.sensor";ct="40";rt="core.rd-ep",  
</rd/4521>;base="coap://[2001:db8:3::129]:61616";ep="node7";  
et="oic.d.sensor";ct="40";d="floor-3";rt="core.rd-ep"
```

Figure 22: Examples of endpoint lookup

7. Security policies

The Resource Directory (RD) provides assistance to applications situated on a selection of nodes to discover endpoints on connected nodes. This section discusses different security aspects of accessing the RD.

The contents of the RD are inserted in two ways:

1. The node hosting the discoverable endpoint fills the RD with the contents of /.well-known/core by:
 - * Storing the contents directly into RD (see Section 5)
 - * Requesting the RD to load the contents from /.well-known/core (see Section 5.1)
2. A Commissioning Tool (CT) fills the RD with endpoint information for a set of discoverable nodes. (see Section 5 with base=authority parameter value)

In both cases, the nodes filling the RD should be authenticated and authorized to change the contents of the RD. An Authorization Server (AS) is responsible to assign a token to the registering node to

authorize the node to discover or register endpoints in a given RD [I-D.ietf-ace-oauth-authz].

It can be imagined that an installation is divided in a set of security regions, each one with its own RD(s) to discover the endpoints that are part of a given security region. An endpoint that wants to discover an RD, responsible for a given region, needs to be authorized to learn the contents of a given RD. Within a region, for a given RD, a more fine-grained security division is possible based on the values of the endpoint registration parameters. Authorization to discover endpoints with a given set of filter values is recommended for those cases.

When a node registers its endpoints, criteria are needed to authorize the node to enter them. An important aspect is the uniqueness of the (endpoint name, and optional sector) pair within the RD. Consider the two cases separately: (1) CT registers endpoints, and (2) the registering node registers its own endpoint(s).

- o A CT needs authorization to register a set of endpoints. This authorization can be based on the region, i.e. a given CT is authorized to register any endpoint (endpoint name, sector) into a given RD, or to register an endpoint with (endpoint name, sector) value pairs assigned by the AS, or can be more fine-grained, including a subset of registration parameter values.
- o A given endpoint that registers itself, needs to prove its possession of its unique (endpoint name, sector) value pair. Alternatively, the AS can authorize the endpoint to register with an (endpoint name, sector) value pair assigned by the AS.

A separate document needs to specify these aspects to ensure interoperability between registering nodes and RD. The subsections below give some hints how to handle a subset of the different aspects.

7.1. Secure RD discovery

The Resource Server (RS) discussed in [I-D.ietf-ace-oauth-authz] is equated to the RD. The client (C) needs to discover the RD as discussed in Section 4.1. C can discover the related AS by sending a request to the RD. The RD denies the request by sending the address of the related AS, as discussed in section 5.1 of [I-D.ietf-ace-oauth-authz]. The client MUST send an authorization request to the AS. When appropriate, the AS returns a token that specifies the authorization permission which needs to be specified in a separate document.

7.2. Secure RD filtering

The authorized parameter values for the queries by a given endpoint must be registered by the AS. The AS communicates the parameter values in the token. A separate document needs to specify the parameter value combinations and their storage in the token. The RD decodes the token and checks the validity of the queries of the client.

7.3. Secure endpoint Name assignment

This section only considers the assignment of a name to the endpoint based on an automatic mechanism without use of AS. More elaborate protocols are out of scope. The registering endpoint is authorized by the AS to discover the RD and add registrations. A token is provided by the AS and communicated from registering endpoint to RD. It is assumed that DTLS is used to secure the channel between registering endpoint and RD, where the registering endpoint is the DTLS client. Assuming that the client is provided by a certificate at manufacturing time, the certificate is uniquely identified by the CN field and the serial number. The RD can assign a unique endpoint name by using the certificate identifier as endpoint name. Proof of possession of the endpoint name by the registering endpoint is checked by encrypting the certificate identifier with the private key of the registering endpoint, which the RD can decrypt with the public key stored in the certificate. Even simpler, the authorized registering endpoint can generate a random number (or string) that identifies the endpoint. The RD can check for the improbable replication of the random value. The RD MUST check that registering endpoint uses only one random value for each authorized endpoint.

8. Security Considerations

The security considerations as described in Section 5 of [RFC8288] and Section 6 of [RFC6690] apply. The `"/.well-known/core"` resource may be protected e.g. using DTLS when hosted on a CoAP server as described in [RFC7252]. DTLS or TLS based security SHOULD be used on all resource directory interfaces defined in this document.

8.1. Endpoint Identification and Authentication

An Endpoint (name, sector) pair is unique within the set of endpoints registered by the RD. An Endpoint MUST NOT be identified by its protocol, port or IP address as these may change over the lifetime of an Endpoint.

Every operation performed by an Endpoint on a resource directory SHOULD be mutually authenticated using Pre-Shared Key, Raw Public Key or Certificate based security.

Consider the following threat: two devices A and B are registered at a single server. Both devices have unique, per-device credentials for use with DTLS to make sure that only parties with authorization to access A or B can do so.

Now, imagine that a malicious device A wants to sabotage the device B. It uses its credentials during the DTLS exchange. Then, it specifies the endpoint name of device B as the name of its own endpoint in device A. If the server does not check whether the identifier provided in the DTLS handshake matches the identifier used at the CoAP layer then it may be inclined to use the endpoint name for looking up what information to provision to the malicious device.

Section 7.3 specifies an example that removes this threat for endpoints that have a certificate installed.

8.2. Access Control

Access control SHOULD be performed separately for the RD registration and Lookup API paths, as different endpoints may be authorized to register with an RD from those authorized to lookup endpoints from the RD. Such access control SHOULD be performed in as fine-grained a level as possible. For example access control for lookups could be performed either at the sector, endpoint or resource level.

8.3. Denial of Service Attacks

Services that run over UDP unprotected are vulnerable to unknowingly become part of a DDoS attack as UDP does not require return routability check. Therefore, an attacker can easily spoof the source IP of the target entity and send requests to such a service which would then respond to the target entity. This can be used for large-scale DDoS attacks on the target. Especially, if the service returns a response that is order of magnitudes larger than the request, the situation becomes even worse as now the attack can be amplified. DNS servers have been widely used for DDoS amplification attacks. There is also a danger that NTP Servers could become implicated in denial-of-service (DoS) attacks since they run on unprotected UDP, there is no return routability check, and they can have a large amplification factor. The responses from the NTP server were found to be 19 times larger than the request. A Resource Directory (RD) which responds to wild-card lookups is potentially vulnerable if run with CoAP over UDP. Since there is no return routability check and the responses can be significantly larger than

requests, RDs can unknowingly become part of a DDoS amplification attack.

9. IANA Considerations

9.1. Resource Types

IANA is asked to enter the following values into the Resource Type (rt=) Link Target Attribute Values sub-registry of the Constrained Restful Environments (CoRE) Parameters registry defined in [RFC6690]:

| Value | Description | Reference |
|--------------------|-----------------------------|---------------------|
| core.rd | Directory resource of an RD | RFCTHIS Section 4.3 |
| core.rd-lookup-res | Resource lookup of an RD | RFCTHIS Section 4.3 |
| core.rd-lookup-ep | Endpoint lookup of an RD | RFCTHIS Section 4.3 |
| core.rd-ep | Endpoint resource of an RD | RFCTHIS Section 6 |

9.2. IPv6 ND Resource Directory Address Option

This document registers one new ND option type under the sub-registry "IPv6 Neighbor Discovery Option Formats":

- o Resource Directory Address Option (38)

9.3. RD Parameter Registry

This specification defines a new sub-registry for registration and lookup parameters called "RD Parameters" under "CoRE Parameters". Although this specification defines a basic set of parameters, it is expected that other standards that make use of this interface will define new ones.

Each entry in the registry must include

- o the human readable name of the parameter,
- o the short name as used in query parameters or target attributes,
- o indication of whether it can be passed as a query parameter at registration of endpoints, as a query parameter in lookups, or be expressed as a target attribute,

- o validity requirements if any, and
- o a description.

The query parameter MUST be both a valid URI query key [RFC3986] and a token as used in [RFC8288].

The description must give details on whether the parameter can be updated, and how it is to be processed in lookups.

The mechanisms around new RD parameters should be designed in such a way that they tolerate RD implementations that are unaware of the parameter and expose any parameter passed at registration or updates on in endpoint lookups. (For example, if a parameter used at registration were to be confidential, the registering endpoint should be instructed to only set that parameter if the RD advertises support for keeping it confidential at the discovery step.)

Initial entries in this sub-registry are as follows:

| Full name | Short | Validity | Use | Description |
|-----------------------|------------|--------------------------|----------|---|
| Endpoint Name | ep | Unicode* | RLA | Name of the endpoint |
| Lifetime | lt | 60-4294967295 | R | Lifetime of the registration in seconds |
| Sector | d | Unicode* | RLA | Sector to which this endpoint belongs |
| Registration Base URI | base | URI | RLA | The scheme, address and port and path at which this server is available |
| Page Count | page count | Integer | L | Used for pagination |
| Endpoint Type | et | Integer Section 9.3.1 | L RLA | Used for pagination Semantic type of the endpoint (see Section 9.4) |

Table 2: RD Parameters

(Short: Short name used in query parameters or target attributes.
 Validity: Unicode* = 63 Bytes of UTF-8 encoded Unicode, with no control characters as per Section 5. Use: R = used at registration, L = used at lookup, A = expressed in target attribute

The descriptions for the options defined in this document are only summarized here. To which registrations they apply and when they are to be shown is described in the respective sections of this document.

The IANA policy for future additions to the sub-registry is "Expert Review" as described in [RFC8126]. The evaluation should consider formal criteria, duplication of functionality (Is the new entry redundant with an existing one?), topical suitability (E.g. is the described property actually a property of the endpoint and not a property of a particular resource, in which case it should go into the payload of the registration and need not be registered?), and the potential for conflict with commonly used target attributes (For example, "if" could be used as a parameter for conditional registration if it were not to be used in lookup or attributes, but would make a bad parameter for lookup, because a resource lookup with an "if" query parameter could ambiguously filter by the registered endpoint property or the [RFC6690] target attribute). It is expected that the registry will receive between 5 and 50 registrations in total over the next years.

9.3.1. Full description of the "Endpoint Type" Registration Parameter

An endpoint registering at an RD can describe itself with endpoint types, similar to how resources are described with Resource Types in [RFC6690]. An endpoint type is expressed as a string, which can be either a URI or one of the values defined in the Endpoint Type sub-registry. Endpoint types can be passed in the "et" query parameter as part of extra-attrs at the Registration step, are shown on endpoint lookups using the "et" target attribute, and can be filtered for using "et" as a search criterion in resource and endpoint lookup. Multiple endpoint types are given as separate query parameters or link attributes.

Note that Endpoint Type differs from Resource Type in that it uses multiple attributes rather than space separated values. As a result, Resource Directory implementations automatically support correct filtering in the lookup interfaces from the rules for unknown endpoint attributes.

9.4. "Endpoint Type" (et=) RD Parameter values

This specification establishes a new sub-registry under "CoRE Parameters" called '"Endpoint Type" (et=) RD Parameter values'. The registry properties (required policy, requirements, template) are identical to those of the Resource Type parameters in [RFC6690], in short:

The review policy is IETF Review for values starting with "core", and Specification Required for others.

The requirements to be enforced are:

- o The values MUST be related to the purpose described in Section 9.3.1.
- o The registered values MUST conform to the ABNF reg-rel-type definition of [RFC6690] and MUST NOT be a URI.
- o It is recommended to use the period "." character for segmentation.

The registry initially contains one value:

- o "core.rd-group": An application group as described in Appendix A.

9.5. Multicast Address Registration

IANA is asked to assign the following multicast addresses for use by CoAP nodes:

IPv4 - "all CoRE resource directories" address MCD2 (suggestion: 224.0.1.189), from the "IPv4 Multicast Address Space Registry". As the address is used for discovery that may span beyond a single network, it has come from the Internetwork Control Block (224.0.1.x, RFC 5771).

IPv6 - "all CoRE resource directories" address MCD1 (suggestions FF0X::FE), from the "IPv6 Multicast Address Space Registry", in the "Variable Scope Multicast Addresses" space (RFC 3307). Note that there is a distinct multicast address for each scope that interested CoAP nodes should listen to; CoAP needs the Link-Local and Site-Local scopes only.

[The RFC editor is asked to replace MCD1 and MCD2 with the assigned addresses throughout the document.]

10. Examples

Two examples are presented: a Lighting Installation example in Section 10.1 and a LWM2M example in Section 10.2.

10.1. Lighting Installation

This example shows a simplified lighting installation which makes use of the Resource Directory (RD) with a CoAP interface to facilitate the installation and start-up of the application code in the lights and sensors. In particular, the example leads to the definition of a group and the enabling of the corresponding multicast address as described in Appendix A. No conclusions must be drawn on the realization of actual installation or naming procedures, because the example only "emphasizes" some of the issues that may influence the use of the RD and does not pretend to be normative.

10.1.1. Installation Characteristics

The example assumes that the installation is managed. That means that a Commissioning Tool (CT) is used to authorize the addition of nodes, name them, and name their services. The CT can be connected to the installation in many ways: the CT can be part of the installation network, connected by WiFi to the installation network, or connected via GPRS link, or other method.

It is assumed that there are two naming authorities for the installation: (1) the network manager that is responsible for the correct operation of the network and the connected interfaces, and (2) the lighting manager that is responsible for the correct functioning of networked lights and sensors. The result is the existence of two naming schemes coming from the two managing entities.

The example installation consists of one presence sensor, and two luminaries, luminary1 and luminary2, each with their own wireless interface. Each luminary contains three lamps: left, right and middle. Each luminary is accessible through one endpoint. For each lamp a resource exists to modify the settings of a lamp in a luminary. The purpose of the installation is that the presence sensor notifies the presence of persons to a group of lamps. The group of lamps consists of: middle and left lamps of luminary1 and right lamp of luminary2.

Before commissioning by the lighting manager, the network is installed and access to the interfaces is proven to work by the network manager.

At the moment of installation, the network under installation is not necessarily connected to the DNS infra structure. Therefore, SLAAC IPv6 addresses are assigned to CT, RD, luminaries and sensor shown in Table 3 below:

| Name | IPv6 address |
|--------------------|----------------|
| luminary1 | 2001:db8:4::1 |
| luminary2 | 2001:db8:4::2 |
| Presence sensor | 2001:db8:4::3 |
| Resource directory | 2001:db8:4::ff |

Table 3: interface SLAAC addresses

In Section 10.1.2 the use of resource directory during installation is presented.

10.1.2. RD entries

It is assumed that access to the DNS infrastructure is not always possible during installation. Therefore, the SLAAC addresses are used in this section.

For discovery, the resource types (rt) of the devices are important. The lamps in the luminaries have rt: light, and the presence sensor has rt: p-sensor. The endpoints have names which are relevant to the light installation manager. In this case luminary1, luminary2, and the presence sensor are located in room 2-4-015, where luminary1 is located at the window and luminary2 and the presence sensor are located at the door. The endpoint names reflect this physical location. The middle, left and right lamps are accessed via path /light/middle, /light/left, and /light/right respectively. The identifiers relevant to the Resource Directory are shown in Table 4 below:

| Name | endpoint | resource path | resource type |
|-----------------|------------------|---------------|---------------|
| luminary1 | lm_R2-4-015_wndw | /light/left | light |
| luminary1 | lm_R2-4-015_wndw | /light/middle | light |
| luminary1 | lm_R2-4-015_wndw | /light/right | light |
| luminary2 | lm_R2-4-015_door | /light/left | light |
| luminary2 | lm_R2-4-015_door | /light/middle | light |
| luminary2 | lm_R2-4-015_door | /light/right | light |
| Presence sensor | ps_R2-4-015_door | /ps | p-sensor |

Table 4: Resource Directory identifiers

It is assumed that the CT knows the RD's address, and has performed URI discovery on it that returned a response like the one in the Section 4.3 example.

The CT inserts the endpoints of the luminaries and the sensor in the RD using the registration base URI parameter (base) to specify the interface address:

```
Req: POST coap://[2001:db8:4::ff]/rd
      ?ep=lm_R2-4-015_wndw&base=coap://[2001:db8:4::1]&d=R2-4-015
Payload:
</light/left>;rt="light",
</light/middle>;rt="light",
</light/right>;rt="light"
```

```
Res: 2.01 Created
Location-Path: /rd/4521
```

```
Req: POST coap://[2001:db8:4::ff]/rd
      ?ep=lm_R2-4-015_door&base=coap://[2001:db8:4::2]&d=R2-4-015
Payload:
</light/left>;rt="light",
</light/middle>;rt="light",
</light/right>;rt="light"
```

```
Res: 2.01 Created
Location-Path: /rd/4522
```

```
Req: POST coap://[2001:db8:4::ff]/rd
      ?ep=ps_R2-4-015_door&base=coap://[2001:db8:4::3]&d=R2-4-015
Payload:
</ps>;rt="p-sensor"
```

```
Res: 2.01 Created
Location-Path: /rd/4523
```

Figure 23: Example of registrations a CT enters into an RD

The sector name d=R2-4-015 has been added for an efficient lookup because filtering on "ep" name is more awkward. The same sector name is communicated to the two luminaries and the presence sensor by the CT.

The group is specified in the RD. The base parameter is set to the site-local multicast address allocated to the group. In the POST in the example below, the resources supported by all group members are published.

```
Req: POST coap://[2001:db8:4::ff]/rd
?ep=grp_R2-4-015&et=core.rd-group&base=coap://[ff05::1]
Payload:
</light/left>;rt="light",
</light/middle>;rt="light",
</light/right>;rt="light"

Res: 2.01 Created
Location-Path: /rd/501
```

Figure 24: Example of a multicast group a CT enters into an RD

After the filling of the RD by the CT, the application in the luminaries can learn to which groups they belong, and enable their interface for the multicast address.

The luminary, knowing its sector and being configured to join any group containing lights, searches for candidate groups and joins them:

```
Req: GET coap://[2001:db8:4::ff]/rd-lookup/ep
?d=R2-4-015&et=core.rd-group&rt=light

Res: 2.05 Content
</rd/501>;ep="grp_R2-4-015";et="core.rd-group";
base="coap://[ff05::1]";rt="core.rd-ep"
```

Figure 25: Example of a lookup exchange to find suitable multicast addresses

From the returned base parameter value, the luminary learns the multicast address of the multicast group.

Alternatively, the CT can communicate the multicast address directly to the luminaries by using the "coap-group" resource specified in [RFC7390].

```
Req: POST coap://[2001:db8:4::1]/coap-group
Content-Format: application/coap-group+json
Payload:
{ "a": "[ff05::1]", "n": "grp_R2-4-015" }

Res: 2.01 Created
Location-Path: /coap-group/1
```

Figure 26: Example use of direct multicast address configuration

Dependent on the situation, only the address, "a", or the name, "n", is specified in the coap-group resource.

The presence sensor can learn the presence of groups that support resources with rt=light in its own sector by sending the same request, as used by the luminary. The presence sensor learns the multicast address to use for sending messages to the luminaries.

10.2. OMA Lightweight M2M (LWM2M) Example

This example shows how the OMA LWM2M specification makes use of Resource Directory (RD).

OMA LWM2M is a profile for device services based on CoAP (OMA Name Authority). LWM2M defines a simple object model and a number of abstract interfaces and operations for device management and device service enablement.

An LWM2M server is an instance of an LWM2M middleware service layer, containing a Resource Directory along with other LWM2M interfaces defined by the LWM2M specification.

CoRE Resource Directory (RD) is used to provide the LWM2M Registration interface.

LWM2M does not provide for registration sectors and does not currently use the rd-lookup interface.

The LWM2M specification describes a set of interfaces and a resource model used between a LWM2M device and an LWM2M server. Other interfaces, proxies, and applications are currently out of scope for LWM2M.

The location of the LWM2M Server and RD URI path is provided by the LWM2M Bootstrap process, so no dynamic discovery of the RD is used. LWM2M Servers and endpoints are not required to implement the /.well-known/core resource.

10.2.1. The LWM2M Object Model

The OMA LWM2M object model is based on a simple 2 level class hierarchy consisting of Objects and Resources.

An LWM2M Resource is a REST endpoint, allowed to be a single value or an array of values of the same data type.

An LWM2M Object is a resource template and container type that encapsulates a set of related resources. An LWM2M Object represents

a specific type of information source; for example, there is a LWM2M Device Management object that represents a network connection, containing resources that represent individual properties like radio signal strength.

Since there may potentially be more than one of a given type object, for example more than one network connection, LWM2M defines instances of objects that contain the resources that represent a specific physical thing.

The URI template for LWM2M consists of a base URI followed by Object, Instance, and Resource IDs:

```
{/base-uri}/{/object-id}/{/object-instance}/{/resource-id}/{/resource-instance}
```

The five variables given here are strings. base-uri can also have the special value "undefined" (sometimes called "null" in RFC 6570). Each of the variables object-instance, resource-id, and resource-instance can be the special value "undefined" only if the values behind it in this sequence also are "undefined". As a special case, object-instance can be "empty" (which is different from "undefined") if resource-id is not "undefined".

base-uri := Base URI for LWM2M resources or "undefined" for default (empty) base URI

object-id := OMNA (OMA Name Authority) registered object ID (0-65535)

object-instance := Object instance identifier (0-65535) or "undefined"/"empty" (see above) to refer to all instances of an object ID

resource-id := OMNA (OMA Name Authority) registered resource ID (0-65535) or "undefined" to refer to all resources within an instance

resource-instance := Resource instance identifier or "undefined" to refer to single instance of a resource

LWM2M IDs are 16 bit unsigned integers represented in decimal (no leading zeroes except for the value 0) by URI format strings. For example, a LWM2M URI might be:

```
/1/0/1
```

The base uri is empty, the Object ID is 1, the instance ID is 0, the resource ID is 1, and the resource instance is "undefined". This example URI points to internal resource 1, which represents the

registration lifetime configured, in instance 0 of a type 1 object (LWM2M Server Object).

10.2.2. LWM2M Register Endpoint

LWM2M defines a registration interface based on the REST API, described in Section 5. The RD registration URI path of the LWM2M Resource Directory is specified to be "/rd".

LWM2M endpoints register object IDs, for example </1>, to indicate that a particular object type is supported, and register object instances, for example </1/0>, to indicate that a particular instance of that object type exists.

Resources within the LWM2M object instance are not registered with the RD, but may be discovered by reading the resource links from the object instance using GET with a CoAP Content-Format of application/link-format. Resources may also be read as a structured object by performing a GET to the object instance with a Content-Format of senml+json.

When an LWM2M object or instance is registered, this indicates to the LWM2M server that the object and its resources are available for management and service enablement (REST API) operations.

LWM2M endpoints may use the following RD registration parameters as defined in Table 2 :

ep - Endpoint Name
lt - registration lifetime

Endpoint Name, Lifetime, and LWM2M Version are mandatory parameters for the register operation, all other registration parameters are optional.

Additional optional LWM2M registration parameters are defined:

| Name | Query | Validity | Description |
|---------------|-------|-------------------------------------|---------------------|
| Binding Mode | b | {"U", "UQ", "S", "SQ", "US", "UQS"} | Available Protocols |
| LWM2M Version | ver | 1.0 | Spec Version |
| SMS Number | sms | | MSISDN |

Table 5: LWM2M Additional Registration Parameters

The following RD registration parameters are not currently specified for use in LWM2M:

et - Endpoint Type
base - Registration Base URI

The endpoint registration must include a payload containing links to all supported objects and existing object instances, optionally including the appropriate link-format relations.

Here is an example LWM2M registration payload:

```
</1>,</1/0>,</3/0>,</5>
```

This link format payload indicates that object ID 1 (LWM2M Server Object) is supported, with a single instance 0 existing, object ID 3 (LWM2M Device object) is supported, with a single instance 0 existing, and object 5 (LWM2M Firmware Object) is supported, with no existing instances.

10.2.3. LWM2M Update Endpoint Registration

The Lwm2M update is really very similar to the registration update as described in Section 5.3.1, with the only difference that there are more parameters defined and available. All the parameters listed in that section are also available with the initial registration but are all optional:

lt - Registration Lifetime
b - Protocol Binding
sms - MSISDN
link payload - new or modified links

A Registration update is also specified to be used to update the LWM2M server whenever the endpoint's UDP port or IP address are changed.

10.2.4. LWM2M De-Register Endpoint

LWM2M allows for de-registration using the delete method on the returned location from the initial registration operation. LWM2M de-registration proceeds as described in Section 5.3.2.

11. Acknowledgments

Oscar Novo, Srdjan Krco, Szymon Sasin, Kerry Lynn, Esko Dijk, Anders Brandt, Matthieu Vial, Jim Schaad, Mohit Sethi, Hauke Petersen, Hannes Tschofenig, Sampo Ukkola, Linyi Tian, Jan Newmarch, Matthias Kovatsch, Jaime Jimenez and Ted Lemon have provided helpful comments, discussions and ideas to improve and shape this document. Zach would also like to thank his colleagues from the EU FP7 SENSEI project, where many of the resource directory concepts were originally developed.

12. Changelog

changes from -22 to -23

- o Explain that updates can not remove attributes
- o Typo fixes

changes from -21 to -22

- o Request a dedicated IPv4 address from IANA (rather than sharing with All CoAP nodes)
- o Fix erroneous examples
- o Editorial changes
 - * Add figure numbers to examples
 - * Update RD parameters table to reflect changes of earlier versions in the text
 - * Typos and minor wording

changes from -20 to -21

(Processing comments during WGLC)

- o Defer outdated description of using DNS-SD to find an RD to the defining document
- o Describe operational conditions in automation example
- o Recommend particular discovery mechanisms for some managed network scenarios

changes from -19 to -20

(Processing comments from the WG chair review)

- o Define the permissible characters in endpoint and sector names
- o Express requirements on NAT situations in more abstract terms
- o Shifted heading levels to have the interfaces on the same level
- o Group instructions for error handling into general section
- o Simple Registration: process reflowed into items list
- o Updated introduction to reflect state of CoRE in general, reference RFC7228 (defining "constrained") and use "IoT" term in addition to "M2M"
- o Update acknowledgements
- o Assorted editorial changes
 - * Unify examples style
 - * Terminology: RDAO defined and not only expanded
 - * Add CT to Figure 1
 - * Consistency in the use of the term "Content Format"

changes from -18 to -19

- o link-local addresses: allow but prescribe split-horizon fashion when used, disallow zone identifiers
- o Remove informative references to documents not mentioned any more

changes from -17 to -18

- o Rather than re-specifying link format (Modernized Link Format), describe a Limited Link Format that's the uncontested subset of Link Format
- o Acknowledging the -17 version as part of the draft
- o Move "Read endpoint links" operation to future specification like PATCH
- o Demote links-json to an informative reference, and removed them from exchange examples
- o Add note on unusability of link-local IP addresses, and describe mitigation.
- o Reshuffling of sections: Move additional operations and endpoint lookup back from appendix, and groups into one
- o Lookup interface tightened to not imply applicability for non link-format lookups (as those can have vastly different views on link cardinality)
- o Simple registration: Change sequence of GET and POST-response, ensuring unsuccessful registrations are reported as such, and suggest how devices that would have required the inverse behavior can still cope with it.
- o Abstract and introduction reworded to avoid the impression that resources are stored in full in the RD
- o Simplify the rules governing when a registration resource can or must be changed.
- o Drop a figure that has become useless due to the changes of and -13 and -17
- o Wording consistency fixes: Use "Registrations" and "target attributes"
- o Fix incorrect use of content negotiation in discovery interface description (Content-Format -> Accept)
- o State that the base attribute value is part of endpoint lookup even when implicit in the registration
- o Update references from RFC5988 to its update RFC8288

- o Remove appendix on protocol-negotiation (which had a note to be removed before publication)

changes from -16 to -17

(Note that -17 is published as a direct follow-up to -16, containing a single change to be discussed at IETF103)

- o Removed groups that are enumerations of registrations and have dedicated mechanism
- o Add groups that are enumerations of shared resources and are a special case of endpoint registrations

changes from -15 to -16

- o Recommend a common set of resources for members of a group
- o Clarified use of multicast group in lighting example
- o Add note on concurrent registrations from one EP being possible but not expected
- o Refresh web examples appendix to reflect current use of Modernized Link Format
- o Add examples of URIs where Modernized Link Format matters
- o Editorial changes

changes from -14 to -15

- o Rewrite of section "Security policies"
- o Clarify that the "base" parameter text applies both to relative references both in anchor and href
- o Renamed "Registree-EP" to Registrant-EP"
- o Talk of "relative references" and "URIs" rather than "relative" and "absolute" URIs. (The concept of "absolute URIs" of [RFC3986] is not needed in RD).
- o Fixed examples
- o Editorial changes

changes from -13 to -14

- o Rename "registration context" to "registration base URI" (and "con" to "base") and "domain" to "sector" (where the abbreviation "d" stays for compatibility reasons)
 - o Introduced resource types core.rd-ep and core.rd-gp
 - o Registration management moved to appendix A, including endpoint and group lookup
 - o Minor editorial changes
 - * PATCH/iPATCH is clearly deferred to another document
 - * Recommend against query / fragment identifier in con=
 - * Interface description lists are described as illustrative
 - * Rewording of Simple Registration
 - o Simple registration carries no error information and succeeds immediately (previously, sequence was unspecified)
 - o Lookup: href are matched against resolved values (previously, this was unspecified)
 - o Lookup: lt are not exposed any more
 - o con/base: Paths are allowed
 - o Registration resource locations can not have query or fragment parts
 - o Default life time extended to 25 hours
 - o clarified registration update rules
 - o lt-value semantics for lookup clarified.
 - o added template for simple registration
- changes from -12 to -13
- o Added "all resource directory" nodes MC address
 - o Clarified observation behavior
 - o version identification

- o example rt= and et= values
- o domain from figure 2
- o more explanatory text
- o endpoints of a groups hosted by different RD
- o resolve RFC6690-vs-8288 resolution ambiguities:
 - * require registered links not to be relative when using anchor
 - * return absolute URIs in resource lookup

changes from -11 to -12

- o added Content Model section, including ER diagram
- o removed domain lookup interface; domains are now plain attributes of groups and endpoints
- o updated chapter "Finding a Resource Directory"; now distinguishes configuration-provided, network-provided and heuristic sources
- o improved text on: atomicity, idempotency, lookup with multiple parameters, endpoint removal, simple registration
- o updated LWM2M description
- o clarified where relative references are resolved, and how context and anchor interact
- o new appendix on the interaction with RFCs 6690, 5988 and 3986
- o lookup interface: group and endpoint lookup return group and registration resources as link targets
- o lookup interface: search parameters work the same across all entities
- o removed all methods that modify links in an existing registration (POST with payload, PATCH and iPATCH)
- o removed plurality definition (was only needed for link modification)
- o enhanced IANA registry text

- o state that lookup resources can be observable
 - o More examples and improved text
- changes from -09 to -10
- o removed "ins" and "exp" link-format extensions.
 - o removed all text concerning DNS-SD.
 - o removed inconsistency in RDAO text.
 - o suggestions taken over from various sources
 - o replaced "Function Set" with "REST API", "base URI", "base path"
 - o moved simple registration to registration section
- changes from -08 to -09
- o clarified the "example use" of the base RD resource values /rd, /rd-lookup, and /rd-group.
 - o changed "ins" ABNF notation.
 - o various editorial improvements, including in examples
 - o clarifications for RDAO
- changes from -07 to -08
- o removed link target value returned from domain and group lookup types
 - o Maximum length of domain parameter 63 bytes for consistency with group
 - o removed option for simple POST of link data, don't require a .well-known/core resource to accept POST data and handle it in a special way; we already have /rd for that
 - o add IPv6 ND Option for discovery of an RD
 - o clarify group configuration section 6.1 that endpoints must be registered before including them in a group
 - o removed all superfluous client-server diagrams

- o simplified lighting example
- o introduced Commissioning Tool
- o RD-Look-up text is extended.

changes from -06 to -07

- o added text in the discovery section to allow content format hints to be exposed in the discovery link attributes
- o editorial updates to section 9
- o update author information
- o minor text corrections

Changes from -05 to -06

- o added note that the PATCH section is contingent on the progress of the PATCH method

changes from -04 to -05

- o added Update Endpoint Links using PATCH
- o http access made explicit in interface specification
- o Added http examples

Changes from -03 to -04:

- o Added http response codes
- o Clarified endpoint name usage
- o Add application/link-format+cbor content-format

Changes from -02 to -03:

- o Added an example for lighting and DNS integration
- o Added an example for RD use in OMA LWM2M
- o Added Read Links operation for link inspection by endpoints
- o Expanded DNS-SD section

- o Added draft authors Peter van der Stok and Michael Koster

Changes from -01 to -02:

- o Added a catalogue use case.
- o Changed the registration update to a POST with optional link format payload. Removed the endpoint type update from the update.
- o Additional examples section added for more complex use cases.
- o New DNS-SD mapping section.
- o Added text on endpoint identification and authentication.
- o Error code 4.04 added to Registration Update and Delete requests.
- o Made 63 bytes a SHOULD rather than a MUST for endpoint name and resource type parameters.

Changes from -00 to -01:

- o Removed the ETag validation feature.
- o Place holder for the DNS-SD mapping section.
- o Explicitly disabled GET or POST on returned Location.
- o New registry for RD parameters.
- o Added support for the JSON Link Format.
- o Added reference to the Groupcomm WG draft.

Changes from -05 to WG Document -00:

- o Updated the version and date.

Changes from -04 to -05:

- o Restricted Update to parameter updates.
- o Added pagination support for the Lookup interface.
- o Minor editing, bug fixes and reference updates.
- o Added group support.

- o Changed rt to et for the registration and update interface.

Changes from -03 to -04:

- o Added the ins= parameter back for the DNS-SD mapping.
- o Integrated the Simple Directory Discovery from Carsten.
- o Editorial improvements.
- o Fixed the use of ETags.
- o Fixed tickets 383 and 372

Changes from -02 to -03:

- o Changed the endpoint name back to a single registration parameter ep= and removed the h= and ins= parameters.
- o Updated REST interface descriptions to use RFC6570 URI Template format.
- o Introduced an improved RD Lookup design as its own function set.
- o Improved the security considerations section.
- o Made the POST registration interface idempotent by requiring the ep= parameter to be present.

Changes from -01 to -02:

- o Added a terminology section.
- o Changed the inclusion of an ETag in registration or update to a MAY.
- o Added the concept of an RD Domain and a registration parameter for it.
- o Recommended the Location returned from a registration to be stable, allowing for endpoint and Domain information to be changed during updates.
- o Changed the lookup interface to accept endpoint and Domain as query string parameters to control the scope of a lookup.

13. References

13.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <<https://www.rfc-editor.org/info/rfc3986>>.
- [RFC6570] Gregorio, J., Fielding, R., Hadley, M., Nottingham, M., and D. Orchard, "URI Template", RFC 6570, DOI 10.17487/RFC6570, March 2012, <<https://www.rfc-editor.org/info/rfc6570>>.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <<https://www.rfc-editor.org/info/rfc6690>>.
- [RFC6763] Cheshire, S. and M. Krochmal, "DNS-Based Service Discovery", RFC 6763, DOI 10.17487/RFC6763, February 2013, <<https://www.rfc-editor.org/info/rfc6763>>.
- [RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <<https://www.rfc-editor.org/info/rfc8126>>.

13.2. Informative References

- [ER] Chen, P., "The entity-relationship model---toward a unified view of data", ACM Transactions on Database Systems Vol. 1, pp. 9-36, DOI 10.1145/320434.320440, March 1976.
- [I-D.bormann-t2trg-rel-impl] Bormann, C., "impl-info: A link relation type for disclosing implementation information", draft-bormann-t2trg-rel-impl-00 (work in progress), January 2018.
- [I-D.hartke-t2trg-coral] Hartke, K., "The Constrained RESTful Application Language (CoRAL)", draft-hartke-t2trg-coral-08 (work in progress), March 2019.

- [I-D.ietf-ace-oauth-authz]
Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "Authentication and Authorization for Constrained Environments (ACE) using the OAuth 2.0 Framework (ACE-OAuth)", draft-ietf-ace-oauth-authz-24 (work in progress), March 2019.
- [I-D.ietf-core-links-json]
Li, K., Rahman, A., and C. Bormann, "Representing Constrained RESTful Environments (CoRE) Link Format in JSON and CBOR", draft-ietf-core-links-json-10 (work in progress), February 2018.
- [I-D.ietf-core-rd-dns-sd]
Stok, P., Koster, M., and C. Amsuess, "CoRE Resource Directory: DNS-SD mapping", draft-ietf-core-rd-dns-sd-05 (work in progress), July 2019.
- [I-D.silverajan-core-coap-protocol-negotiation]
Silverajan, B. and M. Ocak, "CoAP Protocol Negotiation", draft-silverajan-core-coap-protocol-negotiation-09 (work in progress), July 2018.
- [RFC4944] Montenegro, G., Kushalnagar, N., Hui, J., and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks", RFC 4944, DOI 10.17487/RFC4944, September 2007, <<https://www.rfc-editor.org/info/rfc4944>>.
- [RFC6775] Shelby, Z., Ed., Chakrabarti, S., Nordmark, E., and C. Bormann, "Neighbor Discovery Optimization for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs)", RFC 6775, DOI 10.17487/RFC6775, November 2012, <<https://www.rfc-editor.org/info/rfc6775>>.
- [RFC6874] Carpenter, B., Cheshire, S., and R. Hinden, "Representing IPv6 Zone Identifiers in Address Literals and Uniform Resource Identifiers", RFC 6874, DOI 10.17487/RFC6874, February 2013, <<https://www.rfc-editor.org/info/rfc6874>>.
- [RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, DOI 10.17487/RFC7228, May 2014, <<https://www.rfc-editor.org/info/rfc7228>>.
- [RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <<https://www.rfc-editor.org/info/rfc7230>>.

- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<https://www.rfc-editor.org/info/rfc7252>>.
- [RFC7390] Rahman, A., Ed. and E. Dijk, Ed., "Group Communication for the Constrained Application Protocol (CoAP)", RFC 7390, DOI 10.17487/RFC7390, October 2014, <<https://www.rfc-editor.org/info/rfc7390>>.
- [RFC7641] Hartke, K., "Observing Resources in the Constrained Application Protocol (CoAP)", RFC 7641, DOI 10.17487/RFC7641, September 2015, <<https://www.rfc-editor.org/info/rfc7641>>.
- [RFC8132] van der Stok, P., Bormann, C., and A. Sehgal, "PATCH and FETCH Methods for the Constrained Application Protocol (CoAP)", RFC 8132, DOI 10.17487/RFC8132, April 2017, <<https://www.rfc-editor.org/info/rfc8132>>.
- [RFC8288] Nottingham, M., "Web Linking", RFC 8288, DOI 10.17487/RFC8288, October 2017, <<https://www.rfc-editor.org/info/rfc8288>>.

Appendix A. Groups Registration and Lookup

The RD-Groups usage pattern allows announcing application groups inside a Resource Directory.

Groups are represented by endpoint registrations. Their base address is a multicast address, and they SHOULD be entered with the endpoint type "core.rd-group". The endpoint name can also be referred to as a group name in this context.

The registration is inserted into the RD by a Commissioning Tool, which might also be known as a group manager here. It performs third party registration and registration updates.

The links it registers SHOULD be available on all members that join the group. Depending on the application, members that lack some resource MAY be permissible if requests to them fail gracefully.

The following example shows a CT registering a group with the name "lights" which provides two resources. The directory resource path /rd is an example RD location discovered in a request similar to Figure 5.

```
Req: POST coap://rd.example.com/rd?ep=lights&et=core.rd-group
      &base=coap://[ff35:30:2001:db8::1]
Content-Format: 40
Payload:
</light>;rt="light";if="core.a",
</color-temperature>;if="core.p";u="K"

Res: 2.01 Created
Location-Path: /rd/12
```

Figure 27: Example registration of a group

In this example, the group manager can easily permit devices that have no writable color-temperature to join, as they would still respond to brightness changing commands. Had the group instead contained a single resource that sets brightness and color temperature atomically, endpoints would need to support both properties.

The resources of a group can be looked up like any other resource, and the group registrations (along with any additional registration parameters) can be looked up using the endpoint lookup interface.

The following example shows a client performing an endpoint lookup for all groups.

```
Req: GET /rd-lookup/ep?et=core.rd-group

Res: 2.01 Content
Payload:
</rd/501>;ep="GRP_R2-4-015";et="core.rd-group";
      base="coap://[ff05::1]",
</rd/12>;ep=lights&et=core.rd-group;
      base="coap://[ff35:30:2001:db8::1]";rt="core.rd-ep"
```

Figure 28: Example lookup of groups

The following example shows a client performing a lookup of all resources of all endpoints (groups) with et=core.rd-group.

```

Req: GET /rd-lookup/res?et=core.rd-group

<coap://[ff35:30:2001:db8::1]/light>;rt="light";if="core.a";
  et="core.rd-group";anchor="coap://[ff35:30:2001:db8::1]",
<coap://[ff35:30:2001:db8::1]/color-temperature>;if="core.p";u="K";
  et="core.rd-group";
  anchor="coap://[ff35:30:2001:db8::1]"

```

Figure 29: Example lookup of resources inside groups

Appendix B. Web links and the Resource Directory

Understanding the semantics of a link-format document and its URI references is a journey through different documents ([RFC3986] defining URIs, [RFC6690] defining link-format documents based on [RFC8288] which defines link headers, and [RFC7252] providing the transport). This appendix summarizes the mechanisms and semantics at play from an entry in ".well-known/core" to a resource lookup.

This text is primarily aimed at people entering the field of Constrained Restful Environments from applications that previously did not use web mechanisms.

The explanation of the steps makes some shortcuts in the more confusing details of [RFC6690], which are justified as all examples being in Limited Link Format.

B.1. A simple example

Let's start this example with a very simple host, "2001:db8:f0::1". A client that follows classical CoAP Discovery ([RFC7252] Section 7), sends the following multicast request to learn about neighbours supporting resources with resource-type "temperature".

The client sends a link-local multicast:

```

GET coap://[ff02::fd]:5683/.well-known/core?rt=temperature

RES 2.05 Content
</temp>;rt=temperature;ct=0

```

Figure 30: Example of direct resource discovery

where the response is sent by the server, "[2001:db8:f0::1]:5683".

While the client - on the practical or implementation side - can just go ahead and create a new request to "[2001:db8:f0::1]:5683" with

Uri-Path: "temp", the full resolution steps for insertion into and retrieval from the RD without any shortcuts are:

B.1.1. Resolving the URIs

The client parses the single returned record. The link's target (sometimes called "href") is `"/temp"`, which is a relative URI that needs resolving. The base URI `<coap://[ff02::fd]:5683/.well-known/core>` is used to resolve the reference `/temp` against.

The Base URI of the requested resource can be composed from the header options of the CoAP GET request by following the steps of [RFC7252] section 6.5 (with an addition at the end of 8.2) into `"coap://[2001:db8:f0::1]/.well-known/core"`.

Because `"/temp"` starts with a single slash, the record's target is resolved by replacing the path `"/.well-known/core"` from the Base URI (section 5.2 [RFC3986]) with the relative target URI `"/temp"` into `"coap://[2001:db8:f0::1]/temp"`.

B.1.2. Interpreting attributes and relations

Some more information but the record's target can be obtained from the payload: the resource type of the target is "temperature", and its content format is text/plain (ct=0).

A relation in a web link is a three-part statement that specifies a named relation between the so-called "context resource" and the target resource, like `"_This page_ has _its table of contents_ at _/toc.html_"`. In link format documents, there is an implicit "host relation" specified with default parameter: `rel="hosts"`.

In our example, the context resource of the link is the URI specified in the GET request `"coap://[2001:db8:f0::1]/.well-known/core"`. A full English expression of the "host relation" is:

`'"coap://[2001:db8:f0::1]/.well-known/core" is hosting the resource "coap://[2001:db8:f0::1]/temp", which is of the resource type "temperature" and can be accessed using the text/plain content format.'`

B.2. A slightly more complex example

Omitting the `"rt=temperature"` filter, the discovery query would have given some more records in the payload:

```

GET coap://[ff02::fd]:5683/.well-known/core

RES 2.05 Content
</temp>;rt=temperature;ct=0,
</light>;rt=light-lux;ct=0,
</t>;anchor="/sensors/temp";rel=alternate,
<http://www.example.com/sensors/t123>;anchor="/temp";
  rel="describedby"

```

Figure 31: Extended example of direct resource discovery

Parsing the third record, the client encounters the "anchor" parameter. It is a URI relative to the Base URI of the request and is thus resolved to "coap://[2001:db8:f0::1]/sensors/temp". That is the context resource of the link, so the "rel" statement is not about the target and the Base URI any more, but about the target and the resolved URI. Thus, the third record could be read as "coap://[2001:db8:f0::1]/sensors/temp" has an alternate representation at "coap://[2001:db8:f0::1]/t".

Following the same resolution steps, the fourth record can be read as "coap://[2001:db8:f0::1]/sensors/temp" is described by "http://www.example.com/sensors/t123".

B.3. Enter the Resource Directory

The resource directory tries to carry the semantics obtainable by classical CoAP discovery over to the resource lookup interface as faithfully as possible.

For the following queries, we will assume that the simple host has used Simple Registration to register at the resource directory that was announced to it, sending this request from its UDP port "[2001:db8:f0::1]:6553":

```
POST coap://[2001:db8:f01::ff]/.well-known/core?ep=simple-host1
```

Figure 32: Example request starting a simple registration

The resource directory would have accepted the registration, and queried the simple host's ".well-known/core" by itself. As a result, the host is registered as an endpoint in the RD with the name "simple-host1". The registration is active for 90000 seconds, and the endpoint registration Base URI is "coap://[2001:db8:f0::1]" following the resolution steps described in Appendix B.1.1. It should be remarked that the Base URI constructed that way always yields a URI of the form: scheme://authority without path suffix.

If the client now queries the RD as it would previously have issued a multicast request, it would go through the RD discovery steps by fetching "coap://[2001:db8:f0::ff]/.well-known/core?rt=core.rd-lookup-res", obtain "coap://[2001:db8:f0::ff]/rd-lookup/res" as the resource lookup endpoint, and issue a request to "coap://[2001:db8:f0::ff]/rd-lookup/res?rt=temperature" to receive the following data:

```
<coap://[2001:db8:f0::1]/temp>;rt=temperature;ct=0;
  anchor="coap://[2001:db8:f0::1]"
```

Figure 33: Example payload of a response to a resource lookup

This is not literally the same response that it would have received from a multicast request, but it contains the equivalent statement:

```
'"coap://[2001:db8:f0::1]" is hosting the resource
"coap://[2001:db8:f0::1]/temp", which is of the resource type
"temperature" and can be accessed using the text/plain content
format.'
```

(The difference is whether "/" or "/.well-known/core" hosts the resources, which does not matter in this application; if it did, the endpoint would have been more explicit. Actually, /.well-known/core does NOT host the resource but stores a URI reference to the resource.)

To complete the examples, the client could also query all resources hosted at the endpoint with the known endpoint name "simple-host1". A request to "coap://[2001:db8:f0::ff]/rd-lookup/res?ep=simple-host1" would return

```
<coap://[2001:db8:f0::1]/temp>;rt=temperature;ct=0;
  anchor="coap://[2001:db8:f0::1]",
<coap://[2001:db8:f0::1]/light>;rt=light-lux;ct=0;
  anchor="coap://[2001:db8:f0::1]",
<coap://[2001:db8:f0::1]/t>;
  anchor="coap://[2001:db8:f0::1]/sensors/temp";rel=alternate,
<http://www.example.com/sensors/t123>;
  anchor="coap://[2001:db8:f0::1]/sensors/temp";rel="describedby"
```

Figure 34: Extended example payload of a response to a resource lookup

All the target and anchor references are already in absolute form there, which don't need to be resolved any further.

Had the simple host done an equivalent full registration with a base= parameter (e.g. "?ep=simple-host1&base=coap+tcp://simple-host1.example.com"), that context would have been used to resolve the relative anchor values instead, giving

```
<coap+tcp://simple-host1.example.com/temp>;rt=temperature;ct=0;
  anchor="coap+tcp://simple-host1.example.com"
```

Figure 35: Example payload of a response to a resource lookup with a dedicated base URI

and analogous records.

B.4. A note on differences between link-format and Link headers

While link-format and Link headers look very similar and are based on the same model of typed links, there are some differences between [RFC6690] and [RFC8288], which are dealt with differently:

- o "Resolving the target against the anchor": [RFC6690] Section 2.1 states that the anchor of a link is used as the Base URI against which the term inside the angle brackets (the target) is resolved, falling back to the resource's URI with paths stripped off (its "Origin"). In contrast to that, [RFC8288] Section B.2 describes that the anchor is immaterial to the resolution of the target reference.

RFC6690, in the same section, also states that absent anchors set the context of the link to the target's URI with its path stripped off, while according to [RFC8288] Section 3.2, the context is the resource's base URI.

The rules introduced in Appendix C ensure that an RD does not need to deal with those differences when processing input data. Lookup results are required to be absolute references for the same reason.

- o There is no percent encoding in link-format documents.

A link-format document is a UTF-8 encoded string of Unicode characters and does not have percent encoding, while Link headers are practically ASCII strings that use percent encoding for non-ASCII characters, stating the encoding explicitly when required.

For example, while a Link header in a page about a Swedish city might read

```
"Link: </temperature/Malm%C3%B6>;rel="live-environment-data"
```

a link-format document from the same source might describe the link as

```
"</temperature/Malmoe>;rel="live-environment-data"
```

Parsers and producers of link-format and header data need to be aware of this difference.

Appendix C. Limited Link Format

The CoRE Link Format as described in [RFC6690] has been interpreted differently by implementers, and a strict implementation rules out some use cases of a Resource Directory (e.g. base values with path components).

This appendix describes a subset of link format documents called Limited Link Format. The rules herein are not very limiting in practice – all examples in RFC6690, and all deployments the authors are aware of already stick to them – but ease the implementation of resource directory servers.

It is applicable to representations in the application/link-format media type, and any other media types that inherit [RFC6690] Section 2.1.

A link format representation is in Limited Link format if, for each link in it, the following applies:

- o All URI references either follow the URI or the path-absolute ABNF rule of RFC3986 (i.e. target and anchor each either start with a scheme or with a single slash),
- o if the anchor reference starts with a scheme, the target reference starts with a scheme as well (i.e. relative references in target cannot be used when the anchor is a full URI), and
- o the application does not care whether links without an explicitly given anchor have the origin's "/" or "/.well-known/core" resource as their link context.

Authors' Addresses

Zach Shelby
ARM
150 Rose Orchard
San Jose 95134
USA

Phone: +1-408-203-9434
Email: zach.shelby@arm.com

Michael Koster
SmartThings
665 Clyde Avenue
Mountain View 94043
USA

Phone: +1-707-502-5136
Email: Michael.Koster@smarththings.com

Carsten Bormann
Universitaet Bremen TZI
Postfach 330440
Bremen D-28359
Germany

Phone: +49-421-218-63921
Email: cabo@tzi.org

Peter van der Stok
consultant

Phone: +31-492474673 (Netherlands), +33-966015248 (France)
Email: consultancy@vanderstok.org
URI: www.vanderstok.org

Christian Amsuess (editor)
Hollandstr. 12/4
1020
Austria

Phone: +43-664-9790639
Email: christian@amsuess.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 9, 2020

A. Keranen
Ericsson
M. Mohajer
u-blox UK
July 8, 2019

FETCH & PATCH with Sensor Measurement Lists (SenML)
draft-ietf-core-senml-etch-04

Abstract

The Sensor Measurement Lists (SenML) media type and data model can be used to send collections of resources, such as batches of sensor data or configuration parameters. The CoAP iPATCH, PATCH, and FETCH methods enable accessing and updating parts of a resource or multiple resources with one request. This document defines new media types for the CoAP iPATCH, PATCH, and FETCH methods for resources represented with the SenML data model.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on January 9, 2020.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must

include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

| | |
|--|---|
| 1. Introduction | 2 |
| 2. Terminology | 3 |
| 3. Using FETCH and (i)PATCH with SenML | 3 |
| 3.1. SenML FETCH | 4 |
| 3.2. SenML (i)PATCH | 4 |
| 4. Security Considerations | 5 |
| 5. IANA Considerations | 6 |
| 5.1. CoAP Content-Format Registration | 6 |
| 5.2. senml-etch+json Media Type | 6 |
| 5.3. senml-etch+cbor Media Type | 7 |
| 6. Acknowledgements | 8 |
| 7. References | 9 |
| 7.1. Normative References | 9 |
| 7.2. Informative References | 9 |
| Authors' Addresses | 9 |

1. Introduction

The Sensor Measurement Lists (SenML) media type [RFC8428] and data model can be used to transmit collections of resources, such as batches of sensor data or configuration parameters.

An example of a SenML collection is shown below:

```
[
  {"bn":"2001:db8::2/3311/0/", "n":"5850", "vb":true},
  {"n":"5851", "v":42},
  {"n":"5750", "vs":"Ceiling light"}
]
```

Here three resources "3311/0/5850", "3311/0/5851", and "3311/0/5750", of an IPSO dimmable light smart object [IPSO] are represented using a single SenML Pack with three SenML Records. All resources share the same base name "2001:db8::2/3311/0/", hence full names for resources are "2001:db8::2/3311/0/5850", etc.

The CoAP [RFC7252] iPATCH, PATCH, and FETCH methods [RFC8132] enable accessing and updating parts of a resource or multiple resources with one request.

This document defines two new media types, one using the JavaScript Object Notation (JSON) [RFC8259] and one using the Concise Binary

Object Representation (CBOR) [RFC7049], that can be used with the CoAP iPATCH, PATCH, and FETCH methods for resources represented with the SenML data model. The semantics of the new media types are the same for the CoAP PATCH and iPATCH methods. The rest of the document uses term "(i)PATCH" when referring to both methods.

2. 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 BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

Readers should also be familiar with the terms and concepts discussed in [RFC8132] and [RFC8428]. Also the following terms are used in this document:

Fetch Record: One set of parameters that is used to match SenML Record(s).

Fetch Pack: One or more Fetch Records in an array structure.

Patch Record: One set of parameters similar to Fetch Record but also containing instructions on how to change existing SenML Pack(s).

Patch Pack: One or more Patch Records in an array structure.

Target Record: A Record in a SenML Pack that is matching the selection criteria of a Fetch or Patch Record and hence is a target for a Fetch or Patch operation.

(i)PATCH: A term that refers to both CoAP "PATCH" and "iPATCH" methods when there is no difference in this specification in which one is used.

3. Using FETCH and (i)PATCH with SenML

The FETCH/(i)PATCH media types for SenML are modeled as extensions to the SenML media type to enable re-use of existing SenML parsers and generators, in particular on constrained devices. Unless mentioned otherwise, FETCH and PATCH Packs are constructed with the same rules and constraints as SenML Packs.

The key difference to the SenML media type is allowing the use of a "null" value for removing records with the (i)PATCH method. Also the Fetch and Patch Records do not have default time or base version when the fields are omitted.

3.1. SenML FETCH

The FETCH method can be used to select and return a subset of records, in sequence, of one or more SenML Packs. The SenML Records are selected by giving a set of names that, when resolved, match resolved names in a SenML Pack. The names for a Fetch Pack are given using the SenML "name" and/or "base name" Fields. The names are resolved by concatenating the base name with the name field as defined in [RFC8428].

For example, to select the IPSO resources "5850" and "5851" from the example in Section 1, the following Fetch Pack can be used:

```
[
  {"bn":"2001:db8::2/3311/0/", "n":"5850"},
  {"n":"5851"}
]
```

The result to a FETCH request with the example above would be:

```
[
  {"bn":"2001:db8::2/3311/0/", "n":"5850", "vb":true},
  {"n":"5851", "v":42},
]
```

When SenML Records contain also time values, a name may no longer uniquely identify a single Record. When no time is given in a Fetch Record, all SenML Records with the given name are matched (i.e., unlike with SenML Records, lack of time field in a Fetch Record does not imply time value zero). When time is given in the Fetch Record, only a SenML Record (if any) with equal resolved time value and name is matched.

The resolved form of records (Section 4.6 of [RFC8428]) is used when comparing the names and times of the Target and Fetch Records to accommodate for differences in use of the base values.

3.2. SenML (i)PATCH

The (i)PATCH method can be used to change the values of SenML Records, to add new Records, and to remove existing Records. The names and times of the Patch Records are given and matched in same way as for the Fetch Records, except each Patch Record can match at most one Target Record. Patch Packs can also include new values and other SenML Fields for the Records. Application of Patch Packs is idempotent.

When the name in a Patch Record matches with the name in an existing Record, the resolved time values are compared. If the time values either do not exist in both Records or are equal, the Target Record is replaced with the contents of the Patch Record.

If a Patch Record contains a name, or combination of a time value and a name, that do not exist in any existing Record in the Pack, the given Record, with all the fields it contains, is added to the Pack.

If a Patch Record has a value ("v") field with value null, the matched Record (if any) is removed from the Pack.

For example, the following document could be given as an (i)PATCH payload to change/set values of two SenML Records for the example in Section 1:

```
[
  {"bn":"2001:db8::2/3311/0/", "n":"5850", "vb":false},
  {"n":"5851", "v":10}
]
```

If the request is successful, the resulting representation of the example SenML Pack would be as follows:

```
[
  {"bn":"2001:db8::2/3311/0/", "n":"5850", "vb":false},
  {"n":"5851", "v":10},
  {"n":"5750", "vs":"Ceiling light"}
]
```

As another example, the following document could be given as an (i)PATCH payload to remove the two SenML Records:

```
[
  {"bn":"2001:db8::2/3311/0/", "n":"5850", "v":null},
  {"n":"5851", "v":null}
]
```

4. Security Considerations

The security and privacy considerations of SenML apply also with the FETCH and (i)PATCH methods.

In FETCH and (i)PATCH requests, the client can pass arbitrary names to the target resource for manipulation. The resource implementer must take care to only allow access to names that are actually part of (or accessible through) the target resource.

If the client is not allowed to do a GET or PUT on the full target resource (and thus all the names accessible through it), access control rules must be evaluated for each record in the pack.

5. IANA Considerations

This document registers two new media types and CoAP Content-Format IDs for both media types.

Note to RFC Editor: Please replace all occurrences of "RFC-AAAA" with the RFC number of this document.

5.1. CoAP Content-Format Registration

IANA is requested to assign CoAP Content-Format IDs for the SenML PATCH and FETCH media types in the "CoAP Content-Formats" sub-registry, within the "CoRE Parameters" registry [RFC7252]. The assigned IDs are shown in Table 1.

| Media type | Encoding | ID |
|-----------------------------|----------|---------|
| application/senml-etch+json | - | TBD-320 |
| application/senml-etch+cbor | - | TBD-322 |

Table 1: CoAP Content-Format IDs

5.2. senml-etch+json Media Type

Type name: application

Subtype name: senml-etch+json

Required parameters: none

Optional parameters: none

Encoding considerations: Must be encoded as using a subset of the encoding allowed in [RFC8259]. This simplifies implementation of a very simple system and does not impose any significant limitations as all this data is meant for machine to machine communications and is not meant to be human readable.

Security considerations: See Section 4 of RFC-AAAA.

Interoperability considerations: Applications MUST ignore any key value pairs that they do not understand unless the key ends with the '_' character in which case an error MUST be generated. This allows backwards compatible extensions to this specification.

Published specification: RFC-AAAA

Applications that use this media type: Applications that use the SenML media type for resource representation.

Fragment identifier considerations: N/A

Additional information:

Magic number(s): none

File extension(s): senml-etchj

Windows Clipboard Name: "SenML FETCH/PATCH format"

Macintosh file type code(s): none

Macintosh Universal Type Identifier code: org.ietf.senml-etch-json
conforms to public.text

Person & email address to contact for further information: Ari
Keranen ari.keranen@ericsson.com

Intended usage: COMMON

Restrictions on usage: None

Author: Ari Keranen ari.keranen@ericsson.com

Change controller: IESG

5.3. senml-etch+cbor Media Type

Type name: application

Subtype name: senml-etch+cbor

Required parameters: none

Optional parameters: none

Encoding considerations: Must be encoded as using [RFC7049].

Security considerations: See Section 4 of RFC-AAAA.

Interoperability considerations: Applications MUST ignore any key value pairs that they do not understand unless the key ends with the '_' character in which case an error MUST be generated. This allows backwards compatible extensions to this specification.

Published specification: RFC-AAAA

Applications that use this media type: Applications that use the SenML media type for resource representation.

Fragment identifier considerations: N/A

Additional information:

Magic number(s): none

File extension(s): senml-etchc

Macintosh file type code(s): none

Macintosh Universal Type Identifier code: org.ietf.senml-etch-cbor
conforms to public.data

Person & email address to contact for further information: Ari
Keranen ari.keranen@ericsson.com

Intended usage: COMMON

Restrictions on usage: None

Author: Ari Keranen ari.keranen@ericsson.com

Change controller: IESG

6. Acknowledgements

The use of FETCH and (i)PATCH methods with SenML was first introduced by the OMA SpecWorks LwM2M v1.1 specification. This document generalizes the use to any SenML representation. The authors would like to thank Carsten Bormann, Christian Amsuess, Jaime Jimenez, Klaus Hartke, Michael Richardson, and other participants from the IETF CoRE and OMA SpecWorks DMSE working groups who have contributed ideas and reviews.

7. References

7.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://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, <<https://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, <<https://www.rfc-editor.org/info/rfc7252>>.
- [RFC8132] van der Stok, P., Bormann, C., and A. Sehgal, "PATCH and FETCH Methods for the Constrained Application Protocol (CoAP)", RFC 8132, DOI 10.17487/RFC8132, April 2017, <<https://www.rfc-editor.org/info/rfc8132>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <<https://www.rfc-editor.org/info/rfc8259>>.
- [RFC8428] Jennings, C., Shelby, Z., Arkko, J., Keranen, A., and C. Bormann, "Sensor Measurement Lists (SenML)", RFC 8428, DOI 10.17487/RFC8428, August 2018, <<https://www.rfc-editor.org/info/rfc8428>>.

7.2. Informative References

- [IPSO] IPSO, "IPSO Light Control Smart Object", 2018, <<http://www.openmobilealliance.org/tech/profiles/lwm2m/3311.xml>>.

Authors' Addresses

Ari Keranen
Ericsson

Email: ari.keranen@ericsson.com

Mojan Mohajer
u-blox UK

Email: Mojan.Mohajer@u-blox.com

Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: January 9, 2020

M. Veillette, Ed.
Trilliant Networks Inc.
A. Pelov, Ed.
I. Petrov, Ed.
Acklio
July 08, 2019

YANG Schema Item iDentifier (SID)
draft-ietf-core-sid-07

Abstract

YANG Schema Item iDentifiers (SID) are globally unique 64-bit unsigned integers used to identify YANG items. This document defines the semantics, the registration, and assignment processes of SIDs. To enable the implementation of these processes, this document also defines a file format used to persist and publish assigned SIDs.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on January 9, 2020.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of

the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

| | |
|---|----|
| 1. Introduction | 2 |
| 2. Terminology and Notation | 3 |
| 3. ".sid" file lifecycle | 4 |
| 4. ".sid" file format | 5 |
| 5. Security Considerations | 9 |
| 6. IANA Considerations | 9 |
| 6.1. Register SID File Format Module | 9 |
| 6.2. Create new IANA Registry: "SID Mega-Range" registry . . . | 9 |
| 6.2.1. Structure | 9 |
| 6.2.2. Allocation policy | 10 |
| 6.2.2.1. First allocation | 10 |
| 6.2.2.2. Consecutive allocations | 11 |
| 6.2.3. Initial contents of the Registry | 11 |
| 6.3. Create a new IANA Registry: IETF SID Range Registry (managed by IANA) | 11 |
| 6.3.1. Structure | 11 |
| 6.3.2. Allocation policy | 11 |
| 6.3.3. Initial contents of the registry | 12 |
| 6.4. Create new IANA Registry: "IETF SID Registry" | 13 |
| 6.4.1. Structure | 13 |
| 6.4.2. Allocation policy | 14 |
| 6.4.3. Initial contents of the registry | 14 |
| 7. Acknowledgments | 14 |
| 8. References | 14 |
| 8.1. Normative References | 14 |
| 8.2. Informative References | 15 |
| Appendix A. ".sid" file example | 16 |
| Appendix B. SID auto generation | 25 |
| Appendix C. ".sid" file lifecycle | 25 |
| C.1. SID File Creation | 26 |
| C.2. SID File Update | 27 |
| Authors' Addresses | 28 |

1. Introduction

Some of the items defined in YANG [RFC7950] require the use of a unique identifier. In both NETCONF [RFC6241] and RESTCONF [RFC8040], these identifiers are implemented using names. To allow the implementation of data models defined in YANG in constrained devices and constrained networks, a more compact method to identify YANG items is required. This compact identifier, called SID, is encoded using a 64-bit unsigned integer. The following items are identified using SIDs:

- o identities
- o data nodes (Note: including those parts of a YANG template as defined by the 'yang-data' extension.)
- o RPCs and associated input(s) and output(s)
- o actions and associated input(s) and output(s)
- o notifications and associated information
- o YANG modules, submodules and features

SIDs are globally unique integers, a registration system is used in order to guarantee their uniqueness. SIDs are registered in blocks called "SID ranges".

Assignment of SIDs to YANG items can be automated. For more details how this could be achieved, please consult Appendix B.

SIDs are assigned permanently, items introduced by a new revision of a YANG module are added to the list of SIDs already assigned.

Section 3 provides more details about the registration process of YANG modules and associated SIDs. To enable the implementation of this registry, Section 4 defines a standard file format used to store and publish SIDs.

2. Terminology and Notation

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 BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

The following terms are defined in [RFC7950]:

- o action
- o feature
- o module
- o notification
- o RPC

- o schema node
- o schema tree
- o submodule

The following term is defined in [RFC8040]:

- o yang-data extension

This specification also makes use of the following terminology:

- o item: A schema node, an identity, a module, a submodule or a feature defined using the YANG modeling language.
- o path: A path is a string that identifies a schema node within the schema tree. A path consists of the list of schema node identifier(s) separated by slashes ("/"). Schema node identifier(s) are always listed from the top-level schema node up to the targeted schema node. (e.g. "/ietf-system:system-state/clock/current-datetime")
- o YANG Schema Item iDentifier (SID): Unsigned integer used to identify different YANG items.

3. ".sid" file lifecycle

YANG is a language designed to model data accessed using one of the compatible protocols (e.g. NETCONF [RFC6241], RESCONF [RFC8040] and CoRECONF [I-D.ietf-core-comi]). A YANG module defines hierarchies of data, including configuration, state data, RPCs, actions and notifications.

Many YANG modules are not created in the context of constrained applications. YANG modules can be implemented using NETCONF [RFC6241] or RESTCONF [RFC8040] without the need to assign SIDs.

As needed, authors of YANG modules can assign SIDs to their YANG modules. In order to do that, they should first obtain a SID range from a registry and use that range to assign or generate SIDs to items of their YANG module. For example how this could be achieved, please refer to Appendix C.

Registration of the .sid file associated to a YANG module is optional but recommended to promote interoperability between devices and to avoid duplicate allocation of SIDs to a single YANG module. Different registries might have different requirements for the registration and publication of the ".sid" files. For diagram of one

of the possibilities, please refer to the activity diagram on Figure 1 in Appendix C.

Each time a YANG module or one of its imported module(s) or included sub-module(s) is updated, the ".sid" file MAY need to be updated. This update SHOULD also be performed using an automated tool.

If a new revision requires more SIDs than initially allocated, a new SID range MUST be added to the 'assignment-ranges' as defined in Section 4. These extra SIDs are used for subsequent assignments.

For an example of this update process, see activity diagram Figure 2 in Appendix C.

4. ".sid" file format

".sid" files are used to persist and publish SIDs assigned to the different YANG items of a specific YANG module. The following YANG module defined the structure of this file, encoding is performed using the rules defined in [RFC7951].

```
<CODE BEGINS> file "ietf-sid-file@2017-11-26.yang"
module ietf-sid-file {
  namespace "urn:ietf:params:xml:ns:yang:ietf-sid-file";
  prefix sid;

  import ietf-yang-types {
    prefix yang;
  }

  organization
    "IETF Core Working Group";

  contact
    "Michel Veillette
    <mailto:michel.veillette@trilliant.com>

    Andy Bierman
    <mailto:andy@yumaworks.com>

    Alexander Pelov
    <mailto:a@ackl.io>";

  description
    "This module defines the structure of the .sid files.

    Each .sid file contains the mapping between the different
    string identifiers defined by a YANG module and a
```



```
    corresponding numeric value called SID.";

revision 2017-11-26 {
    description
        "Initial revision.";
    reference
        "[I-D.ietf-core-sid] YANG Schema Item iDentifier (SID)";
}

typedef revision-identifier {
    type string {
        pattern '\d{4}-\d{2}-\d{2}';
    }
    description
        "Represents a date in YYYY-MM-DD format.";
}

typedef sid {
    type uint64;
    description
        "YANG Schema Item iDentifier";
    reference
        "[I-D.ietf-core-sid] YANG Schema Item iDentifier (SID)";
}

typedef schema-node-path {
    type string {
        pattern
            '(/[a-zA-Z_][a-zA-Z0-9\-\_\.]*:[a-zA-Z_][a-zA-Z0-9\-\_\.]*' +
            '(/[a-zA-Z_][a-zA-Z0-9\-\_\.]*(:[a-zA-Z_][a-zA-Z0-9\-\_\.]*)?)*)*';
    }
    description
        "Identifies a schema-node path string for use in the
        SID registry. This string format follows the rules
        for an instance-identifier, as defined in RFC 7959,
        except that no predicates are allowed.

        This format is intended to support the YANG 1.1 ABNF
        for a schema node identifier, except module names
        are used instead of prefixes, as specified in RFC 7951.";
    reference
        "RFC 7950, The YANG 1.1 Data Modeling Language;
        Section 6.5: Schema Node Identifier;
        RFC 7951, JSON Encoding of YANG Data;
        Section 6.11: The instance-identifier type";
}

leaf module-name {
```

```
    type yang:yang-identifier;
    description
      "Name of the YANG module associated with this .sid file.";
  }

  leaf module-revision {
    type revision-identifier;
    description
      "Revision of the YANG module associated with this .sid file.
      This leaf is not present if no revision statement is
      defined in the YANG module.";
  }

  list assignment-ranges {
    key "entry-point";
    description
      "SID range(s) allocated to the YANG module identified by
      'module-name' and 'module-revision'.";

    leaf entry-point {
      type sid;
      mandatory true;
      description
        "Lowest SID available for assignment.";
    }

    leaf size {
      type uint64;
      mandatory true;
      description
        "Number of SIDs available for assignment.";
    }
  }

  list items {
    key "namespace identifier";
    description
      "Each entry within this list defined the mapping between
      a YANG item string identifier and a SID. This list MUST
      include a mapping entry for each YANG item defined by
      the YANG module identified by 'module-name' and
      'module-revision'.";

    leaf namespace {
      type enumeration {
        enum module {
          value 0;
          description

```

```
        "All module and submodule names share the same
        global module identifier namespace.";
    }
    enum identity {
        value 1;
        description
            "All identity names defined in a module and its
            submodules share the same identity identifier
            namespace.";
    }
    enum feature {
        value 2;
        description
            "All feature names defined in a module and its
            submodules share the same feature identifier
            namespace.";
    }
    enum data {
        value 3;
        description
            "The namespace for all data nodes, as defined in YANG.";
    }
}
description
    "Namespace of the YANG item for this mapping entry.";
}

leaf identifier {
    type union {
        type yang:yang-identifier;
        type schema-node-path;
    }
    description
        "String identifier of the YANG item for this mapping entry.

        If the corresponding 'namespace' field is 'module',
        'feature', or 'identity', then this field MUST
        contain a valid YANG identifier string.

        If the corresponding 'namespace' field is 'data',
        then this field MUST contain a valid schema node
        path.";
}

leaf sid {
    type sid;
    mandatory true;
    description
```

```
        "SID assigned to the YANG item for this mapping entry.";
    }
}
}
<CODE ENDS>
```

5. Security Considerations

This document defines a new type of identifier used to encode data models defined in YANG [RFC7950]. As such, this identifier does not contribute to any new security issues in addition of those identified for the specific protocols or contexts for which it is used.

6. IANA Considerations

6.1. Register SID File Format Module

This document registers one YANG module in the "YANG Module Names" registry [RFC6020]:

- o name: ietf-sid-file
- o namespace: urn:ietf:params:xml:ns:yang:ietf-sid-file
- o prefix: sid
- o reference: [[THISRFC]]

6.2. Create new IANA Registry: "SID Mega-Range" registry

The name of this registry is "SID Mega-Range". This registry is used to record the delegation of the management of a block of SIDs to third parties (such as SDOs or registrars).

6.2.1. Structure

Each entry in this registry must include:

- o The entry point (first SID) of the registered SID block.
- o The size of the registered SID block. The size MUST be one million (1 000 000) SIDs.
- o The contact information of the requesting organization including:
 - * The policy of SID range allocations: Public, Private or Both.
 - * Organization name

- * URL

The information associated to the Organization name should not be publicly visible in the registry, but should be available. This information includes contact email and phone number and change controller email and phone number.

6.2.2. Allocation policy

The IANA policy for future additions to this registry is "Expert Review" [RFC8126].

An organization requesting to manage a SID Range (and thus have an entry in the SID Mega-Range Registry), must ensure the following capacities:

- o The capacity to manage and operate a SID Range Registry. A SID Range Registry MUST provide the following information for all SID Ranges allocated by the Registry:
 - * Entry Point of allocated SID Range
 - * Size of allocated SID Range
 - * Type: Public or Private
 - + Public Ranges MUST include at least a reference to the YANG module and ".sid" files for that SID Range.
 - + Private Ranges MUST be marked as "Private"
- o A Policy of allocation, which clearly identifies if the SID Range allocations would be Private, Public or Both.
- o Technical capacity to ensure the sustained operation of the registry for a period of at least 5 years. If Private Registrations are allowed, the period must be of at least 10 years.

6.2.2.1. First allocation

For a first allocation to be provided, the requesting organization must demonstrate a functional registry infrastructure.

6.2.2.2. Consecutive allocations

On subsequent allocation request(s), the organization must demonstrate the exhaustion of the prior range. These conditions need to be asserted by the assigned expert(s).

If that extra-allocation is done within 3 years from the last allocation, the experts need to discuss this request on the CORE working group mailing list and consensus needs to be obtained before allocating a new Mega-Range.

6.2.3. Initial contents of the Registry

The initial entry in this registry is allocated to IANA:

| Entry Point | Size | Allocation | Organization name | URL |
|-------------|---------|------------|-------------------|----------|
| 0 | 1000000 | Public | IANA | iana.org |

6.3. Create a new IANA Registry: IETF SID Range Registry (managed by IANA)

6.3.1. Structure

Each entry in this registry must include:

- o The SID range entry point.
- o The SID range size.
- o The YANG module name.
- o Document reference.

6.3.2. Allocation policy

The first million SIDs assigned to IANA is sub-divided as follows:

- o The range of 0 to 999 (size 1000) is "Reserved" as defined in [RFC8126].
- o The range of 1000 to 59,999 (size 59,000) is reserved for YANG modules defined in RFCs. The IANA policy for additions to this registry is "Expert Review" [RFC8126].

- * The Expert MUST verify that the YANG module for which this allocation is made has an RFC (existing RFC) OR is on track to become RFC (early allocation with a request from the WG chairs).
- o The SID range allocated for a YANG module can follow in one of the four categories:
 - * SMALL (50 SIDs)
 - * MEDIUM (100 SIDs)
 - * LARGE (250 SIDs)
 - * CUSTOM (requested by the YANG module author, with a maximum of 1000 SIDs). In all cases, the size of a SID range assigned to a YANG module should be at least 33% above the current number of YANG items. This headroom allows assignment within the same range of new YANG items introduced by subsequent revisions. A larger SID range size may be requested by the authors if this recommendation is considered insufficient. It is important to note that an additional SID range can be allocated to an existing YANG module if the initial range is exhausted.
- o The range of 60,000 to 99,999 (size 40,000) is reserved for experimental YANG modules. This range MUST NOT be used in operational deployments since these SIDs are not globally unique which limit their interoperability. The IANA policy for this range is "Experimental use" [RFC8126].
- o The range of 100,000 to 999,999 (size 900,000) is "Reserved" as defined in [RFC8126].

| Entry Point | Size | IANA policy |
|-------------|---------|------------------|
| 0 | 1,000 | Reserved |
| 1,000 | 59,000 | Expert Review |
| 60,000 | 40,000 | Experimental use |
| 100,000 | 900,000 | Reserved |

6.3.3. Initial contents of the registry

Initial entries in this registry are as follows:

| Entry Point | Size | Module name | Document reference |
|-------------|------|------------------|----------------------|
| 1000 | 100 | ietf-comi | [I-D.ietf-core-comi] |
| 1100 | 50 | ietf-yang-types | [RFC6991] |
| 1150 | 50 | ietf-inet-types | [RFC6991] |
| 1200 | 50 | iana-crypt-hash | [RFC7317] |
| 1250 | 50 | ietf-netconf-acm | [RFC8341] |
| 1300 | 50 | ietf-sid-file | RFCXXXX |
| 1500 | 100 | ietf-interfaces | [RFC8343] |
| 1600 | 100 | ietf-ip | [RFC8344] |
| 1700 | 100 | ietf-system | [RFC7317] |
| 1800 | 400 | iana-if-type | [RFC7224] |

// RFC Ed.: replace XXXX with RFC number assigned to this draft.

For allocation, RFC publication of the YANG module is required as per [RFC8126]. The YANG module must be registered in the "YANG module Name" registry according to the rules specified in section 14 of [RFC6020].

6.4. Create new IANA Registry: "IETF SID Registry"

The name of this registry is "IETF SID Registry". This registry is used to record the allocation of SIDs for individual YANG module items.

6.4.1. Structure

Each entry in this registry must include:

- o The YANG module name. This module name must be present in the "Name" column of the "YANG Module Names" registry.
- o A link to the associated ".yang" file. This file link must be present in the "File" column of the "YANG Module Names" registry.
- o The link to the ".sid" file which defines the allocation.
- o The number of actually allocated SIDs in the ".sid" file.

The ".sid" file is stored by IANA.

6.4.2. Allocation policy

The allocation policy is Expert review. The Expert MUST ensure that the following conditions are met:

- o The ".sid" file has a valid structure:
 - * The ".sid" file MUST be a valid JSON file following the structure of the module defined in RFCXXXX (RFC Ed: replace XXX with RFC number assigned to this draft).
- o The ".sid" file allocates individual SIDs ONLY in the SID Ranges for this YANG module (as allocated in the IETF SID Range Registry):
 - * All SIDs in this ".sid" file MUST be within the ranges allocated to this YANG module in the "IETF SID Range Registry".
- o If another ".sid" file has already allocated SIDs for this YANG module (e.g. for older or newer versions of the YANG module), the YANG items are assigned the same SIDs as in the the other ".sid" file.
- o SIDs never change.

6.4.3. Initial contents of the registry

None.

7. Acknowledgments

The authors would like to thank Andy Bierman, Carsten Bormann, Abhinav Somaraju, Laurent Toutain, Randy Turner and Peter van der Stok for their help during the development of this document and their useful comments during the review process.

8. References

8.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://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, <<https://www.rfc-editor.org/info/rfc7049>>.

- [RFC7120] Cotton, M., "Early IANA Allocation of Standards Track Code Points", BCP 100, RFC 7120, DOI 10.17487/RFC7120, January 2014, <<https://www.rfc-editor.org/info/rfc7120>>.
- [RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <<https://www.rfc-editor.org/info/rfc7950>>.
- [RFC7951] Lhotka, L., "JSON Encoding of Data Modeled with YANG", RFC 7951, DOI 10.17487/RFC7951, August 2016, <<https://www.rfc-editor.org/info/rfc7951>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.

8.2. Informative References

- [I-D.ietf-core-comi] Veillette, M., Stok, P., Pelov, A., and A. Bierman, "CoAP Management Interface", draft-ietf-core-comi-05 (work in progress), May 2019.
- [RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010, <<https://www.rfc-editor.org/info/rfc6020>>.
- [RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <<https://www.rfc-editor.org/info/rfc6241>>.
- [RFC6991] Schoenwaelder, J., Ed., "Common YANG Data Types", RFC 6991, DOI 10.17487/RFC6991, July 2013, <<https://www.rfc-editor.org/info/rfc6991>>.
- [RFC7224] Bjorklund, M., "IANA Interface Type YANG Module", RFC 7224, DOI 10.17487/RFC7224, May 2014, <<https://www.rfc-editor.org/info/rfc7224>>.
- [RFC7317] Bierman, A. and M. Bjorklund, "A YANG Data Model for System Management", RFC 7317, DOI 10.17487/RFC7317, August 2014, <<https://www.rfc-editor.org/info/rfc7317>>.
- [RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <<https://www.rfc-editor.org/info/rfc8040>>.

- [RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <<https://www.rfc-editor.org/info/rfc8126>>.
- [RFC8341] Bierman, A. and M. Bjorklund, "Network Configuration Access Control Model", STD 91, RFC 8341, DOI 10.17487/RFC8341, March 2018, <<https://www.rfc-editor.org/info/rfc8341>>.
- [RFC8343] Bjorklund, M., "A YANG Data Model for Interface Management", RFC 8343, DOI 10.17487/RFC8343, March 2018, <<https://www.rfc-editor.org/info/rfc8343>>.
- [RFC8344] Bjorklund, M., "A YANG Data Model for IP Management", RFC 8344, DOI 10.17487/RFC8344, March 2018, <<https://www.rfc-editor.org/info/rfc8344>>.

Appendix A. ".sid" file example

The following .sid file (ietf-system@2014-08-06.sid) have been generated using the following yang modules:

- o ietf-system@2014-08-06.yang
- o ietf-yang-types@2013-07-15.yang
- o ietf-inet-types@2013-07-15.yang
- o ietf-netconf-acm@2012-02-22.yang
- o iana-crypt-hash@2014-04-04.yang

```
{
  "assignment-ranges": [
    {
      "entry-point": 1700,
      "size": 100
    }
  ],
  "module-name": "ietf-system",
  "module-revision": "2014-08-06",
  "items": [
    {
      "namespace": "module",
      "identifier": "ietf-system",
      "sid": 1700
    }
  ],
}
```

```
{
  "namespace": "identity",
  "identifier": "authentication-method",
  "sid": 1701
},
{
  "namespace": "identity",
  "identifier": "local-users",
  "sid": 1702
},
{
  "namespace": "identity",
  "identifier": "radius",
  "sid": 1703
},
{
  "namespace": "identity",
  "identifier": "radius-authentication-type",
  "sid": 1704
},
{
  "namespace": "identity",
  "identifier": "radius-chap",
  "sid": 1705
},
{
  "namespace": "identity",
  "identifier": "radius-pap",
  "sid": 1706
},
{
  "namespace": "feature",
  "identifier": "authentication",
  "sid": 1707
},
{
  "namespace": "feature",
  "identifier": "dns-udp-tcp-port",
  "sid": 1708
},
{
  "namespace": "feature",
  "identifier": "local-users",
  "sid": 1709
},
{
  "namespace": "feature",
  "identifier": "ntp",
```

```
    "sid": 1710
  },
  {
    "namespace": "feature",
    "identifier": "ntp-udp-port",
    "sid": 1711
  },
  {
    "namespace": "feature",
    "identifier": "radius",
    "sid": 1712
  },
  {
    "namespace": "feature",
    "identifier": "radius-authentication",
    "sid": 1713
  },
  {
    "namespace": "feature",
    "identifier": "timezone-name",
    "sid": 1714
  },
  {
    "namespace": "data",
    "identifier": "/ietf-system:set-current-datetime",
    "sid": 1715
  },
  {
    "namespace": "data",
    "identifier": "/ietf-system:set-current-datetime/
      current-datetime",
    "sid": 1716
  },
  {
    "namespace": "data",
    "identifier": "/ietf-system:system",
    "sid": 1717
  },
  {
    "namespace": "data",
    "identifier": "/ietf-system:system-restart",
    "sid": 1718
  },
  {
    "namespace": "data",
    "identifier": "/ietf-system:system-shutdown",
    "sid": 1719
  },
}
```

```
{
  "namespace": "data",
  "identifier": "/ietf-system:system-state",
  "sid": 1720
},
{
  "namespace": "data",
  "identifier": "/ietf-system:system-state/clock",
  "sid": 1721
},
{
  "namespace": "data",
  "identifier": "/ietf-system:system-state/clock/boot-datetime",
  "sid": 1722
},
{
  "namespace": "data",
  "identifier": "/ietf-system:system-state/clock/
    current-datetime",
  "sid": 1723
},
{
  "namespace": "data",
  "identifier": "/ietf-system:system-state/platform",
  "sid": 1724
},
{
  "namespace": "data",
  "identifier": "/ietf-system:system-state/platform/machine",
  "sid": 1725
},
{
  "namespace": "data",
  "identifier": "/ietf-system:system-state/platform/os-name",
  "sid": 1726
},
{
  "namespace": "data",
  "identifier": "/ietf-system:system-state/platform/os-release",
  "sid": 1727
},
{
  "namespace": "data",
  "identifier": "/ietf-system:system-state/platform/os-version",
  "sid": 1728
},
{
  "namespace": "data",
```

```
    "identifier": "/ietf-system:system/authentication",
    "sid": 1729
  },
  {
    "namespace": "data",
    "identifier": "/ietf-system:system/authentication/user",
    "sid": 1730
  },
  {
    "namespace": "data",
    "identifier": "/ietf-system:system/authentication/
      user-authentication-order",
    "sid": 1731
  },
  {
    "namespace": "data",
    "identifier": "/ietf-system:system/authentication/user/
      authorized-key",
    "sid": 1732
  },
  {
    "namespace": "data",
    "identifier": "/ietf-system:system/authentication/user/
      authorized-key/algorithm",
    "sid": 1733
  },
  {
    "namespace": "data",
    "identifier": "/ietf-system:system/authentication/user/
      authorized-key/key-data",
    "sid": 1734
  },
  {
    "namespace": "data",
    "identifier": "/ietf-system:system/authentication/user/
      authorized-key/name",
    "sid": 1735
  },
  {
    "namespace": "data",
    "identifier": "/ietf-system:system/authentication/user/
      name",
    "sid": 1736
  },
  {
    "namespace": "data",
    "identifier": "/ietf-system:system/authentication/user/
      password",
```

```
    "sid": 1737
  },
  {
    "namespace": "data",
    "identifier": "/ietf-system:system/clock",
    "sid": 1738
  },
  {
    "namespace": "data",
    "identifier": "/ietf-system:system/clock/timezone-name",
    "sid": 1739
  },
  {
    "namespace": "data",
    "identifier": "/ietf-system:system/clock/timezone-utc-offset",
    "sid": 1740
  },
  {
    "namespace": "data",
    "identifier": "/ietf-system:system/contact",
    "sid": 1741
  },
  {
    "namespace": "data",
    "identifier": "/ietf-system:system/dns-resolver",
    "sid": 1742
  },
  {
    "namespace": "data",
    "identifier": "/ietf-system:system/dns-resolver/options",
    "sid": 1743
  },
  {
    "namespace": "data",
    "identifier": "/ietf-system:system/dns-resolver/options/
      attempts",
    "sid": 1744
  },
  {
    "namespace": "data",
    "identifier": "/ietf-system:system/dns-resolver/options/
      timeout",
    "sid": 1745
  },
  {
    "namespace": "data",
    "identifier": "/ietf-system:system/dns-resolver/search",
    "sid": 1746
  }
```



```
    },
    {
      "namespace": "data",
      "identifier": "/ietf-system:system/dns-resolver/server",
      "sid": 1747
    },
    {
      "namespace": "data",
      "identifier": "/ietf-system:system/dns-resolver/server/name",
      "sid": 1748
    },
    {
      "namespace": "data",
      "identifier": "/ietf-system:system/dns-resolver/server/
        udp-and-tcp",
      "sid": 1749
    },
    {
      "namespace": "data",
      "identifier": "/ietf-system:system/dns-resolver/server/
        udp-and-tcp/address",
      "sid": 1750
    },
    {
      "namespace": "data",
      "identifier": "/ietf-system:system/dns-resolver/server/
        udp-and-tcp/port",
      "sid": 1751
    },
    {
      "namespace": "data",
      "identifier": "/ietf-system:system/hostname",
      "sid": 1752
    },
    {
      "namespace": "data",
      "identifier": "/ietf-system:system/location",
      "sid": 1753
    },
    {
      "namespace": "data",
      "identifier": "/ietf-system:system/ntp",
      "sid": 1754
    },
    {
      "namespace": "data",
      "identifier": "/ietf-system:system/ntp/enabled",
      "sid": 1755
    }
  ],
  "leaf-ref": [
    {
      "namespace": "data",
      "identifier": "/ietf-system:system/ntp/enabled",
      "sid": 1755
    }
  ]
}
```

```
,
{
  "namespace": "data",
  "identifier": "/ietf-system:system/ntp/server",
  "sid": 1756
},
{
  "namespace": "data",
  "identifier": "/ietf-system:system/ntp/server/
    association-type",
  "sid": 1757
},
{
  "namespace": "data",
  "identifier": "/ietf-system:system/ntp/server/iburst",
  "sid": 1758
},
{
  "namespace": "data",
  "identifier": "/ietf-system:system/ntp/server/name",
  "sid": 1759
},
{
  "namespace": "data",
  "identifier": "/ietf-system:system/ntp/server/prefer",
  "sid": 1760
},
{
  "namespace": "data",
  "identifier": "/ietf-system:system/ntp/server/udp",
  "sid": 1761
},
{
  "namespace": "data",
  "identifier": "/ietf-system:system/ntp/server/udp/address",
  "sid": 1762
},
{
  "namespace": "data",
  "identifier": "/ietf-system:system/ntp/server/udp/port",
  "sid": 1763
},
{
  "namespace": "data",
  "identifier": "/ietf-system:system/radius",
  "sid": 1764
},
{
```

```
    "namespace": "data",
    "identifier": "/ietf-system:system/radius/options",
    "sid": 1765
  },
  {
    "namespace": "data",
    "identifier": "/ietf-system:system/radius/options/attempts",
    "sid": 1766
  },
  {
    "namespace": "data",
    "identifier": "/ietf-system:system/radius/options/timeout",
    "sid": 1767
  },
  {
    "namespace": "data",
    "identifier": "/ietf-system:system/radius/server",
    "sid": 1768
  },
  {
    "namespace": "data",
    "identifier": "/ietf-system:system/radius/server/
      authentication-type",
    "sid": 1769
  },
  {
    "namespace": "data",
    "identifier": "/ietf-system:system/radius/server/name",
    "sid": 1770
  },
  {
    "namespace": "data",
    "identifier": "/ietf-system:system/radius/server/udp",
    "sid": 1771
  },
  {
    "namespace": "data",
    "identifier": "/ietf-system:system/radius/server/udp/
      address",
    "sid": 1772
  },
  {
    "namespace": "data",
    "identifier": "/ietf-system:system/radius/server/udp/
      authentication-port",
    "sid": 1773
  },
  {

```

```
    "namespace": "data",
    "identifier": "/ietf-system:system/radius/server/udp/
                  shared-secret",
    "sid": 1774
  }
]
```

Appendix B. SID auto generation

Assignment of SIDs to YANG items can be automated, the recommended process to assign SIDs is as follows:

1. A tool extracts the different items defined for a specific YANG module.
2. The list of items is sorted in alphabetical order, 'namespace' in descending order, 'identifier' in ascending order. The 'namespace' and 'identifier' formats are described in the YANG module 'ietf-sid-file' defined in Section 4.
3. SIDs are assigned sequentially from the entry point up to the size of the registered SID range. This approach is recommended to minimize the serialization overhead, especially when delta between a reference SID and the current SID is used by protocols aiming to reduce message size.
4. If the number of items exceeds the SID range(s) allocated to a YANG module, an extra range is added for subsequent assignments.

Changes of SID files can also be automated using the same method described above, only unassigned YANG items are processed at step #3. Already existing items in the SID file should not be given new SIDs.

Appendix C. ".sid" file lifecycle

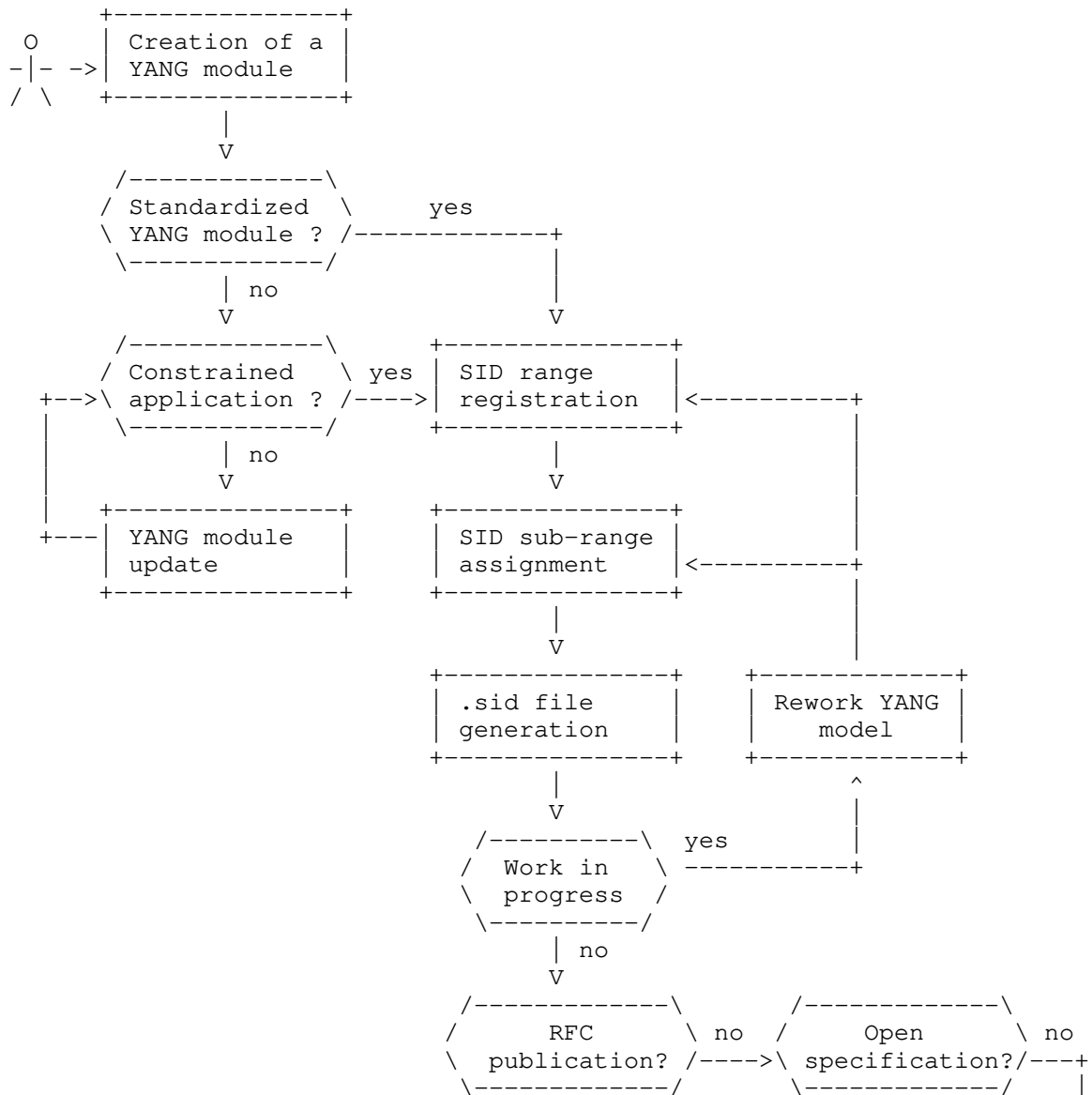
Before assigning SIDs to their YANG modules, YANG module authors must acquire a SID range from a "SID Range Registry". If the YANG module is part of an IETF draft or RFC, the SID range need to be acquired from the "IETF SID Range Registry" as defined in Section 6.3. For the other YANG modules, the authors can acquire a SID range from any "SID Range Registry" of their choice.

Once the SID range is acquired, the owner can use it to generate ".sid" file/s for his YANG module/s. It is recommended to leave some unallocated SIDs following the allocated range in each ".sid" file in order to allow better evolution of the YANG module in the future. Generation of ".sid" files should be performed using an automated

tool. Note that ".sid" files can only be generated for YANG modules and not for submodules.

C.1. SID File Creation

The following activity diagram summarizes the creation of a YANG module and its associated .sid file.



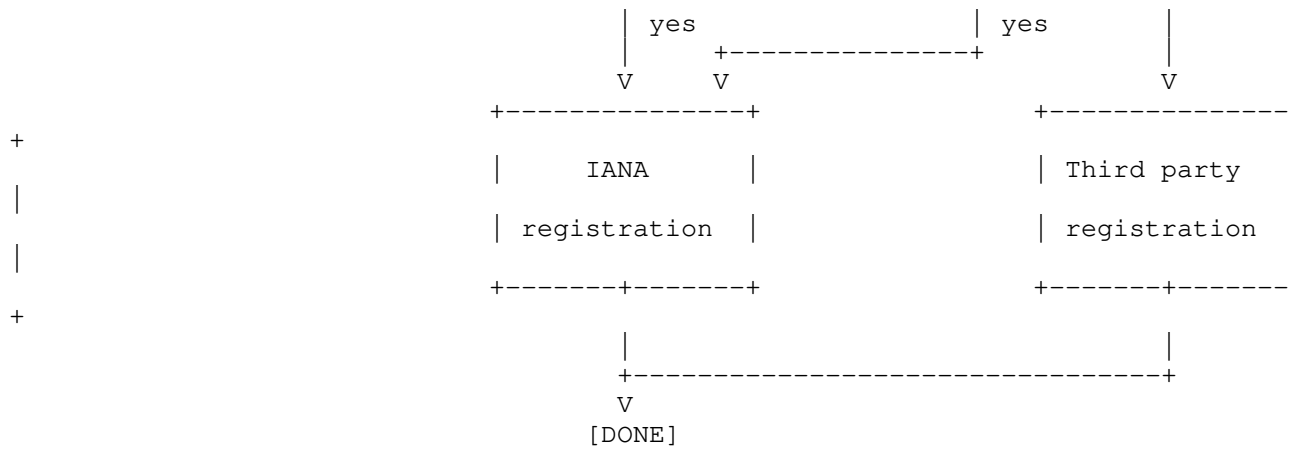


Figure 1: SID Lifecycle

C.2. SID File Update

The following Activity diagram summarizes the update of a YANG module and its associated .sid file.

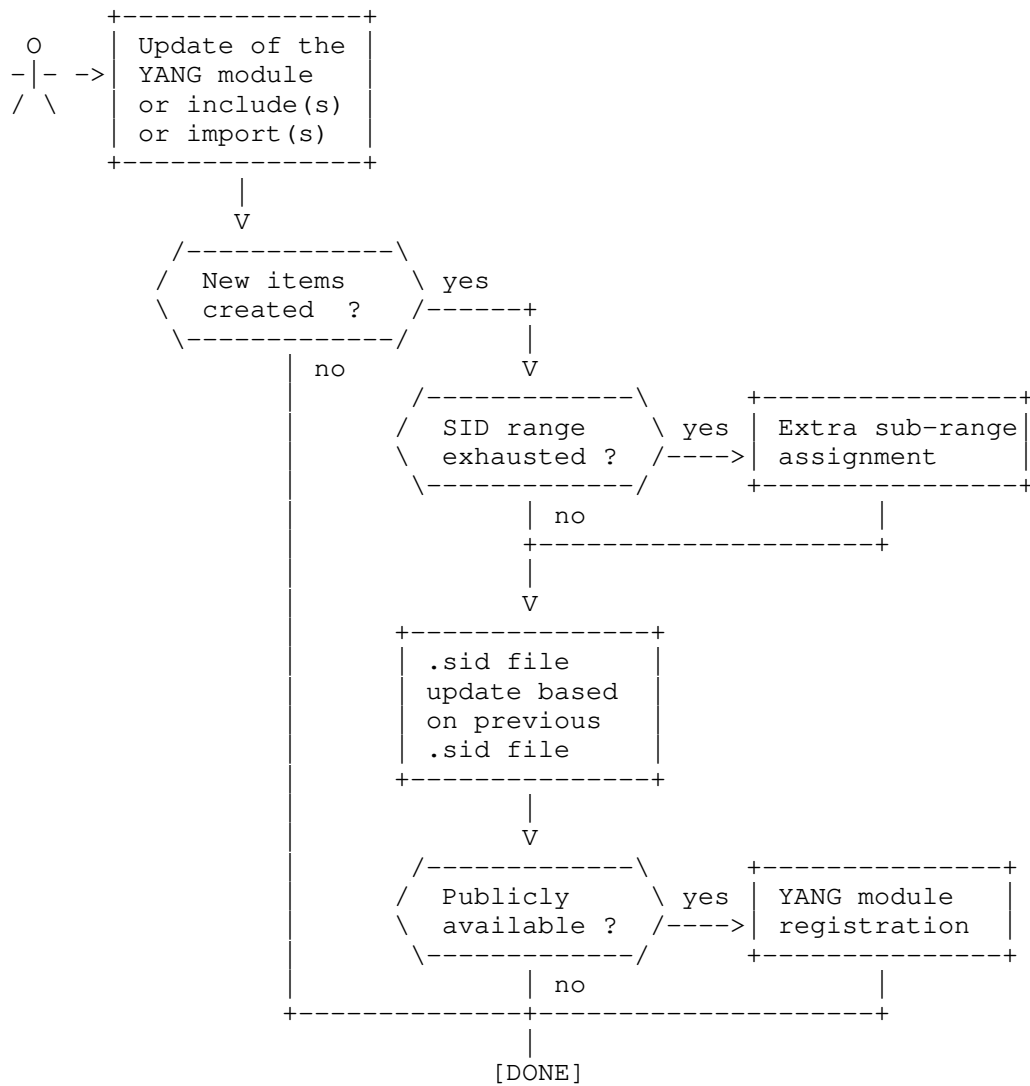


Figure 2: YANG and SID file update

Authors' Addresses

Michel Veillette (editor)
Trilliant Networks Inc.
610 Rue du Luxembourg
Granby, Quebec J2J 2V2
Canada

Phone: +14503750556
Email: michel.veillette@trilliant.com

Alexander Pelov (editor)
Acklio
1137A avenue des Champs Blancs
Cesson-Sevigne, Bretagne 35510
France

Email: a@ackl.io

Ivaylo Petrov (editor)
Acklio
1137A avenue des Champs Blancs
Cesson-Sevigne, Bretagne 35510
France

Email: ivaylo@ackl.io

CoRE Working Group
Internet-Draft
Updates: 7252, 8323 (if approved)
Intended status: Standards Track
Expires: September 12, 2019

K. Hartke
Ericsson
March 11, 2019

Extended Tokens and Stateless Clients
in the Constrained Application Protocol (CoAP)
draft-ietf-core-stateless-01

Abstract

This document provides considerations for alleviating CoAP clients and intermediaries of keeping per-request state. To facilitate this, this document additionally introduces a new, optional CoAP protocol extension for extended token lengths.

This document updates RFCs 7252 and 8323.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 12, 2019.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must

include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

| | |
|---|----|
| 1. Introduction | 2 |
| 1.1. Terminology | 4 |
| 2. Extended Tokens | 4 |
| 2.1. Extended Token Length (TKL) Field | 4 |
| 2.2. Discovering Support | 5 |
| 2.2.1. Extended-Token-Lengths Capability Option | 5 |
| 2.2.2. Trial and Error | 5 |
| 2.3. Intermediaries | 6 |
| 3. Stateless Clients | 7 |
| 3.1. Intermediaries | 7 |
| 3.2. Extended Tokens | 8 |
| 3.3. Message Transmission | 9 |
| 4. Security Considerations | 10 |
| 4.1. Extended Tokens | 10 |
| 4.2. Stateless Clients | 10 |
| 4.2.1. Recommended Algorithms | 11 |
| 5. IANA Considerations | 11 |
| 5.1. CoAP Signaling Option Number | 11 |
| 6. References | 11 |
| 6.1. Normative References | 11 |
| 6.2. Informative References | 12 |
| Appendix A. Updated Message Formats | 12 |
| A.1. CoAP over UDP | 13 |
| A.2. CoAP over TCP | 14 |
| A.3. CoAP over WebSockets | 15 |
| Acknowledgements | 16 |
| Author's Address | 16 |

1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] is a RESTful application-layer protocol for constrained environments [RFC7228]. In CoAP, clients (or intermediaries in the client role) make requests to servers (or intermediaries in the server role), which serve the requests by returning responses.

While a request is ongoing, a client typically needs to keep some state that it requires for processing the response when it arrives. Identification of this state is done by means of a `_token_` in CoAP, an opaque sequence of bytes chosen by the client and included in the CoAP request. The server returns the token verbatim in any resulting CoAP response (Figure 1).

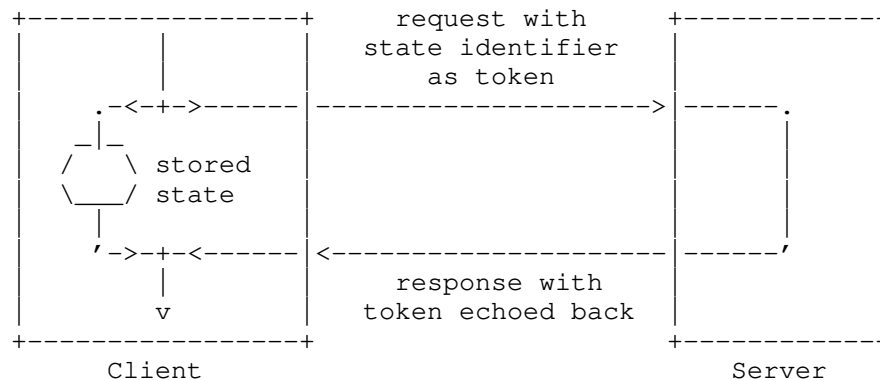


Figure 1: Token as an Identifier for Request State

In some scenarios, it can be beneficial to reduce the amount of state that is stored at the client at the cost of increased message sizes. Clients can implement this by serializing (parts of) their state into the token itself and recovering the state from the token in the response (Figure 2).

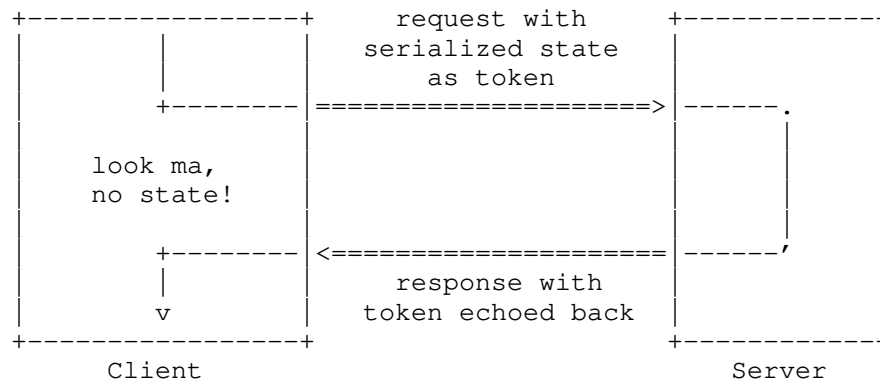


Figure 2: Token as Serialization of Request State

Section 3 of this document provides considerations for making clients "stateless" in this way, i.e., for avoiding per-request state in client implementations. (They'll still need to maintain per-server state and other kinds of state, so they're not entirely stateless.)

Serializing state into tokens is complicated by the fact that both CoAP over UDP [RFC7252] and CoAP over reliable transports [RFC8323] limit the maximum token length to 8 bytes. To overcome this limitation, Section 2 of this document first introduces a CoAP protocol extension for extended token lengths.

While the use case (avoiding per-request state) and the mechanism (extended token lengths) presented in this document are closely related, both can be used independently of each other: Some implementations may be able to fit their state in just 8 bytes; some implementations may have other use cases for extended token lengths.

1.1. Terminology

Stateless

In this document, "stateless" refers to an implementation strategy for a client (or intermediary in the client role) that doesn't require it to keep state for the individual requests it sends to a server (or intermediary in the server role). The client still needs to keep state for each server it communicates with (such as state for generating tokens and congestion control), so it's not free of any state.

Client

In this document, "client" generally refers to any sender of a request and recipient of a response, including intermediaries.

Server

In this document, "server" generally refers to any recipient of a request and sender of a response, including intermediaries.

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 BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

2. Extended Tokens

2.1. Extended Token Length (TKL) Field

This document updates the message formats defined for CoAP over UDP [RFC7252] and CoAP over TCP, TLS, and WebSockets [RFC8323] with the following new definition of the TKL field, increasing the maximum token length to 65804 bytes.

Token Length (TKL): 4-bit unsigned integer. A value between 0 and 12 inclusive indicates the length of the variable-length Token field in bytes. Three values are reserved for special constructs:

13: An 8-bit unsigned integer precedes the Token field and indicates the length of the Token field minus 13.

14: A 16-bit unsigned integer in network byte order precedes the Token field and indicates the length of the Token field minus 269.

15: Reserved. This value MUST NOT be sent and MUST be processed as a message format error.

All other fields retain their definition.

The updated message formats are illustrated in Appendix A.

2.2. Discovering Support

Extended token lengths require support from the server or, if there are one or more intermediaries between the client and the server, the intermediary in the server role that the client is interacting with.

Support can be discovered by a client (or intermediary in the client role) in one of two ways: In case Capabilities and Settings Messages (CSMs) are available, such as in CoAP over TCP, support can be discovered using the Extended-Token-Lengths Capability Option defined in Section 2.2.1. Otherwise, such as in CoAP over UDP, support can only be discovered by trial and error, as described in Section 2.2.2.

2.2.1. Extended-Token-Lengths Capability Option

A sender can use the elective Extended-Token-Lengths Capability Option to indicate its support for the new TKL field definition specified in Section 2.1.

| # | C | R | Appli es to | Name | Forma t | Length | Base Value |
|---------|---|---|----------------|----------------------------|------------|--------|---------------|
| TB D | | | CSM | Extended-Token- Lengths | empty | 0 | (none) |

C=Critical, R=Repeatable

Table 1: The Extended-Token-Lengths Capability Option

2.2.2. Trial and Error

A request with a TKL field value outside the range from 0 to 8 will be considered a message format error (Section 3 of RFC 7252) and be rejected by a recipient that does not support the updated TKL field definition. A client thus can determine support by sending a request

with an extended token length and checking whether it's rejected by the recipient or not.

In CoAP over UDP, a recipient rejects a malformed confirmable message by sending a Reset message (Section 4.2 of RFC 7252). In case of a non-confirmable message, sending a Reset message is permitted but not required (Section 4.3 of RFC 7252). It is therefore RECOMMENDED that clients use a confirmable message for determining support.

As per RFC 7252, Reset messages are empty and don't contain a token; they only return the Message ID (Figure 3). They also don't contain any indication of what caused a message format error. It is therefore RECOMMENDED that clients use a request that contains no potential message format error other than the extended token length.

In CoAP over TCP, TLS, and WebSockets, a recipient rejects a malformed message by sending an Abort message and shutting down the connection (Section 5.6 of RFC 8323).

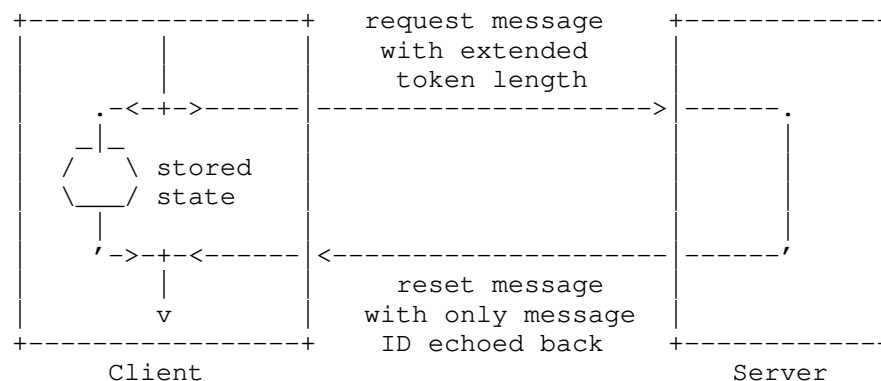


Figure 3: A Confirmable Request With an Extended Token is Rejected With a Reset Message if the Next Hop Does Not Support It

If a server supports extended token lengths but receives a request with a token of a length it is unwilling or unable to process, it MUST NOT reject the message. Instead, it SHOULD return a 4.00 (Bad Request) response. This implies that the server returns the entire token verbatim.

2.3. Intermediaries

Tokens are a hop-by-hop feature: When an intermediary receives a request, the only requirement is that it echoes the token back in any resulting response. There is no requirement or expectation that an intermediary passes a client's token on to a server or that an

intermediary uses extended token lengths itself when receiving a request with an extended token length.

3. Stateless Clients

A client can be alleviated of keeping per-request state by serializing the state into a sequence of bytes and sending the bytes as the token of the request. The server will return the token in the response to the client, so that the client can recover the state and process the response as if it had kept the state locally.

The format of the serialized state is an implementation detail of the client and opaque to any server implementation. However, using tokens to serialize state has significant and non-obvious security and privacy implications that need to be mitigated; see Section 4.

3.1. Intermediaries

Tokens are a hop-by-hop feature: If a client makes a request to an intermediary, that intermediary needs to store the client's token (along with the client's transport address) while it makes its own request to the next hop towards the origin server and waits for the response. When the intermediary receives the response, it looks up the client's token and transport address for the ongoing request and sends an appropriate response to the client.

Such an intermediary might want to be "stateless" as well, i.e., be alleviated of storing the client's token and transport address for ongoing requests. This can be implemented by serializing this information along the request state into the token to the next hop. When the next hop returns the response, the intermediary can recover the information from the token and use it to satisfy the client's request.

The downside of this approach is that an intermediary, without keeping request state, is unable to aggregate multiple requests for the same target resource, which reduces efficiency.

When multiple clients observe [RFC7641] the same resource, aggregating requests is REQUIRED (Section 3.1 of RFC 7641). As this cannot be satisfied without keeping request state, an intermediary MUST NOT include an Observe Option in requests it sends without keeping request state.

When using blockwise transfers [RFC7959], a server might not be able to distinguish blocks originating from different clients once they have been forwarded by an intermediary. To ensure that this does not lead to inconsistent resource state, a stateless intermediary MUST

include the Request-Tag Option [I-D.ietf-core-echo-request-tag] in blockwise transfers with a value that uniquely identifies the next hop towards the client in the intermediary's namespace.

3.2. Extended Tokens

A client (or intermediary in the role of a client) that depends on support for extended token lengths (Section 2) from the next hop to avoid keeping request state **MUST** perform a discovery of support (Section 2.2) before it can be stateless. This discovery **MUST** be performed in a stateful way, i.e., keeping state for the request (Figure 4): If the client was stateless from the start and the next hop doesn't support extended tokens, then any error message couldn't be processed since the state would neither be present at the client nor returned in the Reset message (Figure 5).

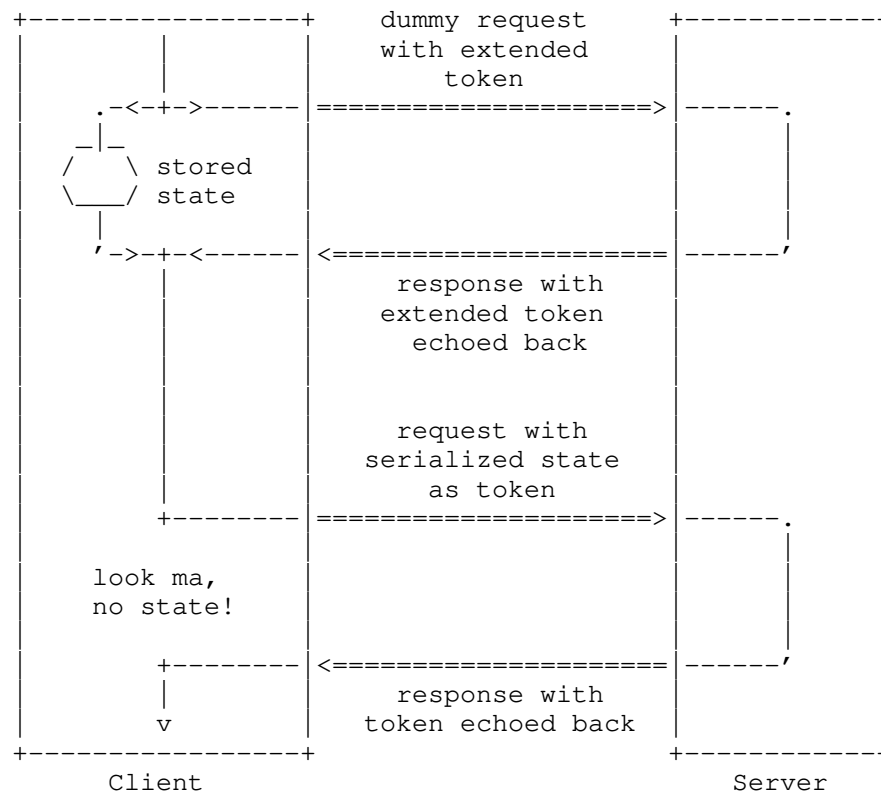


Figure 4: Depending on Extended Tokens for Being Stateless First Requires a Successful Stateful Discovery of Support

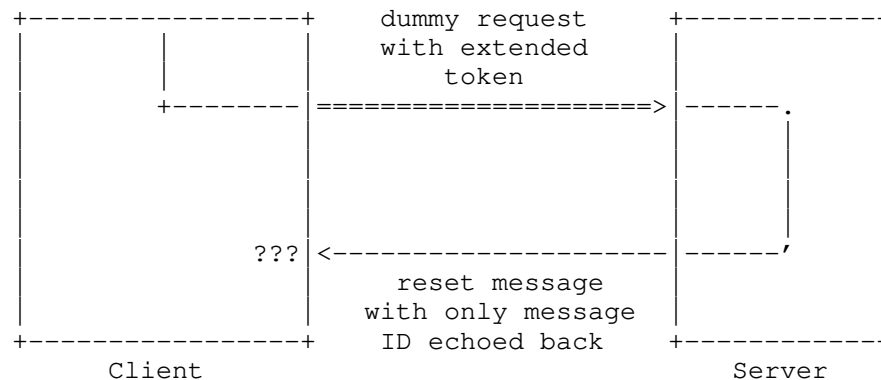


Figure 5: Stateless Discovery of Support Does Not Work

3.3. Message Transmission

As a further step, in the case of CoAP over UDP [RFC7252], a client (or intermediary in the client role) might want to also avoid keeping message transmission state.

Generally, a client can use confirmable or non-confirmable messages for requests. When using confirmable messages, it needs to keep message exchange state for performing retransmissions and handling Acknowledgement and Reset messages. When using non-confirmable messages, it can keep no message exchange state. However, in either case the client needs to keep congestion control state. That is, it needs to maintain state for each node it communicates with and, e.g., enforce NSTART.

As per RFC 7252, a client must be prepared to receive a response as a piggybacked response, a separate response or non-confirmable response (Section 5.2 of RFC 7252), regardless of the message type used for the request. A stateless client needs to handle these response types as follows:

- o If a piggybacked response contains a valid authentication tag and freshness indicator in the token, the client MUST process the message as specified in RFC 7252; otherwise, it MUST silently ignore the message.
- o If a separate response contains a valid authentication tag and freshness indicator in the token, the client MUST process the message as specified in RFC 7252; otherwise, it MUST reject the message as specified in Section 4.2 of RFC 7252.

- o If a non-confirmable response contains a valid authentication tag and freshness indicator in the token, the client MUST process the message as specified in RFC 7252; otherwise, it MUST reject the message as specified in Section 4.3 of RFC 7252.

4. Security Considerations

4.1. Extended Tokens

Tokens significantly larger than the 8 bytes specified in RFC 7252 have implications for nodes in particular with constrained memory size that need to be mitigated.

A node in the server role supporting extended token lengths may be vulnerable to a denial-of-service when an attacker (either on-path or a malicious client) sends large tokens to fill up the memory of the node. Implementations MUST be prepared for this and mitigate it.

4.2. Stateless Clients

Transporting the state needed by a client to process a response as serialized state information in the token has several significant and non-obvious security and privacy implications that need to be mitigated.

Serialized state information is an attractive target for both unwanted nodes (attackers between the node in client role and the next hop) and wanted nodes (the next hop itself) on the path. Therefore, a node in the client role MUST integrity protect the state information, unless processing a response does not modify state or cause other significant side effects.

Even when the serialized state is integrity protected, an attacker may still replay a response, making the client believe it sent the same request twice. Therefore, the node in client role MUST implement replay protection (e.g., by using sequence numbers and a replay window), unless processing a response does not modify state or cause other significant side effects. Integrity protection is REQUIRED for replay protection.

If processing a response without keeping request state is sensitive to the time elapsed to sending the request, then the serialized state MUST include freshness information (e.g., a timestamp).

Information in the serialized state may be privacy sensitive. A node in client role MUST encrypt the serialized state if it contains privacy sensitive information that an attacker would not get otherwise. For example, an intermediary that serializes the client's

token and transport address into its token leaks that information to the next hop, which may be undesirable. In wireless mesh networks, where all traffic is visible to a passive attacker, encryption may not be needed as the attacker can get the same information from analyzing the traffic flows.

4.2.1. Recommended Algorithms

The use of encryption, integrity protection, and replay protection of serialized state is recommended in general, unless a careful analysis of any potential attacks to security and privacy is performed. AES-CCM with a 64 bit tag is recommended, combined with a sequence number and a replay window. Where encryption is not needed, HMAC-SHA-256, combined with a sequence number and a replay window, may be used.

5. IANA Considerations

5.1. CoAP Signaling Option Number

The following entries are added to the "CoAP Signaling Option Numbers" registry within the "CoRE Parameters" registry.

| Applies to | Number | Name | Reference |
|------------|--------|------------------------|-------------------|
| 7.01 | TBD | Extended-Token-Lengths | [[this document]] |

6. References

6.1. Normative References

- [I-D.ietf-core-echo-request-tag]
Amsuess, C., Mattsson, J., and G. Selander, "Echo and Request-Tag", draft-ietf-core-echo-request-tag-03 (work in progress), October 2018.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<https://www.rfc-editor.org/info/rfc7252>>.

- [RFC7641] Hartke, K., "Observing Resources in the Constrained Application Protocol (CoAP)", RFC 7641, DOI 10.17487/RFC7641, September 2015, <<https://www.rfc-editor.org/info/rfc7641>>.
- [RFC7959] Bormann, C. and Z. Shelby, Ed., "Block-Wise Transfers in the Constrained Application Protocol (CoAP)", RFC 7959, DOI 10.17487/RFC7959, August 2016, <<https://www.rfc-editor.org/info/rfc7959>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8323] Bormann, C., Lemay, S., Tschofenig, H., Hartke, K., Silverajan, B., and B. Raymor, Ed., "CoAP (Constrained Application Protocol) over TCP, TLS, and WebSockets", RFC 8323, DOI 10.17487/RFC8323, February 2018, <<https://www.rfc-editor.org/info/rfc8323>>.

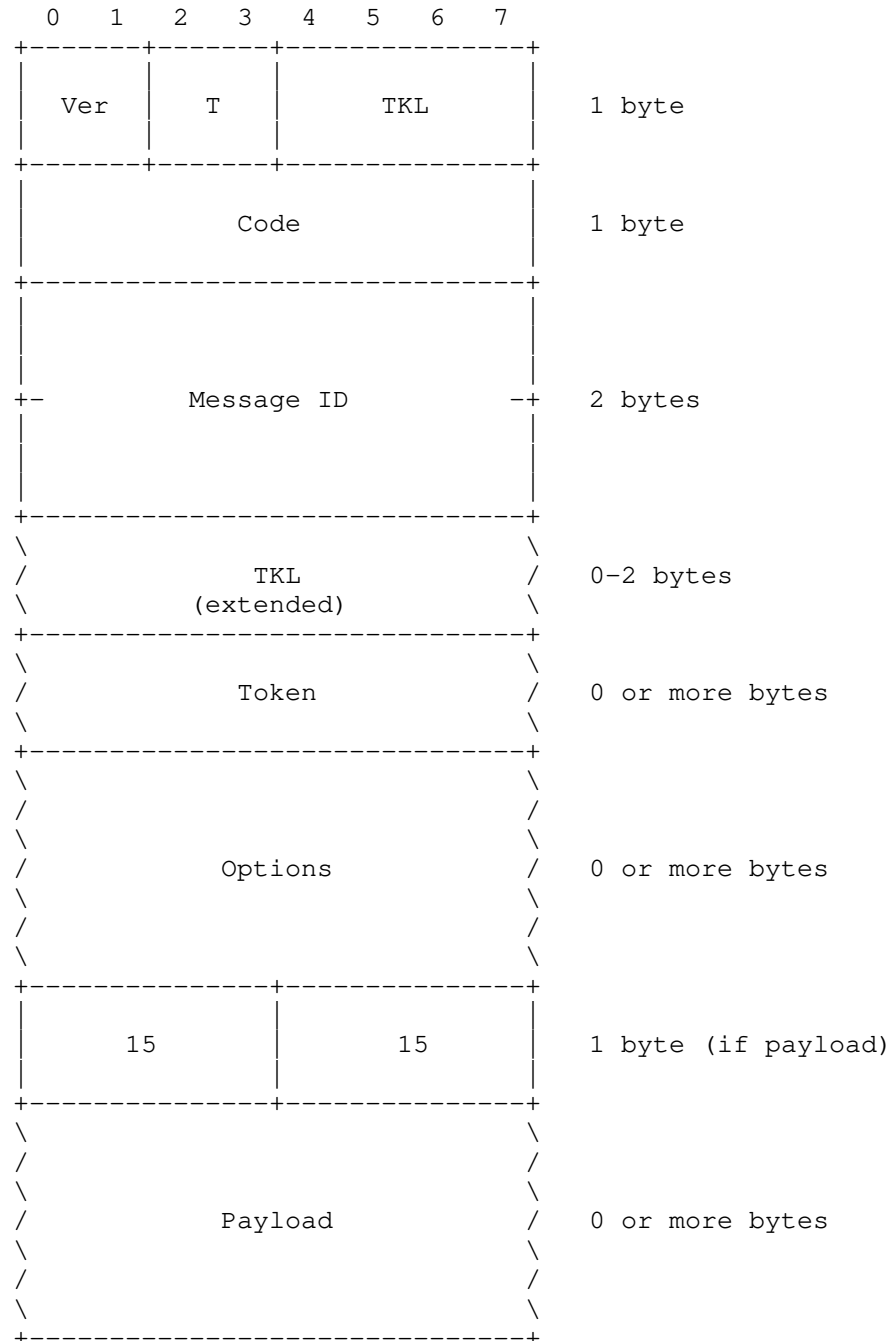
6.2. Informative References

- [I-D.ietf-6tisch-minimal-security] Vucinic, M., Simon, J., Pister, K., and M. Richardson, "Minimal Security Framework for 6TiSCH", draft-ietf-6tisch-minimal-security-09 (work in progress), November 2018.
- [RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, DOI 10.17487/RFC7228, May 2014, <<https://www.rfc-editor.org/info/rfc7228>>.

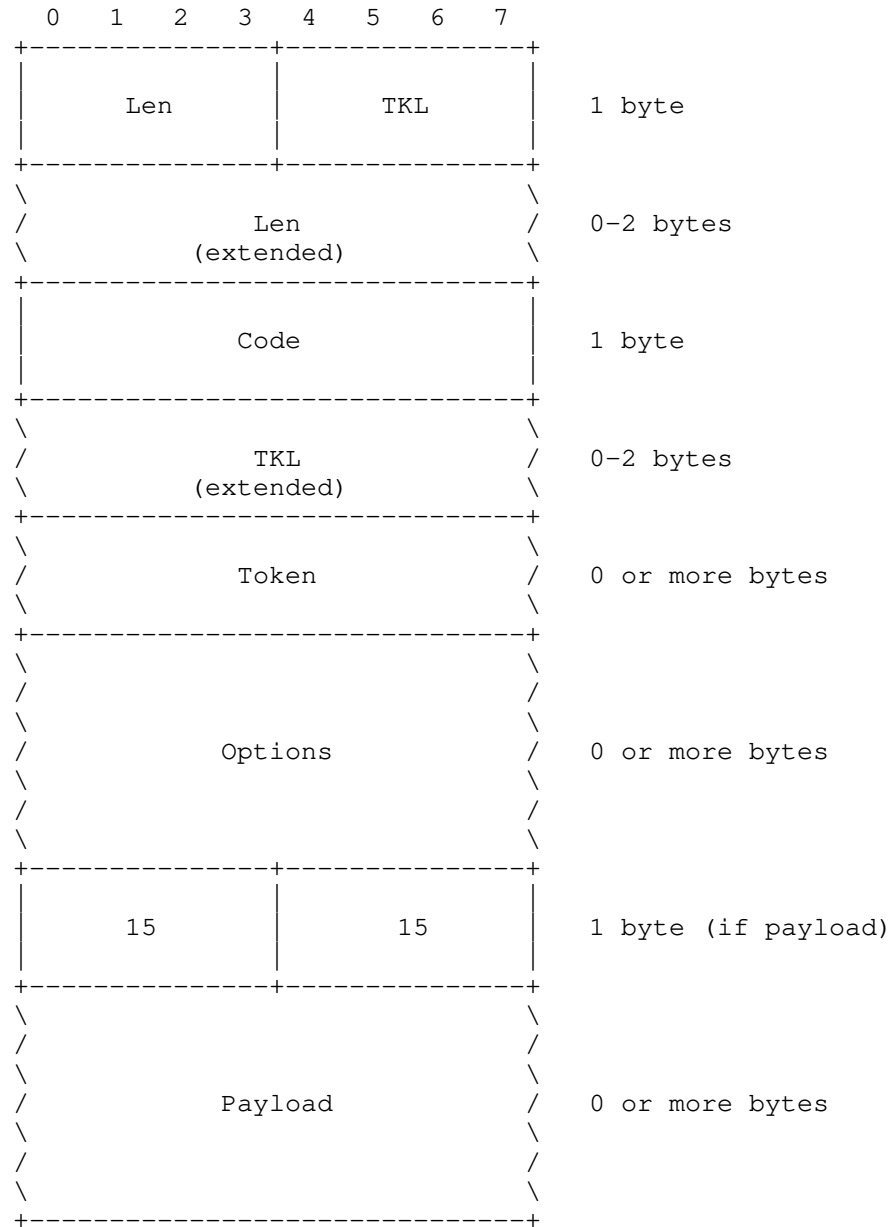
Appendix A. Updated Message Formats

This appendix illustrates the CoAP message formats updated with the new definition of the TKL field (Section 2).

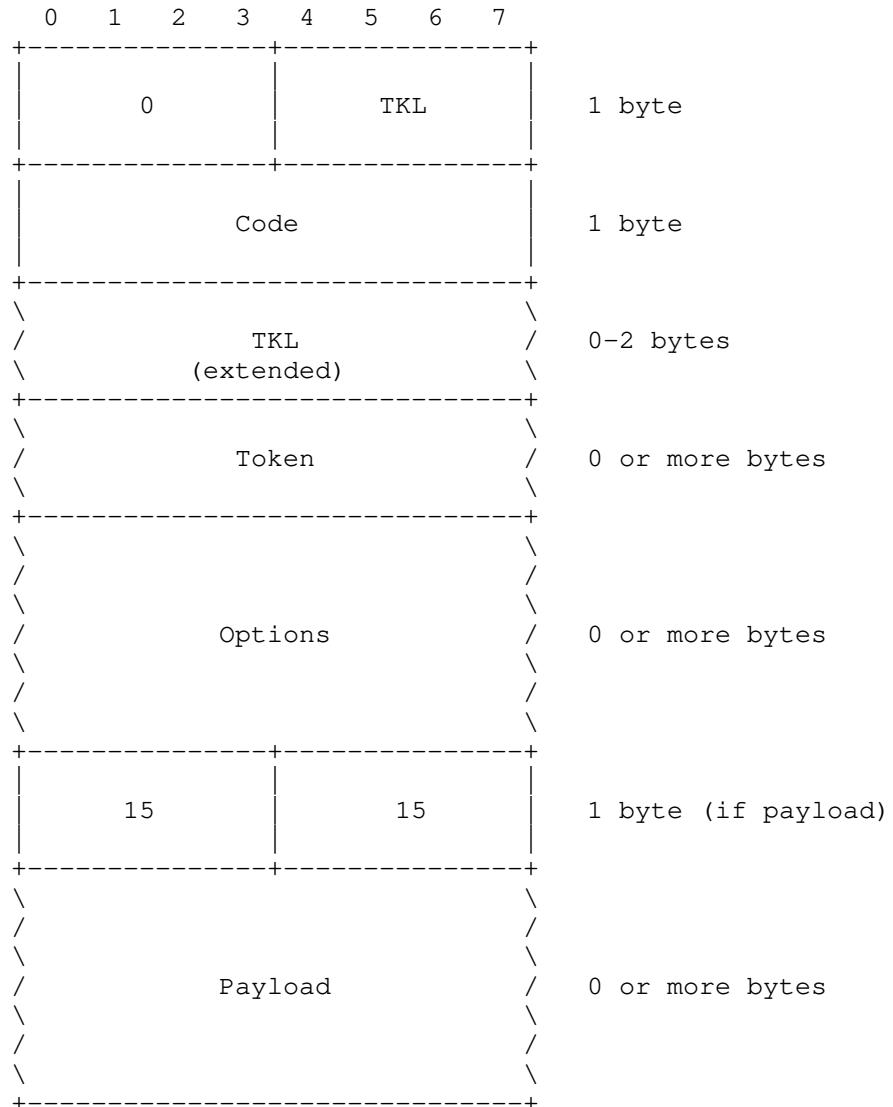
A.1. CoAP over UDP



A.2. CoAP over TCP



A.3. CoAP over WebSockets



Acknowledgements

This document is based on the requirements of and work on the Minimal Security Framework for 6TiSCH [I-D.ietf-6tisch-minimal-security] by Malisa Vucinic, Jonathan Simon, Kris Pister, and Michael Richardson.

Thanks to Carsten Bormann, Ari Keranen, John Mattsson, Jim Schaad, Goeran Selander, and Malisa Vucinic for helpful comments and discussions that have shaped the document.

Author's Address

Klaus Hartke
Ericsson
Torshamnsgatan 23
Stockholm SE-16483
Sweden

Email: klaus.hartke@ericsson.com

Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: October 10, 2019

M. Veillette, Ed.
Trilliant Networks Inc.
I. Petrov, Ed.
A. Pelov
Acklio
April 08, 2019

CBOR Encoding of Data Modeled with YANG
draft-ietf-core-yang-cbor-10

Abstract

This document defines encoding rules for serializing configuration data, state data, RPC input and RPC output, Action input, Action output and notifications defined within YANG modules using the Concise Binary Object Representation (CBOR) [RFC7049].

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on October 10, 2019.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of

the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

| | |
|--|----|
| 1. Introduction | 3 |
| 2. Terminology and Notation | 3 |
| 3. Properties of the CBOR Encoding | 4 |
| 3.1. CBOR diagnostic notation | 5 |
| 3.2. YANG Schema Item iDentifier (SID) | 6 |
| 3.3. Name | 7 |
| 4. Encoding of YANG Schema Node Instances | 9 |
| 4.1. The 'leaf' | 9 |
| 4.2. The 'container' and other collections | 9 |
| 4.2.1. SIDs as keys | 10 |
| 4.2.2. Names as keys | 11 |
| 4.3. The 'leaf-list' | 13 |
| 4.4. The 'list' and 'list' instance(s) | 14 |
| 4.4.1. SIDs as keys | 15 |
| 4.4.2. Names as keys | 17 |
| 4.5. The 'anydata' | 19 |
| 4.6. The 'anyxml' | 21 |
| 5. Encoding of YANG data templates | 22 |
| 5.1. SIDs as keys | 23 |
| 5.2. Names as keys | 24 |
| 6. Representing YANG Data Types in CBOR | 25 |
| 6.1. The unsigned integer Types | 25 |
| 6.2. The integer Types | 26 |
| 6.3. The 'decimal64' Type | 26 |
| 6.4. The 'string' Type | 27 |
| 6.5. The 'boolean' Type | 27 |
| 6.6. The 'enumeration' Type | 27 |
| 6.7. The 'bits' Type | 28 |
| 6.8. The 'binary' Type | 30 |
| 6.9. The 'leafref' Type | 30 |
| 6.10. The 'identityref' Type | 31 |
| 6.10.1. SIDs as identityref | 31 |
| 6.10.2. Name as identityref | 32 |
| 6.11. The 'empty' Type | 32 |
| 6.12. The 'union' Type | 33 |
| 6.13. The 'instance-identifier' Type | 34 |
| 6.13.1. SIDs as instance-identifier | 34 |
| 6.13.2. Names as instance-identifier | 37 |
| 7. Security Considerations | 39 |
| 8. IANA Considerations | 39 |
| 8.1. Tags Registry | 39 |
| 9. Acknowledgments | 40 |
| 10. References | 40 |

| | |
|--|----|
| 10.1. Normative References | 40 |
| 10.2. Informative References | 40 |
| Authors' Addresses | 41 |

1. Introduction

The specification of the YANG 1.1 data modelling language [RFC7950] defines an XML encoding for data instances, i.e. contents of configuration datastores, state data, RPC inputs and outputs, action inputs and outputs, and event notifications.

A new set of encoding rules has been defined to allow the use of the same data models in environments based on the JavaScript Object Notation (JSON) Data Interchange Format [RFC7159]. This is accomplished in the JSON Encoding of Data Modeled with YANG specification [RFC7951].

The aim of this document is to define a set of encoding rules for the Concise Binary Object Representation (CBOR) [RFC7049]. The resulting encoding is more compact compared to XML and JSON and more suitable for Constrained Nodes and/or Constrained Networks as defined by [RFC7228].

2. Terminology and Notation

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

The following terms are defined in [RFC7950]:

- o action
- o anydata
- o anyxml
- o data node
- o data tree
- o datastore
- o feature
- o identity
- o module

- o notification
- o RPC
- o schema node
- o schema tree
- o submodule

The following terms are defined in [RFC8040]:

- o yang-data (YANG extension)
- o YANG data template

This specification also makes use of the following terminology:

- o child: A schema node defined within a collection such as a container, a list, a case, a notification, an RPC input, an RPC output, an action input, an action output.
- o delta: Difference between the current SID and a reference SID. A reference SID is defined for each context for which deltas are used.
- o item: A schema node, an identity, a module, a submodule or a feature defined using the YANG modeling language.
- o parent: The collection in which a schema node is defined.
- o YANG Schema Item iDentifier (SID): Unsigned integer used to identify different YANG items.

3. Properties of the CBOR Encoding

This document defines CBOR encoding rules for YANG schema trees and their subtrees.

A collection such as container, list instance, notification, RPC input, RPC output, action input and action output is serialized using a CBOR map in which each child schema node is encoded using a key and a value. This specification supports two type of CBOR keys; YANG Schema Item iDentifier (SID) as defined in Section 3.2 and names as defined in Section 3.3. Each of these key types is encoded using a specific CBOR type which allows their interpretation during the deserialization process. Protocols or mechanisms implementing this specification can mandate the use of a specific key type.

In order to minimize the size of the encoded data, the proposed mapping avoids any unnecessary meta-information beyond those natively supported by CBOR. For instance, CBOR tags are used solely in the case of anyxml schema nodes and the union datatype to distinguish explicitly the use of different YANG datatypes encoded using the same CBOR major type.

Unless specified otherwise by the protocol or mechanism implementing this specification, the infinite lengths encoding as defined in [RFC7049] section 2.2 SHALL be supported by CBOR decoders.

Data nodes implemented using a CBOR array, map, byte string, and text string can be instantiated but empty. In this case, they are encoded with a length of zero.

Application payloads carrying a value serialized using the rules defined by this specification (e.g. CoAP Content-Format) SHOULD include the identifier (e.g. SID, namespace qualified name, instance-identifier) of this value. When SIDs are used as identifiers, the reference SID SHALL be included in the payload to allow stateless conversion of delta values to SIDs. Formats of these application payloads are not defined by the current specification.

3.1. CBOR diagnostic notation

Within this document, CBOR binary contents are represented using an equivalent textual form called CBOR diagnostic notation as defined in [RFC7049] section 6. This notation is used strictly for documentation purposes and is never used in the data serialization. Table 1 below provides a summary of this notation.

| CBOR content | CBOR type | Diagnostic notation | Example | CBOR encoding |
|------------------|--------------|---|--------------------|--------------------|
| Unsigned integer | 0 | Decimal digits | 123 | 18 7B |
| Negative integer | 1 | Decimal digits prefixed by a minus sign | -123 | 38 7A |
| Byte string | 2 | Hexadecimal value enclosed between single quotes and prefixed by an 'h' | h'F15C' | 42 f15C |
| Text string | 3 | String of Unicode characters enclosed between double quotes | "txt" | 63 747874 |
| Array | 4 | Comma-separated list of values within square brackets | [1, 2] | 82 01 02 |
| Map | 5 | Comma-separated list of key : value pairs within curly braces | { 1: 123, 2: 456 } | a2 01187B 021901C8 |
| Boolean | 7/20 7/21 | false true | false true | F4 F5 |
| Null | 7/22 | null | null | F6 |
| Not assigned | 7/23 | undefined | undefined | F7 |

Table 1: CBOR diagnostic notation summary

The following extensions to the CBOR diagnostic notation are supported:

- o Any text within and including a pair of slashes is considered a comment.
- o Deltas are visualized as numbers preceded by a '+' or '-' sign. The use of the '+' sign for positive deltas represents an extension to the CBOR diagnostic notation as defined by [RFC7049] section 6.

3.2. YANG Schema Item iDentifier (SID)

Some of the items defined in YANG [RFC7950] require the use of a unique identifier. In both NETCONF [RFC6241] and RESTCONF [RFC8040], these identifiers are implemented using strings. To allow the implementation of data models defined in YANG in constrained devices and constrained networks, a more compact method to identify YANG

items is required. This compact identifier, called YANG Schema Item Identifier (SID), is an unsigned integer. The following items are identified using SIDs:

- o identities
- o data nodes
- o RPCs and associated input(s) and output(s)
- o actions and associated input(s) and output(s)
- o notifications and associated information
- o YANG modules, submodules and features

To minimize its size, in certain positions, SIDs are represented using a (signed) delta from a reference SID and the current SID. Conversion from SIDs to deltas and back to SIDs are stateless processes solely based on the data serialized or deserialized.

Mechanisms and processes used to assign SIDs to YANG items and to guarantee their uniqueness is outside the scope of the present specification. If SIDs are to be used, the present specification is used in conjunction with a specification defining this management. One example for such a specification is under development as [I-D.ietf-core-sid].

3.3. Name

This specification also supports the encoding of YANG item identifiers as string, similar as those used by the JSON Encoding of Data Modeled with YANG [RFC7951]. This approach can be used to avoid the management overhead associated to SIDs allocation. The main drawback is the significant increase in size of the encoded data.

YANG items identifiers implemented using names MUST be in one of the following forms:

- o simple - the identifier of the YANG item (i.e. schema node or identity).
- o namespace qualified - the identifier of the YANG item is prefixed with the name of the module in which this item is defined, separated by the colon character (":").

The name of a module determines the namespace of all YANG items defined in that module. If an item is defined in a submodule, then

the namespace qualified name uses the name of the main module to which the submodule belongs.

ABNF syntax [RFC5234] of a name is shown in Figure 1, where the production for "identifier" is defined in Section 14 of [RFC7950].

```
name = [identifier ":" ] identifier
```

Figure 1: ABNF Production for a simple or namespace qualified name

A namespace qualified name MUST be used for all members of a top-level CBOR map and then also whenever the namespaces of the data node and its parent node are different. In all other cases, the simple form of the name SHOULD be used.

Definition example:

```
module example-foomod {
  container top {
    leaf foo {
      type uint8;
    }
  }
}

module example-barmod {
  import example-foomod {
    prefix "foomod";
  }
  augment "/foomod:top" {
    leaf bar {
      type boolean;
    }
  }
}
```

A valid CBOR encoding of the 'top' container is as follow.

CBOR diagnostic notation:

```
{
  "example-foomod:top": {
    "foo": 54,
    "example-barmod:bar": true
  }
}
```


Both the 'top' container and the 'bar' leaf defined in a different YANG module as its parent container are encoded as namespace qualified names. The 'foo' leaf defined in the same YANG module as its parent container is encoded as simple name.

4. Encoding of YANG Schema Node Instances

Schema node instances defined using the YANG modeling language are encoded using CBOR [RFC7049] based on the rules defined in this section. We assume that the reader is already familiar with both YANG [RFC7950] and CBOR [RFC7049].

4.1. The 'leaf'

A 'leaf' MUST be encoded accordingly to its datatype using one of the encoding rules specified in Section 6.

4.2. The 'container' and other collections

Collections such as containers, list instances, notification contents, rpc inputs, rpc outputs, action inputs and action outputs MUST be encoded using a CBOR map data item (major type 5). A map is comprised of pairs of data items, with each data item consisting of a key and a value. Each key within the CBOR map is set to a schema node identifier, each value is set to the value of this schema node instance according to the instance datatype.

This specification supports two type of CBOR keys; SID as defined in Section 3.2 and names as defined in Section 3.3.

The following examples shows the encoding of a 'system-state' container instance using SIDs or names.

Definition example from [RFC7317]:

```
typedef date-and-time {
  type string {
    pattern '\d{4}-\d{2}-\d{2}T\d{2}:\d{2}:\d{2}(\.\d+)?(Z|[\+\-]
      \d{2}:\d{2})';
  }
}

container system-state {

  container clock {
    leaf current-datetime {
      type date-and-time;
    }

    leaf boot-datetime {
      type date-and-time;
    }
  }
}
```

4.2.1. SIDs as keys

CBOR map keys implemented using SIDs MUST be encoded using a CBOR unsigned integer (major type 0) or CBOR negative integer (major type 1), depending on the actual delta or to a SID preceded by the CBOR tag 42.

Delta values are computed as follows:

- o In the case of a 'container', deltas are equal to the SID of the current schema node minus the SID of the parent 'container'.
- o In the case of a 'list', deltas are equal to the SID of the current schema node minus the SID of the parent 'list'.
- o In the case of an 'rpc input' or 'rpc output', deltas are equal to the SID of the current schema node minus the SID of the 'rpc'.
- o In the case of an 'action input' or 'action output', deltas are equal to the SID of the current schema node minus the SID of the 'action'.
- o In the case of an 'notification content', deltas are equal to the SID of the current schema node minus the SID of the 'notification'.

This example assumes that the Media Type used to carry this container consists of a CBOR map composed of the data node SID and data node

encoding. This root CBOR map is not part of the present encoding rules and is not compulsory.

CBOR diagnostic notation:

```
{
  1720 : {
    +1 : {
      +2 : "2015-10-02T14:47:24Z-05:00", / current-datetime (SID 1723) /
      +1 : "2015-09-15T09:12:58Z-05:00" / boot-datetime (SID 1722) /
    }
  }
}
```

CBOR encoding:

```
A1                                # map(1)
  19 06B8                         # unsigned(1720)
  A1                              # map(1)
    01                           # unsigned(1)
    A2                           # map(2)
      02                         # unsigned(2)
      78 1A                     # text(26)
        323031352D31302D30325431343A34373A32345A2D30353A3030
      01                         # unsigned(1)
      78 1A                     # text(26)
        323031352D30392D31355430393A31323A35385A2D30353A3030
```

4.2.2. Names as keys

CBOR map keys implemented using names MUST be encoded using a CBOR text string data item (major type 3). A namespace-qualified name MUST be used each time the namespace of a schema node and its parent differ. In all other cases, the simple form of the name MUST be used. Names and namespaces are defined in [RFC7951] section 4.

The following example shows the encoding of a 'system' container instance using names.

Definition example from [RFC7317]:

```
typedef date-and-time {
  type string {
    pattern '\d{4}-\d{2}-\d{2}T\d{2}:\d{2}:\d{2}(\.\d+)?(Z|[\+\-]
      \d{2}:\d{2})';
  }
}

container system-state {

  container clock {
    leaf current-datetime {
      type date-and-time;
    }

    leaf boot-datetime {
      type date-and-time;
    }
  }
}
```

This example assumes that the Media Type used to carry this container consists of a CBOR map composed of the data node namespace qualified name and data node encoding. This root CBOR map is not part of the present encoding rules and is not compulsory.

CBOR diagnostic notation:

```
{
  "ietf-system:system-state" : {
    "ietf-system:clock" : {
      "current-datetime" : "2015-10-02T14:47:24Z-05:00",
      "boot-datetime" : "2015-09-15T09:12:58Z-05:00"
    }
  }
}
```

CBOR encoding:

```

A1                                     # map(1)
  78 18                               # text(24)
    696574662D73797374656D3A73797374656D2D7374617465
A1                                     # map(1)
  71                                   # text(17)
    696574662D73797374656D3A636C6F636B
A2                                     # map(2)
  70                                   # text(16)
    63757272656E742D6461746574696D65
  78 1A                               # text(26)
    323031352D31302D30325431343A34373A32345A2D30353A3030
  6D                                   # text(13)
    626F6F742D6461746574696D65
  78 1A                               # text(26)
    323031352D30392D31355430393A31323A35385A2D30353A3030

```

4.3. The 'leaf-list'

A leaf-list MUST be encoded using a CBOR array data item (major type 4). Each entry of this array MUST be encoded accordingly to its datatype using one of the encoding rules specified in Section 6.

The following example shows the encoding of the 'search' leaf-list instance containing two entries, "ietf.org" and "ieee.org".

Definition example [RFC7317]:

```

typedef domain-name {
  type string {
    length "1..253";
    pattern '((([a-zA-Z0-9_]([a-zA-Z0-9\_-]){0,61})?[a-zA-Z0-9].)
              *([a-zA-Z0-9_]([a-zA-Z0-9\_-]){0,61})?[a-zA-Z0-9]\.?
              )|\.';
  }
}

leaf-list search {
  type domain-name;
  ordered-by user;
}

```

CBOR diagnostic notation: ["ietf.org", "ieee.org"]

CBOR encoding: 82 68 696574662E6F7267 68 696565652E6F7267

4.4. The 'list' and 'list' instance(s)

A list or a subset of a list MUST be encoded using a CBOR array data item (major type 4). Each list instance within this CBOR array is encoded using a CBOR map data item (major type 5) based on the encoding rules of a collection as defined in Section 4.2.

It is important to note that this encoding rule also apply to a single 'list' instance.

The following examples show the encoding of a 'server' list using SIDs or names.

Definition example from [RFC7317]:

```
list server {
  key name;

  leaf name {
    type string;
  }
  choice transport {
    case udp {
      container udp {
        leaf address {
          type host;
          mandatory true;
        }
        leaf port {
          type port-number;
        }
      }
    }
  }
  leaf association-type {
    type enumeration {
      enum server;
      enum peer;
      enum pool;
    }
    default server;
  }
  leaf iburst {
    type boolean;
    default false;
  }
  leaf prefer {
    type boolean;
    default false;
  }
}
```

4.4.1. SIDs as keys

The encoding rules of each 'list' instance are defined in Section 4.2.1. Deltas of list members are equal to the SID of the current schema node minus the SID of the 'list'.

This example assumes that the Media Type used to carry this list consists of a CBOR map composed of the data node SID and data node encoding. This root CBOR map is not part of the present encoding rules and is not compulsory.

CBOR diagnostic notation:

```
{
  1756 : [
    {
      +3 : "NRC TIC server",    / name (SID 1759) /
      +5 : {
        +1 : "tic.nrc.ca",    / address (SID 1762) /
        +2 : 123              / port (SID 1763) /
      },
      +1 : 0,                  / association-type (SID 1757) /
      +2 : false,              / iburst (SID 1758) /
      +4 : true                / prefer (SID 1760) /
    },
    {
      +3 : "NRC TAC server",    / name (SID 1759) /
      +5 : {
        +1 : "tac.nrc.ca"      / address (SID 1762) /
      }
    }
  ]
}
```

CBOR encoding:


```

A1                                # map(1)
  19 06DC                        # unsigned(1756)
  82                              # array(2)
    A5                          # map(5)
      03                        # unsigned(3)
      6E                        # text(14)
        4E52432054494320736572766572 # "NRC TIC server"
      05                        # unsigned(5)
      A2                        # map(2)
        01                    # unsigned(1)
        6A                    # text(10)
          74696332E6E72632E6361      # "tic.nrc.ca"
        02                    # unsigned(2)
        18 7B                  # unsigned(123)
      01                    # unsigned(1)
      00                    # unsigned(0)
      02                    # unsigned(2)
      F4                    # primitive(20)
      04                    # unsigned(4)
      F5                    # primitive(21)
    A2                        # map(2)
      03                    # unsigned(3)
      6E                    # text(14)
        4E52432054414320736572766572 # "NRC TAC server"
      05                    # unsigned(5)
      A1                    # map(1)
        01                    # unsigned(1)
        6A                    # text(10)
          74616332E6E72632E6361      # "tac.nrc.ca"

```

4.4.2. Names as keys

The encoding rules of each 'list' instance are defined in Section 4.2.2.

This example assumes that the Media Type used to carry this container consists of a CBOR map composed of the data node namespace qualified name and data node encoding. This root CBOR map is not part of the present encoding rules and is not compulsory.

CBOR diagnostic notation:

```
{
  "ietf-system:server" : [
    {
      "name" : "NRC TIC server",
      "udp" : {
        "address" : "tic.nrc.ca",
        "port" : 123
      },
      "association-type" : 0,
      "iburst" : false,
      "prefer" : true
    },
    {
      "name" : "NRC TAC server",
      "udp" : {
        "address" : "tac.nrc.ca"
      }
    }
  ]
}
```

CBOR encoding:

```

A1                                     # map(1)
  72                                 # text(18)
    696574662D73797374656D3A736572766572
  82                                 # array(2)
    A5                             # map(5)
      64                           # text(4)
        6E616D65                  # "name"
      6E                           # text(14)
        4E52432054494320736572766572
      63                           # text(3)
        756470                    # "udp"
      A2                           # map(2)
        67                        # text(7)
          61646472657373          # "address"
        6A                        # text(10)
          7469632E6E72632E6361    # "tic.nrc.ca"
        64                        # text(4)
          706F7274                # "port"
        18 7B                    # unsigned(123)
      70                          # text(16)
        6173736F63696174696F6E2D74797065
      00                          # unsigned(0)
      66                          # text(6)
        696275727374             # "iburst"
      F4                          # primitive(20)
      66                          # text(6)
        707265666572             # "prefer"
      F5                          # primitive(21)
    A2                            # map(2)
      64                          # text(4)
        6E616D65                  # "name"
      6E                          # text(14)
        4E52432054414320736572766572
      63                          # text(3)
        756470                    # "udp"
      A1                          # map(1)
        67                        # text(7)
          61646472657373          # "address"
        6A                        # text(10)
          7461632E6E72632E6361    # "tac.nrc.ca"

```

4.5. The 'anydata'

An anydata serves as a container for an arbitrary set of schema nodes that otherwise appear as normal YANG-modeled data. An anydata instance is encoded using the same rules as a container, i.e., CBOR map. The requirement that anydata content can be modeled by YANG implies the following:

- o CBOR map keys of any inner schema nodes MUST be set to valid deltas or names.
- o The CBOR array MUST contain either unique scalar values (as a leaf-list, see Section 4.3), or maps (as a list, see Section 4.4).
- o CBOR map values MUST follow the encoding rules of one of the datatypes listed in Section 4.

The following example shows a possible use of an anydata. In this example, an anydata is used to define a schema node containing a notification event, this schema node can be part of a YANG list to create an event logger.

Definition example:

```
module event-log {  
    ...  
    anydata last-event;          # SID 60123
```

This example also assumes the assistance of the following notification.

```
module example-port {  
    ...  
  
    notification example-port-fault { # SID 60200  
        leaf port-name { # SID 60201  
            type string;  
        }  
        leaf port-fault { # SID 60202  
            type string;  
        }  
    }  
}
```

This example assumes that the Media Type used to carry this anydata consists of a CBOR map composed of the data node SID and data node encoding. This root CBOR map is not part of the present encoding rules and is not compulsory.

CBOR diagnostic notation:

```

{
  60123 : {
    +77 : {
      +1 : "0/4/21",
      +2 : "Open pin 2"
    }
  }
}

```

CBOR encoding:

```

A1                                # map(1)
  19 EADB                        # unsigned(60123)
    A1                          # map(1)
      18 4D                      # unsigned(77)
        A2                      # map(2)
          18 4E                  # unsigned(78)
            66                  # text(6)
              302F342F3231      # "0/4/21"
          18 4F                  # unsigned(79)
            6A                  # text(10)
              4F70656E2070696E2032 # "Open pin 2"

```

In some implementations, it might be simpler to use the absolute SID tag encoding for the anydata root element. The resulting encoding is as follow:

```

{
  60123 : {
    42(60200) : {
      +1 : "0/4/21",
      +2 : "Open pin 2"
    }
  }
}

```

4.6. The 'anyxml'

An anyxml schema node is used to serialize an arbitrary CBOR content, i.e., its value can be any CBOR binary object. anyxml value MAY contain CBOR data items tagged with one of the tag listed in Section 8.1, these tags shall be supported.

The following example shows a valid CBOR encoded instance consisting of a CBOR array containing the CBOR simple values 'true', 'null' and 'true'.

Definition example from [RFC7951]:

```
anyxml bar;
```

Note: This example assumes that the Media Type used to carry this anyxml consists of a CBOR map composed of the data node SID and data node encoding. This root CBOR map is not part of the present encoding rules and is not compulsory.

CBOR diagnostic notation:

```
{
  60000 : [true, null, true]   / bar (SID 60000) /
}
```

CBOR encoding:

```
A1          # map(1)
  19 EA60    # unsigned(60000)
  83         # array(3)
    F5       # primitive(21)
    F6       # primitive(22)
    F5       # primitive(21)
```

5. Encoding of YANG data templates

YANG data templates are data structures defined in YANG but not intended to be implemented as part of a datastore. YANG data templates are defined using the 'yang-data' extension as described by RFC 8040.

YANG data templates SHOULD be encoded using the encoding rules of a collection as defined in Section 4.2.

Just like YANG containers, YANG data templates can be encoded using either SIDs or names.

Definition example from [I-D.ietf-core-comi]:

```
import ietf-restconf {
  prefix rc;
}

rc:yang-data yang-errors {
  container error {
    leaf error-tag {
      type identityref {
        base error-tag;
      }
    }
    leaf error-app-tag {
      type identityref {
        base error-app-tag;
      }
    }
    leaf error-data-node {
      type instance-identifier;
    }
    leaf error-message {
      type string;
    }
  }
}
```

5.1. SIDs as keys

YANG template encoded using SIDs are carried in a CBOR map containing a single item pair. The key of this item is set to the SID assigned to the YANG template container, the value is set the CBOR encoding of this container as defined in Section 4.2.

This example shows a serialization example of the yang-errors template as defined in [I-D.ietf-core-comi] using SIDs as defined in Section 3.2.

CBOR diagnostic notation:

```

{
  1024 : {
    +4 : 1011,
    +1 : 1018,
    +2 : 1740,
    +3 : "Maximum exceeded"
  }
}

```

CBOR encoding:

```

A1                                # map(1)
  19 0400                        # unsigned(1024)
  A4                             # map(4)
    04                          # unsigned(4)
    19 03F3                     # unsigned(1011)
    01                          # unsigned(1)
    19 03FA                     # unsigned(1018)
    02                          # unsigned(2)
    19 06CC                     # unsigned(1740)
    03                          # unsigned(3)
    70                          # text(16)
                                4D6178696D756D2065786365565646564

```

5.2. Names as keys

YANG template encoded using names are carried in a CBOR map containing a single item pair. The key of this item is set to the namespace qualified name of the YANG template container, the value is set the CBOR encoding of this container as defined in Section 3.3.

This example shows a serialization example of the yang-errors template as defined in [I-D.ietf-core-comi] using names as defined Section 3.3.

CBOR diagnostic notation:

```

{
  "ietf-comi:error" : {
    "error-tag" : "invalid-value",
    "error-app-tag" : "not-in-range",
    "error-data-node" : "timezone-utc-offset",
    "error-message" : "Maximum exceeded"
  }
}

```


CBOR encoding:

```

A1                                     # map(1)
  6F                                 # text(15)
    696574662D636F6D693A6572726F72 # "ietf-comi:error"
  A4                                 # map(4)
    69                             # text(9)
      6572726F722D746167          # "error-tag"
    6D                             # text(13)
      696E76616C69642D76616C7565 # "invalid-value"
    6D                             # text(13)
      6572726F722D6170702D746167 # "error-app-tag"
    6C                             # text(12)
      6E6F742D696E2D72616E6765   # "not-in-range"
    6F                             # text(15)
      6572726F722D646174612D6E6F6465 # "error-data-node"
    73                             # text(19)
      74696D657A6F6E652D7574632D6F6666736574 # "timezone-utc-offset"
    6D                             # text(13)
      6572726F722D6D657373616765   # "error-message"
    70                             # text(16)
      4D6178696D756D20657863655646564

```

6. Representing YANG Data Types in CBOR

The CBOR encoding of an instance of a leaf or leaf-list schema node depends on the built-in type of that schema node. The following subsection defined the CBOR encoding of each built-in type supported by YANG as listed in [RFC7950] section 4.2.4. Each subsection shows an example value assigned to a schema node instance of the discussed built-in type.

6.1. The unsigned integer Types

Leafs of type uint8, uint16, uint32 and uint64 MUST be encoded using a CBOR unsigned integer data item (major type 0).

The following example shows the encoding of a 'mtu' leaf instance set to 1280 bytes.

Definition example from [RFC7277]:

```

leaf mtu {
  type uint16 {
    range "68..max";
  }
}

```

CBOR diagnostic notation: 1280

CBOR encoding: 19 0500

6.2. The integer Types

Leafs of type `int8`, `int16`, `int32` and `int64` MUST be encoded using either CBOR unsigned integer (major type 0) or CBOR negative integer (major type 1), depending on the actual value.

The following example shows the encoding of a 'timezone-utc-offset' leaf instance set to -300 minutes.

Definition example from [RFC7317]:

```
leaf timezone-utc-offset {  
  type int16 {  
    range "-1500 .. 1500";  
  }  
}
```

CBOR diagnostic notation: -300

CBOR encoding: 39 012B

6.3. The 'decimal64' Type

Leafs of type `decimal64` MUST be encoded using a decimal fraction as defined in [RFC7049] section 2.4.3.

The following example shows the encoding of a 'my-decimal' leaf instance set to 2.57.

Definition example from [RFC7317]:

```
leaf my-decimal {  
  type decimal64 {  
    fraction-digits 2;  
    range "1 .. 3.14 | 10 | 20..max";  
  }  
}
```

CBOR diagnostic notation: 4([-2, 257])

CBOR encoding: C4 82 21 19 0101

6.4. The 'string' Type

Leafs of type string MUST be encoded using a CBOR text string data item (major type 3).

The following example shows the encoding of a 'name' leaf instance set to "eth0".

Definition example from [RFC7223]:

```
leaf name {  
    type string;  
}
```

CBOR diagnostic notation: "eth0"

CBOR encoding: 64 65746830

6.5. The 'boolean' Type

Leafs of type boolean MUST be encoded using a CBOR simple value 'true' (major type 7, additional information 21) or 'false' (major type 7, additional information 20).

The following example shows the encoding of an 'enabled' leaf instance set to 'true'.

Definition example from [RFC7317]:

```
leaf enabled {  
    type boolean;  
}
```

CBOR diagnostic notation: true

CBOR encoding: F5

6.6. The 'enumeration' Type

Leafs of type enumeration MUST be encoded using a CBOR unsigned integer (major type 0) or CBOR negative integer (major type 1), depending on the actual value. Enumeration values are either explicitly assigned using the YANG statement 'value' or automatically assigned based on the algorithm defined in [RFC7950] section 9.6.4.2.

The following example shows the encoding of an 'oper-status' leaf instance set to 'testing'.

Definition example from [RFC7317]:

```
leaf oper-status {  
  type enumeration {  
    enum up { value 1; }  
    enum down { value 2; }  
    enum testing { value 3; }  
    enum unknown { value 4; }  
    enum dormant { value 5; }  
    enum not-present { value 6; }  
    enum lower-layer-down { value 7; }  
  }  
}
```

CBOR diagnostic notation: 3

CBOR encoding: 03

To avoid overlap of 'value' defined in different 'enumeration' statements, 'enumeration' defined in a Leafs of type 'union' MUST be encoded using a CBOR text string data item (major type 3) and MUST contain one of the names assigned by 'enum' statements in YANG. The encoding MUST be prefixed with the enumeration CBOR tag as specified in Section 8.1.

Definition example from [RFC7950]:

```
type union {  
  type int32;  
  type enumeration {  
    enum "unbounded";  
  }  
}
```

CBOR diagnostic notation: 44("unbounded")

CBOR encoding: D8 2C 69 756E626F756E646564

6.7. The 'bits' Type

Leafs of type bits MUST be encoded using a CBOR byte string data item (major type 2). Bits position are either explicitly assigned using the YANG statement 'position' or automatically assigned based on the algorithm defined in [RFC7950] section 9.7.4.2.

Bits position 0 to 7 are assigned to the first byte within the byte string, bits 8 to 15 to the second byte, and subsequent bytes are

assigned similarly. Within each byte, bits are assigned from least to most significant.

The following example shows the encoding of an 'alarm-state' leaf instance with the 'under-repair' and 'critical' flags set.

Definition example from [RFC8348]:

```
typedef alarm-state {
  type bits {
    bit unknown;
    bit under-repair;
    bit critical;
    bit major;
    bit minor;
    bit warning;
    bit indeterminate;
  }
}

leaf alarm-state {
  type alarm-state;
}
```

CBOR diagnostic notation: h'06'

CBOR encoding: 41 06

To avoid overlap of 'bit' defined in different 'bits' statements, 'bits' defined in a Leafs of type 'union' MUST be encoded using a CBOR text string data item (major type 3) and MUST contain a space-separated sequence of names of 'bit' that are set. The encoding MUST be prefixed with the bits CBOR tag as specified in Section 8.1.

The following example shows the encoding of an 'alarm-state' leaf instance defined using a union type with the 'under-repair' and 'critical' flags set.

Definition example:

```
leaf alarm-state-2 {
  type union {
    type alarm-state;
    type bits {
      bit extra-flag;
    }
  }
}
```

CBOR diagnostic notation: 43("under-repair critical")

CBOR encoding: D8 2B 75 756E6465722D72657061697220637269746963616C

6.8. The 'binary' Type

Leafs of type binary MUST be encoded using a CBOR byte string data item (major type 2).

The following example shows the encoding of an 'aes128-key' leaf instance set to 0x1f1ce6a3f42660d888d92a4d8030476e.

Definition example:

```
leaf aes128-key {  
  type binary {  
    length 16;  
  }  
}
```

CBOR diagnostic notation: h'1F1CE6A3F42660D888D92A4D8030476E'

CBOR encoding: 50 1F1CE6A3F42660D888D92A4D8030476E

6.9. The 'leafref' Type

Leafs of type leafref MUST be encoded using the rules of the schema node referenced by the 'path' YANG statement.

The following example shows the encoding of an 'interface-state-ref' leaf instance set to "eth1".

Definition example from [RFC7223]:

```
typedef interface-state-ref {  
  type leafref {  
    path "/interfaces-state/interface/name";  
  }  
}  
  
container interfaces-state {  
  list interface {  
    key "name";  
    leaf name {  
      type string;  
    }  
    leaf-list higher-layer-if {  
      type interface-state-ref;  
    }  
  }  
}
```

CBOR diagnostic notation: "eth1"

CBOR encoding: 64 65746831

6.10. The 'identityref' Type

This specification supports two approaches for encoding `identityref`, a YANG Schema Item identifier (SID) as defined in Section 3.2 or a name as defined in [RFC7951] section 6.8.

6.10.1. SIDs as `identityref`

When schema nodes of type `identityref` are implemented using SIDs, they MUST be encoded using a CBOR unsigned integer data item (major type 0). (Note that no delta mechanism is employed for SIDs as `identityref`.)

The following example shows the encoding of a 'type' leaf instance set to the value 'iana-if-type:ethernetCsmacd' (SID 1880).

Definition example from [RFC7317]:

```
identity interface-type {  
}  
  
identity iana-interface-type {  
    base interface-type;  
}  
  
identity ethernetCsmacd {  
    base iana-interface-type;  
}  
  
leaf type {  
    type identityref {  
        base interface-type;  
    }  
}
```

CBOR diagnostic notation: 1880

CBOR encoding: 19 0758

6.10.2. Name as identityref

Alternatively, an identityref MAY be encoded using a name as defined in Section 3.3. When names are used, identityref MUST be encoded using a CBOR text string data item (major type 3). If the identity is defined in different module than the leaf node containing the identityref data node, the namespace qualified form MUST be used. Otherwise, both the simple and namespace qualified forms are permitted. Names and namespaces are defined in Section 3.3.

The following example shows the encoding of the identity 'iana-if-type:ethernetCsmacd' using its namespace qualified name. This example is described in Section 6.10.1.

CBOR diagnostic notation: "iana-if-type:ethernetCsmacd"

CBOR encoding: 78 1b
69616E612D696662D747970653A65746865726E657443736D616364

6.11. The 'empty' Type

Leafs of type empty MUST be encoded using the CBOR null value (major type 7, additional information 22).

The following example shows the encoding of a 'is-router' leaf instance when present.

Definition example from [RFC7277]:

```
leaf is-router {  
  type empty;  
}
```

CBOR diagnostic notation: null

CBOR encoding: F6

6.12. The 'union' Type

Leafs of type union MUST be encoded using the rules associated with one of the types listed. When used in a union, the following YANG datatypes are prefixed by CBOR tag to avoid confusion between different YANG datatypes encoded using the same CBOR major type.

- o bits
- o enumeration
- o identityref
- o instance-identifier

See Section 8.1 for the assigned value of these CBOR tags.

As mentioned in Section 6.6 and in Section 6.7, 'enumeration' and 'bits' are encoded as CBOR text string data item (major type 3) when defined within a 'union' type.

The following example shows the encoding of an 'ip-address' leaf instance when set to "2001:db8:a0b:12f0::1".

Definition example from [RFC7317]:

In the case of a schema node member of a YANG list, a SID is combined with the list key(s) to identify each instance within the YANG list(s).

Single instance schema nodes MUST be encoded using a CBOR unsigned integer data item (major type 0) and set to the targeted schema node SID.

Schema nodes member of a YANG list MUST be encoded using a CBOR array data item (major type 4) containing the following entries:

- o The first entry MUST be encoded as a CBOR unsigned integer data item (major type 0) and set to the targeted schema node SID.
- o The following entries MUST contain the value of each key required to identify the instance of the targeted schema node. These keys MUST be ordered as defined in the 'key' YANG statement, starting from top level list, and follow by each of the subordinate list(s).

Examples within this section assume the definition of a schema node of type 'instance-identifier':

Definition example from [RFC7950]:

```
container system {  
  ...  
  leaf reporting-entity {  
    type instance-identifier;  
  }  
  
  leaf contact { type string; }  
  
  leaf hostname { type inet:domain-name; } } ~~~~
```

First example:

The following example shows the encoding of the 'reporting-entity' value referencing data node instance "/system/contact" (SID 1741).

Definition example from [RFC7317]:

```
container system {  
    leaf contact {  
        type string;  
    }  
  
    leaf hostname {  
        type inet:domain-name;  
    }  
}
```

CBOR diagnostic notation: 1741

CBOR encoding: 19 06CD

Second example:

The following example shows the encoding of the 'reporting-entity' value referencing list instance `"/system/authentication/user/authorized-key/key-data"` (SID 1734) for username `"bob"` and authorized-key `"admin"`.

Definition example from [RFC7317]:

```
list user {  
    key name;  
  
    leaf name {  
        type string;  
    }  
    leaf password {  
        type ianach:crypt-hash;  
    }  
  
    list authorized-key {  
        key name;  
  
        leaf name {  
            type string;  
        }  
        leaf algorithm {  
            type string;  
        }  
        leaf key-data {  
            type binary;  
        }  
    }  
}
```

CBOR diagnostic notation: [1734, "bob", "admin"]

CBOR encoding:

```
83          # array(3)
 19 06C6    # unsigned(1734)
 63         # text(3)
    626F62  # "bob"
 65         # text(5)
    61646D696E # "admin"
```

Third example:

The following example shows the encoding of the 'reporting-entity' value referencing the list instance "/system/authentication/user" (SID 1730) corresponding to user name "jack".

CBOR diagnostic notation: [1730, "jack"]

CBOR encoding:

```
82          # array(2)
 19 06C2    # unsigned(1730)
 64         # text(4)
    6A61636B # "jack"
```

6.13.2. Names as instance-identifier

An "instance-identifier" value is encoded as a string that is analogical to the lexical representation in XML encoding; see Section 9.13.2 in [RFC7950]. However, the encoding of namespaces in instance-identifier values follows the rules stated in Section 3.3, namely:

- o The leftmost (top-level) data node name is always in the namespace qualified form.
- o Any subsequent data node name is in the namespace qualified form if the node is defined in a module other than its parent node, and the simple form is used otherwise. This rule also holds for node names appearing in predicates.

For example,

```
/ietf-interfaces:interfaces/interface[name='eth0']/ietf-ip:ipv4/ip
```

is a valid instance-identifier value because the data nodes "interfaces", "interface", and "name" are defined in the module "ietf-interfaces", whereas "ipv4" and "ip" are defined in "ietf-ip".

The resulting xpath MUST be encoded using a CBOR text string data item (major type 3).

First example:

This example is described in Section 6.13.1.

CBOR diagnostic notation: "/ietf-system:system/contact"

CBOR encoding:

78 1c 2F696574662D73797374656D3A73797374656D2F636F6E74616374

Second example:

This example is described in Section 6.13.1.

CBOR diagnostic notation:

"/ietf-system:system/authentication/user[name='bob']/authorized-key
[name='admin']/key-data"

CBOR encoding:

78 59
2F696574662D73797374656D3A73797374656D2F61757468656E74696361
74696F6E2F757365725B6E616D653D27626F62275D2F617574686F72697A
65642D6B65790D0A5B6E616D653D2761646D696E275D2F6B65792D64617461

Third example:

This example is described in Section 6.13.1.

CBOR diagnostic notation:

"/ietf-system:system/authentication/user[name='bob']"

CBOR encoding:

78 33
2F696574662D73797374656D3A73797374656D2F61757468656E74696361
74696F6E2F757365725B6E616D653D27626F62275D

7. Security Considerations

The security considerations of [RFC7049] and [RFC7950] apply.

This document defines an alternative encoding for data modeled in the YANG data modeling language. As such, this encoding does not contribute any new security issues in addition of those identified for the specific protocol or context for which it is used.

To minimize security risks, software on the receiving side SHOULD reject all messages that do not comply to the rules of this document and reply with an appropriate error message to the sender.

8. IANA Considerations

8.1. Tags Registry

This specification requires the assignment of CBOR tags for the following YANG datatypes. These tags are added to the Tags Registry as defined in section 7.2 of [RFC7049].

| Tag | Data Item | Semantics | Reference |
|-----|--|--|-----------------------------|
| 42 | unsigned integer | YANG Schema Item identifier (sid); see Section 3.2. | [draft-ietf-core-yang-cbor] |
| 43 | byte | YANG bits datatype; see Section 6.7. | [draft-ietf-core-yang-cbor] |
| 44 | string | YANG enumeration datatype; see Section 6.6. | [draft-ietf-core-yang-cbor] |
| 45 | unsigned integer | YANG identityref datatype; see Section 6.10. | [draft-ietf-core-yang-cbor] |
| 46 | or text string unsigned integer or text string or array | YANG instance-identifier datatype; see Section 6.13. | [draft-ietf-core-yang-cbor] |

// RFC Ed.: replace [draft-ietf-core-yang-cbor] with RFC number and remove this note

9. Acknowledgments

This document has been largely inspired by the extensive works done by Andy Bierman and Peter van der Stok on [I-D.ietf-core-comi]. [RFC7951] has also been a critical input to this work. The authors would like to thank the authors and contributors to these two drafts.

The authors would also like to acknowledge the review, feedback, and comments from Ladislav Lhotka and Juergen Schoenwaelder.

10. References

10.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <<https://www.rfc-editor.org/info/rfc6241>>.
- [RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <<https://www.rfc-editor.org/info/rfc7049>>.
- [RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <<https://www.rfc-editor.org/info/rfc7950>>.

10.2. Informative References

- [I-D.ietf-core-comi] Veillette, M., Stok, P., Pelov, A., and A. Bierman, "CoAP Management Interface", draft-ietf-core-comi-04 (work in progress), November 2018.
- [I-D.ietf-core-sid] Veillette, M., Pelov, A., and I. Petrov, "YANG Schema Item Identifier (SID)", draft-ietf-core-sid-06 (work in progress), March 2019.
- [RFC7159] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014, <<https://www.rfc-editor.org/info/rfc7159>>.

- [RFC7223] Bjorklund, M., "A YANG Data Model for Interface Management", RFC 7223, DOI 10.17487/RFC7223, May 2014, <<https://www.rfc-editor.org/info/rfc7223>>.
- [RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, DOI 10.17487/RFC7228, May 2014, <<https://www.rfc-editor.org/info/rfc7228>>.
- [RFC7277] Bjorklund, M., "A YANG Data Model for IP Management", RFC 7277, DOI 10.17487/RFC7277, June 2014, <<https://www.rfc-editor.org/info/rfc7277>>.
- [RFC7317] Bierman, A. and M. Bjorklund, "A YANG Data Model for System Management", RFC 7317, DOI 10.17487/RFC7317, August 2014, <<https://www.rfc-editor.org/info/rfc7317>>.
- [RFC7951] Lhotka, L., "JSON Encoding of Data Modeled with YANG", RFC 7951, DOI 10.17487/RFC7951, August 2016, <<https://www.rfc-editor.org/info/rfc7951>>.
- [RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <<https://www.rfc-editor.org/info/rfc8040>>.
- [RFC8348] Bierman, A., Bjorklund, M., Dong, J., and D. Romascanu, "A YANG Data Model for Hardware Management", RFC 8348, DOI 10.17487/RFC8348, March 2018, <<https://www.rfc-editor.org/info/rfc8348>>.

Authors' Addresses

Michel Veillette (editor)
Trilliant Networks Inc.
610 Rue du Luxembourg
Granby, Quebec J2J 2V2
Canada

Email: michel.veillette@trilliantinc.com

Ivaylo Petrov (editor)
Acklio
1137A avenue des Champs Blancs
Cesson-Sevigne, Bretagne 35510
France

Email: ivaylo@ackl.io

Alexander Pelov
Acklio
1137A avenue des Champs Blancs
Cesson-Sevigne, Bretagne 35510
France

Email: a@ackl.io

CoRE Working Group
Internet-Draft
Intended status: Experimental
Expires: January 8, 2020

I. Jarvinen
M. Kojo
I. Raitahila
University of Helsinki
Z. Cao
Huawei
July 7, 2019

Fast-Slow Retransmission Timeout and Congestion Control Algorithm for
CoAP
draft-jarvinen-core-fasor-02

Abstract

This document specifies an alternative retransmission timeout and congestion control back off algorithm for the CoAP protocol, called Fast-Slow RTO (FASOR).

The algorithm specified in this document employs an appropriate and large enough back off of Retransmission Timeout (RTO) as the major congestion control mechanism to allow acquiring unambiguous RTT samples with high probability and to prevent building a persistent queue when retransmitting. The algorithm also aims to retransmit quickly using an accurately managed retransmission timeout when link-errors are occurring, basing RTO calculation on unambiguous round-trip time (RTT) samples.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on January 8, 2020.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

| | |
|--|----|
| 1. Introduction | 2 |
| 2. Conventions | 3 |
| 3. Problems with Existing CoAP Congestion Control Algorithms . . | 3 |
| 4. FASOR Algorithm | 4 |
| 4.1. Computing Normal RTO (FastRTO) | 4 |
| 4.2. Slow RTO | 5 |
| 4.3. Retransmission Timeout Back Off Logic | 6 |
| 4.3.1. Overview | 6 |
| 4.3.2. Retransmission State Machine | 7 |
| 4.4. Retransmission Count Option | 9 |
| 4.5. Alternatives for Exchanging Retransmission Count Information | 11 |
| 5. Security Considerations | 11 |
| 6. IANA Considerations | 11 |
| 7. References | 11 |
| 7.1. Normative References | 11 |
| 7.2. Informative References | 12 |
| Appendix A. Pseudocode for Basic FASOR without Dithering | 13 |
| Authors' Addresses | 15 |

1. Introduction

CoAP senders use retransmission timeout (RTO) to infer losses that have occurred in the network. For such a heuristic to be correct, the RTT estimate used for calculating the retransmission timeout must match to the real end-to-end path characteristics. Otherwise, unnecessary retransmission may occur. Both default RTO mechanism for CoAP [RFC7252] and CoCoA [I-D.ietf-core-cocoa] have issues in dealing with unnecessary retransmissions and in the worst-case the situation can persist causing congestion collapse [JRCK18a].

This document specifies FASOR retransmission timeout and congestion control algorithm [JRCK18b]. FASOR algorithm ensures unnecessary retransmissions that a sender may have sent due to an inaccurate RTT estimate will not persist avoiding the threat of congestion collapse. FASOR also aims to quickly restore the accuracy of the RTT estimate. Armed with an accurate RTT estimate, FASOR not only handles congestion robustly but also can quickly infer losses due to link errors.

2. Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14, RFC 2119 [RFC2119].

3. Problems with Existing CoAP Congestion Control Algorithms

Correctly inferring losses requires the retransmission timeout (RTO) to be longer than the real RTT in the network. Under certain circumstances the RTO may be incorrectly small. If the real end-to-end RTT is larger than the retransmission timeout, it is impossible for the sender to avoid making unnecessary retransmissions that duplicate data still existing in the network because the sender cannot receive any feedback in time. Unnecessary retransmissions cause two basic problems. First, they increase the perceived end-to-end RTT if the bottleneck has buffering capacity, and second, they prevent getting unambiguous RTT samples. Making unnecessary retransmissions is also a pre-condition for the congestion collapse [RFC0896], which may occur in the worst case if retransmissions are not well controlled [JRCK18a]. Therefore, the sender retransmission timeout algorithm should actively attempt to prevent unnecessary retransmissions from persisting under any circumstance.

Karn's algorithm [KP87] has prevented unnecessary retransmission from turning into congestion collapse for decades due to robust RTT estimation and retransmission timeout backoff handling. The recent CoAP congestion control algorithms, however, diverge from the principles of Karn's algorithm in significant ways and may pose a threat to the stability of the Internet due to those differences.

The default RTO mechanism for CoAP [RFC7252] uses only an initial RTO dithered between 2 and 3 seconds, while CoCoA [I-D.ietf-core-cocoa] measures RTT both from unambiguous and ambiguous RTT samples and applies a modified version of the TCP RTO algorithm [RFC6298]. The algorithm in RFC 7252 lacks solution to persistent congestion. The binary exponential back off used for the retransmission timeout does not properly address unnecessary retransmissions when RTT is larger

than the default RTO (ACK_TIMEOUT). If the CoAP sender performs exchanges over an end-to-end path with such a high RTT, it persistently keeps making unnecessary retransmissions for every exchange wasting some fraction of the used resources (network capacity, battery power).

CoCoA [I-D.ietf-core-cocoa] attempts to improve scenarios with link-error related losses and solve persistent congestion by basing its RTO value on an estimated RTT. However, there are couple of exceptions when the RTT estimation is not available:

- At the beginning of a flow where initial RTO of 2 seconds is used.
- When RTT suddenly jumps high enough to trigger the rule in CoCoA that prevents taking RTT samples when more than two retransmissions are needed. This may also occur when the packet drop rate on the path is high enough.

When RTT estimate is too small, unnecessary retransmission will occur also with CoCoA. CoCoA being unable to take RTT samples at all is a particularly problematic phenomenon as it is similarly persisting state as with the algorithm outlined in RFC 7252 and the network remains in a congestion collapsed state due to persisting unnecessary retransmissions.

4. FASOR Algorithm

FASOR [JRCK18b] is composed of three key components: RTO computation, Slow RTO, and novel retransmission timeout back off logic.

4.1. Computing Normal RTO (FastRTO)

The FASOR algorithm measures the RTT for an CoAP message exchange over an end-to-end path and computes the RTO value using the TCP RTO algorithm specified in [RFC6298]. We call this normal RTO or FastRTO. In contrast to the TCP RTO mechanism, FASOR SHOULD NOT use 1 second lower-bound when setting the RTO because RTO is only a backup mechanisms for loss detection with TCP, whereas with CoAP RTO is the primary and only loss detection mechanism. A lower-bound of 1 second would impact timeliness of the loss detection in low RTT environments. The RTO value MAY be upper-bounded by at least 60 seconds. A CoAP sender using the FASOR algorithm SHOULD set initial RTO to 2 seconds. The computed RTO value as well as the initial RTO value is subject to dithering; they are dithered between $RTO + 1/4 \times SRTT$ and $RTO + SRTT$. For dithering initial RTO, SRTT is unset; therefore, SRTT is replaced with initial RTO / 3 which is derived from the RTO formula and equals to a hypothetical initial RTT that

would yield the initial RTO using the SRTT and RTTVAR initialization rule of RFC 6298. That is, for initial RTO of 2 seconds we use SRTT value of 2/3 seconds.

FastRTO is updated only with unambiguous RTT samples. Therefore, it closely tracks the actual RTT of the network and can quickly trigger a retransmission when the network state is not dubious. Retransmitting without extra delay is very useful when the end-to-end path is subject to losses that are unrelated to congestion. When the first unambiguous RTT sample is received, the RTT estimator is initialized with that sample as specified in [RFC6298] except RTTVAR that is set to R/2K.

4.2. Slow RTO

We introduce Slow RTO as a safe way to ensure that only a unique copy of message is sent before at least one RTT has elapsed. To achieve this the sender must ensure that its retransmission timeout is set to a value that is larger than the path end-to-end RTT that may be inflated by the unnecessary retransmission themselves. Therefore, whenever a message needs to be retransmitted, we measure Slow RTO as the elapsed time required for getting an acknowledgement. That is, Slow RTO is measured starting from the original transmission of the request message until the receipt of the acknowledgement, regardless of the number of retransmissions. In this way, Slow RTO always covers the worst-case RTT during which a number of unnecessary retransmissions were made but the acknowledgement is received for the original transmission. In contrast to computing normal RTO, Slow RTO is not smoothed because it is derived from the sending pattern of the retransmissions (that may turn out unnecessary). In order to drain the potential unnecessary retransmissions successfully from the network, it makes sense to wait for the time used for sending them rather than some smoothed value. However, Slow RTO is multiplied by a factor to allow some growth in load without making Slow RTO too aggressive (by default the factor of 1.5 is used). FASOR then applies Slow RTO as one of the backed off timer values used with the next request message.

Slow RTO allows rapidly converging towards stable operating point because 1) it lets the duplicate copies sent earlier to drain from the network reducing the perceived end-to-end RTT, and 2) allows enough time to acquire an unambiguous RTT sample for the RTO computation. Robustly acquiring the RTT sample ensures that the next RTO is set according to the recent measurement and further unnecessary retransmissions are avoided. Slow RTO itself is a form of back off because it includes the accumulated time from the retransmission timeout back off of the previous exchange. FASOR uses this for its advantage as the time included into Slow RTO is what is

needed to drain all unnecessary retransmissions possibly made during the previous exchange. Assuming a stable RTT and that all of the retransmissions were unnecessary, the time to drain them is the time elapsed from the original transmission to the sending time of the last retransmission plus one RTT. When the acknowledgement for the original transmission arrives, one RTT has already elapsed, leaving only the sending time difference still unaccounted for which is at minimum the value for Slow RTO (when an RTT sample arrives immediately after the last retransmission). Even if RTT would be increasing, the draining still occurs rapidly due to exponentially backed off frequency in sending the unnecessary retransmissions.

4.3. Retransmission Timeout Back Off Logic

4.3.1. Overview

FASOR uses normal RTO as the base for binary exponential back off when no retransmission were needed for the previous CoAP message exchange. When retransmission were needed for the previous CoAP message exchange, the algorithm rules, however, are more complicated than with the traditional RTO back off because Slow RTO is injected into the back off series to reduce high impact of using Slow RTO. FASOR logic chooses from three possible back off series alternatives:

FAST back off: Perform traditional RTO back off with the normal RTO as the base. Applied when the previous message was not retransmitted.

FAST_SLOW_FAST back off: First perform a probe using the normal RTO for the original transmission of the request message to improve cases with losses unrelated to congestion. If the probe for the original transmission of the request message is successful without retransmissions, continue with FAST back off for the next message exchange. If the request message needs to be retransmitted, continue by using Slow RTO for the first retransmission in order to respond to congestion and drain the network from the unnecessary retransmissions that were potentially sent for the previous exchange. If still further RTOs are needed, continue by backing off the normal RTO further on each timeout. FAST_SLOW_FAST back off is applied just once when the previous request message using FAST back off required one or more retransmissions.

SLOW_FAST back off: Perform Slow RTO first for the original transmission to respond to congestion and to acquire an unambiguous RTT sample with high probability. Then, if the original request needs to be retransmitted, continue with the normal RTO-based RTO back off serie by backing off the normal RTO

on each timeout. SLOW_FAST back off is applied when the previous request message using FAST_SLOW_FAST or SLOW_FAST back off required one or more retransmissions. Once an acknowledgement for the original transmission with unambiguous RTT sample is received, continue with FAST back off for the next message exchange.

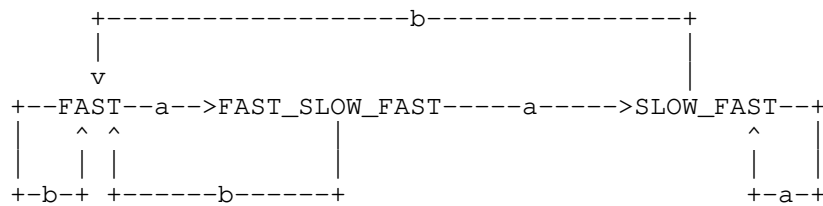
For the initial message, FAST is used with INITIAL_RTO as the FastRTO value. From there on, state is updated when an acknowledgement arrives. Following unambiguous RTT samples, FASOR always uses FAST. Whenever retransmissions are needed, the back off series selection is first downgraded to FAST_SLOW_FAST back off and then to SLOW_FAST back off if further retransmission are needed in FAST_SLOW_FAST.

When Slow RTO is used as the first RTO value, the sender is likely to acquire unambiguous RTT sample even when the network has high delay due to congestion because Slow RTO is based on a very recent measurement of the worst-case RTT. However, using Slow RTO may negatively impact the performance when losses unrelated to congestion are occurring. Due to its potential high cost, FASOR algorithm attempts to avoid using Slow RTO unnecessarily.

The CoAP protocol is often used by devices that are connected through a wireless network where non-congestion related losses are much more frequent than in their wired counterparts. This has implications for the retransmission timeout algorithm. While it would be possible to implement FASOR such that it immediately uses Slow RTO when a dubious network state is detected, which would handle congestion very well, it would do significant harm for performance when RTOs occur due to non-congestion related losses. Instead, FASOR uses first normal RTO for one transmission and only responds using Slow RTO if RTO expires also for that request message. Such a pattern quickly probes if the losses were unrelated to congestion and only slightly delays response if real congestion event is taking place. To ensure that an unambiguous RTT sample is also acquired on a congested network path, FASOR then needs to use Slow RTO for the original transmission of the subsequent packet if the probe was not successful.

4.3.2. Retransmission State Machine

FASOR consists of the three states discussed above while making retransmission decisions, FAST, FAST_SLOW_FAST and SLOW_FAST. The state machine of the FASOR algorithm is depicted in Figure 1.



a: retransmission acknowledged, ambiguous RTT sample acquired;
b: no retransmission, unambiguous RTT sample acquired;

Figure 1: State Machine of FASOR

In the FAST state, if the original transmission of the message has not been acknowledged by the receiver within the time defined by `FastRTO`, the sender will retransmit it. If there is still no acknowledgement of the retransmitted packet within $2 * \text{FastRTO}$, the sender performs the second retransmission and if necessary, each further retransmission applying binary exponential back off of `FastRTO`. The retransmission interval in this state is defined as `FastRTO`, $2^1 * \text{FastRTO}$, ..., $2^i * \text{FastRTO}$.

When there is an acknowledgement after any retransmission, the sender will calculate `SlowRTO` value based on the algorithm defined in Section 4.2.

When there is an acknowledgement after any retransmission, the sender will also switch to the second state, `FAST_FLOW_FAST`. In this state, the retransmission interval is defined as `FastRTO`, $\text{Max}(\text{SlowRTO}, 2 * \text{FastRTO})$, $\text{FastRTO} * 2^1$, ..., $2^i * \text{FastRTO}$. The state will be switched back to the FAST state once an acknowledgement is returned within `FastRTO`, i.e., no retransmission happens for a message. This is reasonable because it shows the network has recovered from congestion or bloated queue.

If some retransmission has been made before the acknowledged arrives in the `FAST_SLOW_FAST` state, the sender updates the `SlowRTO` value, and moves to the third state, `SLOW_FAST`. The retransmission interval in the `SLOW_FAST` state is defined as `SlowRTO`, `FastRTO`, $\text{FastRTO} * 2^1$, ..., $2^i * \text{FastRTO}$.

In `SLOW_FAST` state, the sender switches back to the FAST state if an unambiguous acknowledgement arrives. Otherwise, the sender stays in the `SLOW_FAST` state if retransmission happens again.

4.4. Retransmission Count Option

When retransmissions are needed to deliver a CoAP message, it is not possible to measure RTT for the RTO computation as the RTT sample becomes ambiguous. Therefore, it would be beneficial to be able to distinguish whether an acknowledgement arrives for the original transmission of the message or for a retransmission of it. This would allow reliably acquiring an RTT sample for every CoAP message exchange and thereby compute a more accurate RTO even during periods of congestion and loss.

The Retransmission Count Option is used to distinguish whether an Acknowledgement message arrives for the original transmission or one of the retransmissions of a Confirmable message. However, the Retransmission Count Option cannot be used with an Empty Acknowledgement (or Reset) message because the CoAP protocol specification [RFC7252] does not allow adding options to an Empty message. Therefore, Retransmission Count Option is useful only for the common case of Piggybacked Response. In case of Empty Acknowledgements the operation of FASOR is the same as without the option.

| No. | C | U | N | R | Name | Format | Length | Default |
|-----|---|---|---|---|------------|--------|--------|---------|
| TBD | | | X | | Rexmit-Cnt | uint | 0-1 | 0 |

C=Critical, U=Unsafe, N=NoCacheKey, R=Repeatable

Table 1: Retransmission Count Option

Implementation of the Retransmission Count option is optional and it is identified as elective. However, when it is present in a CoAP message and a CoAP endpoint processes it, it MUST be processed as described in this document. The Retransmission Count option MUST NOT occur more than once in a single message.

The value of the Retransmission Count option is a variable-size (0 to 1 byte) unsigned integer. The default value for the option is the number 0 and it is represented with an empty option value (a zero-length sequence of bytes). However, when a client intends to use Retransmit Count option, it MUST reserve space for it by limiting the request message size also when the value is empty in order to fit the full-sized option into retransmissions.

The Retransmission Count option can be present in both the request and response message. When the option is present in a request it

indicates the ordinal number of the transmission for the request message.

If the server supports (implements) the Retransmission Count option and the option is present in a request, the server MUST echo the option value in its Piggybacked Response unmodified. If the server replies with an Empty Acknowledgement the server MUST silently ignore the option and MUST NOT include it in a later separate response to that request.

When Piggybacked Response carrying the Retransmission Count option arrives, the client uses the option to match the response message to the corresponding transmission of the request. In order to measure a correct RTT, the client must store the timestamp for the original transmission of the request as well as the timestamp for each retransmission, if any, of the request. The resulting RTT sample is used for the RTO computation. If the client retransmitted the request without the option but the response includes the option, the client MUST silently ignore the option.

The original transmission of a request is indicated with the number 0, except when sending the first request to a new destination endpoint. The first original transmission of the request to a new endpoint carries the number 255 (0xFF) and is interpreted the same as an original transmission carrying the number 0. Retransmissions, if any, carry the ordinal number of the retransmission. Once the first Piggybacked Response from the new endpoint arrives the client learns whether or not the other endpoint implements the option. If the first response includes the echoed option, the client learns that the other endpoint supports the option and may continue including the option to each retransmitted request. From this point on the original transmissions of requests implicitly include the option number 0 and a zero-byte integer will be sent according to the CoAP uint-encoding rules. If the first Piggybacked Response does not include the option, the client SHOULD stop including the option into the requests to that endpoint.

When the Retransmission Count option is in use, the client bases the retransmission timeout for the normal RTO in the back off series as follows:

$\max(\text{RTO}, \text{Previous-RTT-Sample})$

Previous-RTT-Sample is the RTT sample acquired from the previous message exchange. If no RTT sample was available with the previous message exchange (e.g., the server replied with an Empty Acknowledgement), RTO computed earlier is used like in case the Retransmission Count option is not in use.

4.5. Alternatives for Exchanging Retransmission Count Information

An alternative way of exchanging the retransmission count information between a client and server is to encode it in the Token. The Token is a client-local identifier and a client solely decides how it generates the Token. Therefore, including a varying Token value to retransmissions of the same request is all possible as long as the client can use the Token to differentiate between requests and match a response to the corresponding request. The server is required to make no assumptions about the content or structure of a Token and always echo the Token unmodified in its response.

How exactly a client encodes the retransmission count into a Token is an implementation issue. Note that the original transmission of a request may carry a zero-length Token given that the rules for generating a Token as specified in RFC 7252 [RFC7252] are followed. This allows reducing the overhead of including the Token into the requests in such cases where Token could otherwise be omitted. However, similar to Retransmit Count option the maximum request message size MUST be limited to accommodate the Token with retransmit count into the retransmissions of the request.

5. Security Considerations

6. IANA Considerations

This memo includes no request to IANA.

7. References

7.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC6298] Paxson, V., Allman, M., Chu, J., and M. Sargent, "Computing TCP's Retransmission Timer", RFC 6298, DOI 10.17487/RFC6298, June 2011, <<https://www.rfc-editor.org/info/rfc6298>>.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<https://www.rfc-editor.org/info/rfc7252>>.

7.2. Informative References

- [I-D.ietf-core-cocoa]
Bormann, C., Betzler, A., Gomez, C., and I. Demirkol,
"CoAP Simple Congestion Control/Advanced", draft-ietf-
core-cocoa-03 (work in progress), February 2018.
- [JRCK18a] Jarvinen, I., Raitahila, I., Cao, Z., and M. Kojo, "Is
CoAP Congestion Safe?", Applied Networking Research
Workshop (ANRW'18), July 2018.
- [JRCK18b] Jarvinen, I., Raitahila, I., Cao, Z., and M. Kojo, "FASOR
Retransmission Timeout and Congestion Control Mechanism
for CoAP?", Proceedings of IEEE Global Communications
Conference (Globecom 2018), to appear, December 2018.
- [KP87] Karn, P. and C. Partridge, "Improving Round-trip Time
Estimates in Reliable Transport Protocols", SIGCOMM'87
Proceedings of the ACM Workshop on Frontiers in Computer
Communications Technology, August 1987.
- [RFC0896] Nagle, J., "Congestion Control in IP/TCP Internetworks",
RFC 896, DOI 10.17487/RFC0896, January 1984,
<<https://www.rfc-editor.org/info/rfc896>>.

Appendix A. Pseudocode for Basic FASOR without Dithering

```
var state = NORMAL_RTO

rfc6298_init(var fastrto, 2 secs)

var slowrto
SLOWRTO_FACTOR = 1.5

var original_sendtime
var retransmit_count

/*
 * Sending Original Copy and Retransmitting 'req'
 */
send_request(req) {
    original_sendtime = time.now
    retransmit_count = 0

    arm_rto(calculate_rto())
    send(req)
}

rto_for(req) {
    retransmit_count += 1

    arm_rto(calculate_rto())
    send(req)
}

/*
 * ACK Processings
 */
ack() {
    sample = time.now - original_sendtime
    if (retransmit_count == 0)
        unambiguous_ack(sample)
    else
        ambiguous_ack(sample)
}

unambiguous_ack(sample) {
    k = 4 // RFC6298 default K = 4
    if (rfc6298_is_first_sample(fastrto))
        k = 1
    rfc6298_update(fastrto, k, sample) // Normal RFC6298 processing
    state = NORMAL_RTO
}
```

```
ambiguous_nextstate = {
    [NORMAL_RTO] = FAST_SLOW_FAST_RTO,
    [FAST_SLOW_FAST_RTO] = SLOW_FAST_RTO,
    [SLOW_FAST_RTO] = SLOW_FAST_RTO
}

ambiguous_ack(sample) {
    slowrto = sample * SLOWRTO_FACTOR
    state = ambiguous_nextstate[state]
}

/*
 * RTO Calculations
 */
calculate_rto() {
    return <state>_rtoseries()
}

normal_rtoseries() {
    switch (retransmit_count) {
        case 0: return fastrto_series_init()
        default: return fastrto_series_backoff()
    }
}

fastslowfast_rtoseries() {
    switch (retransmit_count) {
        case 0: return fastrto_series_init()
        case 1: return MAX(slowrto, 2*fastrto)
        default: return fastrto_series_backoff()
    }
}

slowfast_rtoseries() {
    switch (retransmit_count) {
        case 0: return slowrto
        case 1: return fastrto_series_init()
        default: return fastrto_series_backoff()
    }
}

var backoff_series_timer

fastrto_series_init() {
    backoff_series_timer = fastrto
    return backoff_series_timer
}
```



```
fastrto_series_backoff() {  
    backoff_series_timer *= 2  
    return backoff_series_timer  
}
```

Figure 2

Authors' Addresses

Ilpo Jarvinen
University of Helsinki
P.O. Box 68
FI-00014 UNIVERSITY OF HELSINKI
Finland

EMail: ilpo.jarvinen@cs.helsinki.fi

Markku Kojo
University of Helsinki
P.O. Box 68
FI-00014 UNIVERSITY OF HELSINKI
Finland

EMail: markku.kojo@cs.helsinki.fi

Iivo Raitahila
University of Helsinki
P.O. Box 68
FI-00014 UNIVERSITY OF HELSINKI
Finland

EMail: iivo.raitahila@helsinki.fi

Zhen Cao
Huawei
Beijing
China

EMail: zhencao.ietf@gmail.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 9, 2020

A. Keranen
Ericsson
C. Bormann
Universitaet Bremen TZI
July 8, 2019

SenML Data Value Content-Format Indication
draft-keranen-core-senml-data-ct-02

Abstract

The Sensor Measurement Lists (SenML) media type supports multiple types of values, from numbers to text strings and arbitrary binary data values. In order to simplify processing of the data values this document proposes to specify a new SenML field for indicating the Content-Format of the data.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on January 9, 2020.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of

the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

| | |
|--|---|
| 1. Introduction | 2 |
| 2. Terminology | 3 |
| 3. SenML Content-Format ("ct") Field | 3 |
| 4. Using Content-Type and Content-Coding With the "ct" Field . . | 3 |
| 5. SenML Base Content-Format ("bct") Field | 4 |
| 6. Security Considerations | 4 |
| 7. IANA Considerations | 4 |
| Acknowledgements | 4 |
| 9. References | 5 |
| 9.1. Normative References | 5 |
| 9.2. Informative References | 5 |
| Authors' Addresses | 6 |

1. Introduction

The Sensor Measurement Lists (SenML) media type [RFC8428] can be used to send various different kinds of data. In the example given in Figure 1, a temperature value, an indication whether a lock is open, and a data value (with SenML field "vd") read from an NFC reader is sent in a single SenML pack.

```
[
  { "bn": "urn:dev:ow:10e2073a01080063:", "n": "temp", "u": "Cel", "v": 7.1 },
  { "n": "open", "vb": false },
  { "n": "nfc-reader", "vd": "aGkgCg" }
]
```

Figure 1: SenML pack with unidentified binary data

The receiver is expected to know how to decode the data in the "vd" field based on the context, e.g., name of the data source and out-of-band knowledge of the application. However, this context may not always be easily available to entities processing the SenML pack. To facilitate automatic decoding it is useful to be able to indicate an Internet media type and content-coding right in the SenML Record. The CoAP Content-Format (Section 12.3 in [RFC7252]) provides just this information; enclosing a Content-Format number (in this case number 60 as defined for media type application/cbor in [RFC7049]) in the Record is illustrated in Figure 2. All registered CoAP content formats are listed in the Content Formats subregistry of the CoRE Parameters registry [IANA.core-parameters].

```
{"n":"nfc-reader", "vd":"gmNmb28YKg", "ct":60}
```

Figure 2: SenML Record with binary data identified as CBOR

In this example SenML Record the data value contains a string "foo" and a number 42 encoded in a CBOR [RFC7049] array. Since the example above uses the JSON format of SenML, the data value containing the binary CBOR value is base64-encoded. The data value after base64 decoding is shown with CBOR diagnostic notation in Figure 3.

```
82          # array(2)
  63        # text(3)
    666F6F  # "foo"
  18 2A     # unsigned(42)
```

Figure 3: Example Data Value in CBOR diagnostic notation

2. 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 BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

Readers should also be familiar with the terms and concepts discussed in [RFC8428]. Awareness of terminology issues discussed in [I-D.bormann-core-media-content-type-format] can also be very helpful.

3. SenML Content-Format ("ct") Field

When a SenML Record contains a Data Value field ("vd"), the Record MAY also include a Content-Format indication field. The Content-Format indication uses label "ct" and an unsigned integer value in the range of 0-65535 indicating the CoAP Content-Format of the data (similar to CoRE Link Format [RFC6690] "ct" attribute, except that the Link Format attribute happens to be encoded as a string).

4. Using Content-Type and Content-Coding With the "ct" Field

Since some Internet media types and their content coding and parameter alternatives do not have assigned CoAP Content-Format identifiers, the "ct" field can alternatively be used to indicate the Content-Type and Content-Coding of the data. If Content-Coding is not specified with Content-Type, identity (i.e., no) transformation is used.

If Content-Coding is used, the content-coding value (e.g., "deflate") MUST be prefixed with "@" and concatenated with the media type and parameters (if any). For example: "application/json@deflate".

5. SenML Base Content-Format ("bct") Field

The Base Content-Format Field, label "bct", provides a default value for the Content-Format Field (label "ct") within its range. The range of the base field includes the record containing it, up to (but not including) the next record containing a "bct" field, if any, or up to the end of the pack otherwise. Resolution (Section 4.6 of [RFC8428]) of this base field is performed by adding its value with the label "ct" to all records in this range that carry a "vd" field but do not already contain a Content-Format ("ct") field.

6. Security Considerations

The indication of a media type in the data does not exempt a consuming application from properly checking its inputs. Also, the ability for an attacker to supply crafted SenML data that specify media types chosen by the attacker may expose vulnerabilities of handlers for these media types to the attacker.

7. IANA Considerations

IANA is requested to assign new labels in the "SenML Labels" subregistry of the SenML registry [IANA.senml] (as defined in [RFC8428]) for the Content-Format indication as per Table 1:

| Name | Label | JSON Type | XML Type | Reference |
|---------------------|-------|---------------|------------|---------------|
| Base Content-Format | bct | Number/String | int/string | this document |
| Content-Format | ct | Number/String | int/string | this document |

Table 1: IANA Registration for new SenML Labels

Acknowledgements

The authors would like to thank Sergio Abreu for the discussions leading to the design of this extension.

9. References

9.1. Normative References

- [IANA.senml]
IANA, "Sensor Measurement Lists (SenML)",
<<http://www.iana.org/assignments/senml>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://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, <<https://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, <<https://www.rfc-editor.org/info/rfc7252>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8428] Jennings, C., Shelby, Z., Arkko, J., Keranen, A., and C. Bormann, "Sensor Measurement Lists (SenML)", RFC 8428, DOI 10.17487/RFC8428, August 2018, <<https://www.rfc-editor.org/info/rfc8428>>.

9.2. Informative References

- [I-D.bormann-core-media-content-type-format]
Bormann, C., "On Media-Types, Content-Types, and related terminology", draft-bormann-core-media-content-type-format-00 (work in progress), March 2019.
- [IANA.core-parameters]
IANA, "Constrained RESTful Environments (CoRE) Parameters",
<<http://www.iana.org/assignments/core-parameters>>.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <<https://www.rfc-editor.org/info/rfc6690>>.

Authors' Addresses

Ari Keranen
Ericsson
Jorvas 02420
Finland

Email: ari.keranen@ericsson.com

Carsten Bormann
Universitaet Bremen TZI
Postfach 330440
Bremen D-28359
Germany

Phone: +49-421-218-63921
Email: cabo@tzi.org

CoRE Working Group
Internet-Draft
Updates: 7252, 7390, 7641 (if approved)
Intended status: Standards Track
Expires: January 7, 2020

M. Tiloca
R. Hoeglund
RISE AB
C. Amsuess

F. Palombini
Ericsson AB
July 06, 2019

Observe Notifications as CoAP Multicast Responses
draft-tiloca-core-observe-multicast-notifications-00

Abstract

The Constrained Application Protocol (CoAP) allows clients to "observe" resources at a server, and receive notifications as unicast responses upon changes of the resource state. In some use cases, such as based on publish-subscribe, it would be convenient for the server to send a single notification to all the clients observing a same target resource. This document defines how a CoAP server sends observe notifications as response messages over multicast, by synchronizing all the observers of a same resource on a same shared Token value. Besides, this document defines how Group OSCORE can be used to protect multicast notifications end-to-end from the CoAP server to the multiple CoAP clients registered as observers.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on January 7, 2020.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

| | |
|--|----|
| 1. Introduction | 2 |
| 1.1. Terminology | 4 |
| 2. The Override-Token Option | 4 |
| 3. The Override-AAD Option | 5 |
| 4. Resource Observation | 6 |
| 4.1. Client Registration | 6 |
| 4.2. Multicast Notifications | 7 |
| 4.3. Example | 8 |
| 5. Token Values for Multicast Notifications | 9 |
| 6. Intermediaries | 11 |
| 7. Protection of Multicast Notifications with Group OSCORE | 11 |
| 7.1. Secure Binding of Multicast Notifications | 12 |
| 7.2. Example | 13 |
| 8. Security Considerations | 15 |
| 9. IANA Considerations | 15 |
| 9.1. CoAP Option Numbers Registry | 15 |
| 10. References | 16 |
| 10.1. Normative References | 16 |
| 10.2. Informative References | 17 |
| Acknowledgments | 17 |
| Authors' Addresses | 18 |

1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] has been extended with a number of mechanisms, including resource Observation [RFC7641]. This enables CoAP clients to register at a CoAP server as "observers" of a resource, and hence being automatically notified with an unsolicited response upon changes of the resource state.

CoAP supports group communication over IP multicast [RFC7390], and [I-D.dijk-core-groupcomm-bis] has been enabling Observe registration requests over multicast, in order for clients to efficiently register as observers of a resource hosted at multiple servers.

However, in a number of use cases, the opposite usage of multicast messages would be also desirable. That is, it would be useful that a server sends observe notifications for a same target resource to multiple observer clients, as responses over IP multicast.

For instance, in CoAP publish-subscribe [I-D.ietf-core-coap-pubsub], multiple clients can subscribe to a topic, by observing the related resource hosted at the responsible broker. When a new value is published on that topic, it would be convenient for the broker to send a single multicast notification at once, to all the subscriber clients observing that topic.

A different use case concerns clients observing a registration resource at the CoRE Resource Directory [I-D.ietf-core-resource-directory]. For example, multiple clients can benefit of observation for discovering (to-be-created) OSCORE groups [I-D.ietf-core-oscore-groupcomm] and retrieving updated information to join them through their respective Group Manager [I-D.tiloca-core-oscore-discovery].

More in general, multicast notifications would be beneficial whenever several CoAP clients observe a same target resource at a CoAP server, and can be all notified at once by means of a single response message. However, CoAP does not currently define response messages over IP multicast. This specification fills this gap and provides the following twofold contribution.

First, it defines a method to deliver Observe notifications as CoAP responses over IP multicast. The proposed method relies on the server managing the Token space for multicast notifications, by providing all the observers of a target resource with the same Token value to bind to their own observation. That Token value is used in every multicast notification for that target resource. This is achieved by introducing a new CoAP option used in the notification response to the original registration request from each client.

Second, this specification defines how to use Group OSCORE [I-D.ietf-core-oscore-groupcomm] to protect multicast notifications end-to-end between the server and the observer clients. This is achieved by introducing a second new CoAP option used in the notification response to the original registration request from each client. The option specifies parameter values that the server uses to secure every multicast notification for the target resource by

using Group OSCORE. This provides a secure binding between each of such notifications and the observation of each of the clients.

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 BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

Readers are expected to be familiar with terms and concepts described in CoAP [RFC7252], group communication for CoAP [RFC7390] [I-D.dijk-core-groupcomm-bis], Observe [RFC7641], CBOR [RFC7049], OSCORE [I-D.ietf-core-object-security], and Group OSCORE [I-D.ietf-core-oscore-groupcomm].

2. The Override-Token Option

The Override-Token option defined in this section has the properties summarized in Figure 1, which extends Table 4 of [RFC7252].

Since the option is not Safe-to-Forward, the column "N" is filled with a dash. The Override-Token option contains a Token value.

| No. | C | U | N | R | Name | Format | Length | Default |
|------|---|---|---|---|----------------|--------|--------|---------|
| TBD1 | X | x | - | | Override-Token | opaque | 1-8 | (none) |

C=Critical, U=Unsafe, N=NoCacheKey, R=Repeatable

Figure 1: The Override-Token Option.

This document specifically defines how this option is used to support Observe notifications over IP multicast (see Section 4).

If a server provides multicast notifications for a target resource, the server includes the Override-Token option in the unicast notification response sent as the first reply to a registration request from each client to that resource, i.e. a GET request with the Observe option set to 0. The server indicates a same immutable Token value T in the Override-Token option of each of these first replies. In any other circumstance, this option is not included.

The server will use that same Token value T when sending multicast notifications to registered clients observing that resource. This

ensures that every multicast notification for that resource is expected on the same Token value T by each observing client.

The Override-Token option is of class U for OSCORE [I-D.ietf-core-object-security][I-D.ietf-core-oscore-groupcomm], since intermediaries may be present, and may replace it with a new instance (see Section 6).

3. The Override-AAD Option

The Override-AAD option defined in this section has the properties summarized in Figure 2, which extends Table 4 of [RFC7252].

| No. | C | U | N | R | Name | Format | Length | Default |
|------|---|---|---|---|--------------|--------|--------|---------|
| TBD2 | X | | | | Override-AAD | (*) | 3-255 | (none) |

C=Critical, U=Unsafe, N=NoCacheKey, R=Repeatable
(*) See below.

Figure 2: The Override-AAD Option

If a server that provides multicast notifications for a target resource protects them with Group OSCORE [I-D.ietf-core-oscore-groupcomm], the server includes the Override-AAD option in the unicast notification response sent as the first reply to a registration request from each client to that resource, i.e. a GET request with the Observe option set to 0. In any other circumstance, this option is not included.

The Override-AAD option contains a CBOR array [RFC7049] composed of the two following elements.

- o The first element is a CBOR byte string, which encodes the Sender ID of the server in the OSCORE group.
- o The second element is a CBOR integer, which encodes a particular value SN of the Sender Sequence Number of the server in the OSCORE group.

The server uses this same immutable pair to build the two OSCORE 'external_aad' (see Section 5.4 of [I-D.ietf-core-object-security] and Sections 3.1 and 3.2 of [I-D.ietf-core-oscore-groupcomm]), when encrypting and countersigning every multicast notification for the observed resource using Group OSCORE [I-D.ietf-core-oscore-groupcomm].

This ensures that every multicast notification for a same observed resource is securely bound to the first unicast notification sent to each client observing that resource.

The Override-AAD option is of class E for OSCORE [I-D.ietf-core-object-security] [I-D.ietf-core-oscore-groupcomm].

4. Resource Observation

Clients interested in receiving multicast notifications from a server have to first register their interest, as described in Section 4.1. This registration is performed over unicast, i.e. comprising both the observation request and the first notification response.

Upon a change of the state of the target resource, the server sends a multicast notification, i.e. a single CoAP response over Multicast IP intended to all the clients in the list of observers of that resource, as described in Section 4.2. Multicast notifications MUST be non-confirmable.

Interested clients need to know the IP multicast address and UDP port number where the server sends multicast notifications for the target resource(s). To this end, a possible approach may rely on the CoRE Resource Directory (RD) and the RD-Groups usage pattern (see Appendix A of [I-D.ietf-core-resource-directory]). In particular, application groups may be registered to the RD, as composed of the resource(s) for which a server provides multicast notifications, and specifying the used IP multicast address and UDP port number. Further details on how clients retrieve this information are out of the scope of this specification.

The server MUST NOT send multicast notifications to unmanaged IP multicast addresses, such as All CoAP Nodes (see Section 12.8 of [RFC7252]).

4.1. Client Registration

The registration process occurs according to the following steps.

1. A client sends an observation request to the server as described in [RFC7641], i.e. a GET request with an Observe option set to 0 (register).
2. If the list of observers for the target resource has just changed from empty to including one observer, the server selects a currently available value T from its Token space and exclusively assigns it as Token value to the list of observers of the target resource (see also related considerations in Section 5). That

is, from then on, the server MUST use T as its own local Token value associated to that observation, with respect to the (next hop towards the) client.

3. The server adds the client to the list of observers of the target resource.
4. The server sends a unicast response notification to the client as described in [RFC7641], i.e. a 2.05 (Content) or 2.03 (Valid) response with an Observe option including a sequence number. Additionally, the server includes an Override-Token option defined in Section 2, which MUST contain the value T. Note that every client further added to the same non-empty list of observers of that target resource receives a notification response to its registration request with the same value T included in the Override-Token option.
5. Upon receiving the unicast notification response to the observation request, the client retrieves the Override-Token option and the conveyed value T. The client interprets this response as if the server: i) has successfully added an entry with the client endpoint and Token value T to the list of observers of the target resource; and ii) will notify changes to the state of the target resource, by means of multicast notifications with Token value T. The client MAY adopt a policy for re-registering its interest for observation, if the Override-Token option includes a Token value already in use with that server. Clients MUST treat as non valid and silently discard responses that include an Override-Token option but do not include also an Observe option.
6. From then on, the client MUST be able to receive, accept and process multicast notifications about the state of the target resource from the server. To this end, the client is required to know in advance the IP multicast address and port number where the server will send multicast notifications to. Also, from then on, the client MUST use T as its own local Token value associated to that observation, with respect to the (next hop towards the) server. The particular way to achieve this is implementation specific.

4.2. Multicast Notifications

Upon a change of the status of the target resource, the server sends a multicast notification intended to all the clients in the list of observers of that resource. In particular, a multicast notification MUST include an Observe option, as specified in [RFC7641].

Compared to notifications as described in [RFC7641], the following two differences apply for a multicast notification.

- o It is sent as a single CoAP response over Multicast IP.
- o It has Token value T, as indicated to every interested client in the Override-Token option of the notification response to its observation request to the target resource (see Section 4.1).

That is, every multicast notification for a target resource is not bound to the different original observation requests, but rather to the whole set of clients currently in the list of observers of that resource.

4.3. Example

The following example refers to two clients C_1 and C_2 that register to observe a resource /r at a Server S.

Before the following exchanges occur, no clients are observing the resource /r , which has value "1234".

```

C_1 ----- [ Unicast ] -----> S /r
  GET
  Token: 0x4a
  Observe: 0 (Register)

  (S adds C_1 to the list of observers of /r .)

  (S allocates the available Token value 0xff .)

C_1 <----- [ Unicast ] ----- S
  2.05
  Token: 0x4a
  Observe: 10
  Override-Token: 0xff
  Payload: "1234"

C_2 ----- [ Unicast ] -----> S /r
  GET
  Token: 0x01
  Observe: 0 (Register)

  (S adds C_2 to the list of observers of /r .)

C_2 <----- [ Unicast ] ----- S
  2.05
  Token: 0x01
  Observe: 10
  Override-Token: 0xff
  Payload: "1234"

  (The value of the resource /r changes to "5678".)

C_1
+ <----- [ Multicast ] ----- S
C_2
  2.05
  Token: 0xff
  Observe: 11
  Payload: "5678"

```

5. Token Values for Multicast Notifications

The Token space for multicast notifications is shared by all the clients that have registered to observe resources at a server. As described in Section 4, the server aligns all the clients observing a

same resource to consider a same Token value, which is then used in every multicast notification sent for that resource.

This specification updates [RFC7252] by defining the Token values in Figure 3 as intended for multicast notifications.

A server supporting the Override-Token option and Observe multicast notifications MUST use the Token values in Figure 3 for its multicast notifications and as content of the Override-Token option. A server MUST NOT use the same Token value in multicast notifications for multiple resources currently under observation.

A client supporting the Override-Token option and Observe multicast notifications MUST NOT use the Token values in Figure 3 for its outgoing messages, except when explicitly cancelling the observation, i.e. a GET request to the server with an Observe option set to 1 (see Section 3.6 of [RFC7641]). That GET request has the same Token value used by the server in the multicast notifications for that observation.

This ensures clients to correctly distinguish a multicast notification from a regular (notification) response, as well as to correctly bind the former with the corresponding observation, while the latter with the corresponding original request.

| Token size (Bytes) | Token value range | Range size |
|-----------------------|--------------------------------|---------------|
| 1 | [0xf0 , 0xff] | 16 |
| 2 | [0xffc0 , 0xffff] | 64 |
| 3 | [0xfffff00 , 0xffffffff] | 256 |
| 4 | [0xfffffbbf , 0xfffffffff] | 1024 |
| 5 | [0xfffffff7ff , 0xfffffffffff] | 2048 |
| 6 | [0xfffffffef , 0xfffffffffff] | 4096 |
| 7 | [0xfffffffdf , 0xfffffffffff] | 8192 |
| 8 | [0xfffffffdb , 0xfffffffffff] | 16384 |

Figure 3: Range of Token values for multicast notifications.

6. Intermediaries

In case intermediaries such as CoAP proxies are involved, the same approach described in Section 4 is used independently on both the client- and server-side of each proxy. Care must be taken to only use IP multicast addresses that have all the same meaning on all interfaces of the involved hops.

Upon receiving the unicast response to an original observation request, a proxy on the path between the server and a client performs the following actions.

- o The proxy retrieves the Token value T from the Override-Token option. From then on, the proxy MUST use T as its own local Token value associated to that observation, with respect to the next hop towards the server.
- o The proxy MAY remove the original Override-Token option. In such a case, the proxy MUST include a new Override-Token option. The newly included Override-Token option specifies a Token value T' (which may be equal to T), consistently with the rules defined in Section 5. From then on, the proxy uses T' as its own local Token value associated to that observation, with respect to the next hop towards the clients. Otherwise, if the proxy does not remove the Override-Token option, the proxy uses T as its own local Token value associated to that observation, with respect to the next hop towards the clients.

The process described above starts at the server and continues until the clients are eventually reached. Even in the presence of intermediaries, this ensures general conflict-free synchronization of Token values at each hop on the path from the server to the clients.

7. Protection of Multicast Notifications with Group OSCORE

A server can protect multicast notifications by using Group OSCORE [I-D.ietf-core-oscore-groupcomm]. In such a case, both the server and the clients interested in receiving multicast notifications from that server have to be members of the same OSCORE group.

Building on the approach suggested in Section 4.1 to discover IP multicast addresses and UDP port numbers, clients may discover the OSCORE group to refer to by using the method in [I-D.tiloca-core-oscore-discovery], also based on the CoRE Resource Directory (RD) [I-D.ietf-core-resource-directory].

Furthermore, both the clients and server may join the OSCORE group by using the approach described in [I-D.ietf-ace-key-groupcomm-oscore]

and based on the ACE framework for Authentication and Authorization in constrained environments [I-D.ietf-ace-oauth-authz].

Further details on how to discover the OSCORE group and join it are out of the scope of this specification.

Alternative security protocols than Group OSCORE, such as OSCORE [I-D.ietf-core-object-security] and/or DTLS [RFC6347], can be used to protect other unicast exchanges between the server and each client, including the original client registration described in Section 4.1.

7.1. Secure Binding of Multicast Notifications

When using Group OSCORE to protect multicast notifications, the registration process occurs as described in Section 4.1, with the following additions.

- o If the list of observers for the target resource has just changed from empty to including one observer, the server consumes the current value of its own Sender Sequence Number SN in the OSCORE group, and hence updates it to $SN^* = (SN + 1)$.

Note for implementation: a possible way to achieve this is for the server to produce a dummy request addressed to the OSCORE group, and protect it using its own Sender Context of the Group OSCORE Security Context. This dummy request is not actually transmitted, i.e. it does not hit the wire.

- o Upon sending the unicast first notification response to a just registered client, the server includes in that response an Override-AAD option defined in Section 3. The option MUST contain the pair ('kid' ; 'piv') encoded as defined in Section 3, where 'kid' is the Sender ID of the server in the OSCORE group, while 'piv' is the previously consumed Sender Sequence Number value SN of the server in the OSCORE group, i.e. $(SN^* - 1)$. Note that every client further added to the same non-empty list of observers of that target resource receives a notification response to its registration request including the exact same pair ('kid' ; 'piv') in the Override-AAD option.
- o Upon receiving the unicast notification response to the observation request, the client retrieves the Override-AAD option and the conveyed pair ('kid' ; 'piv'). From then on, when verifying multicast notifications as described in Section 6.4 of [I-D.ietf-core-oscore-groupcomm], the client MUST use 'kid' as 'request_kid' and 'piv' as 'request_piv' in the two 'external_aad' for decrypting and verifying every multicast notification from the server for the target resource (see Sections 3.1 and 3.2 of

[I-D.ietf-core-oscore-groupcomm])). The particular way to achieve this is implementation specific. Clients MUST treat as non valid and silently discard responses that include an Override-AAD option but that do not include also both an Override-Token option and an Observe option. A client has to be a current member of the OSCORE group comprising also the server and associated to the target resource, and MUST otherwise silently discard responses that include an Override-AAD option.

Upon sending every multicast notification for the target resource as described in Section 4.2, the server protects it with Group OSCORE. In particular, the process described in Section 6.3 of [I-D.ietf-core-oscore-groupcomm] applies, with the following differences when building the two OSCORE 'external_aad' to encrypt and countersign the multicast notification (see Sections 3.1 and 3.2 of [I-D.ietf-core-oscore-groupcomm]).

- o The 'request_kid' contains the 'kid' value that the server specifies to clients as first element in the Override-AAD option, when replying to the their registration request.
- o The 'request_piv' contains the 'piv' value that the server specifies to clients as second element in the Override-AAD option, when replying to their registration request.

7.2. Example

The following example refers to two clients C_1 and C_2 that register to observe a resource /r at a Server S. Pairwise communication over unicast are protected with OSCORE, while S protects multicast notifications with Group OSCORE.

Before the following exchanges occur, no clients are observing the resource /r , which has value "1234". In addition:

- o C_1 and S have a pairwise OSCORE Security Context. In particular, C_1 has 'kid' = 1 as Sender ID, and SN_1 = 101 as Sequence Number. Also, S has 'kid' = 3 as Sender ID, and SN_3 = 301 as Sequence Number.
- o C_2 and S have a pairwise OSCORE Security Context. In particular, C_2 has 'kid' = 2 as Sender ID, and SN_2 = 201 as Sequence Number. Also, S has 'kid' = 4 as Sender ID, and SN_4 = 401 as Sequence Number.
- o C_1, C_2 and S are members of an OSCORE group with 'kid_context' = "feedca57ab2e" as Group ID. In the OSCORE group, S has 'kid' = 5 as Sender ID, and SN_5 = 501 as Sequence Number.

```

C_1 ----- [ Unicast w/ OSCORE ] -----> S /r
| GET
| Token: 0x4a
| Observe: 0 (Register)
| OSCORE: {kid: 1 ; piv: 101 ; ...}
|
| (S adds C_1 to the list of observers of /r .)
|
| (S allocates the available Token value 0xff .)
|
| (S steps SN_5 in the Group OSCORE Sec. Ctx : SN_5 <= 502)
|
C_1 <----- [ Unicast w/ OSCORE ] ----- S
| 2.05
| Token: 0x4a
| Observe: 10
| OSCORE: {piv: 301; ...}
| Override-Token: 0xff
| Override-AAD: {5 ; 501}
| Payload: "1234"
|
C_2 ----- [ Unicast w/ OSCORE ] -----> S /r
| GET
| Token: 0x01
| Observe: 0 (Register)
| OSCORE: {kid: 2 ; piv: 201 ; ...}
|
| (S adds C_2 to the list of observers of /r .)
|
C_2 <----- [ Unicast w/ OSCORE ] ----- S
| 2.05
| Token: 0x01
| Observe: 10
| OSCORE: {piv: 401 ; ...}
| Override-Token: 0xff
| Override-AAD: {5 ; 501}
| Payload: "1234"
|
| (The value of the resource /r changes to "5678".)
|
C_1
+ <----- [ Multicast w/ Group OSCORE ] ----- S
C_2
| 2.05
| Token: 0xff
| Observe: 11
| OSCORE: {kid: 5 ; piv: 502 ; ...}
| Payload: "5678"

```

The two `external_aad` used to encrypt and countersign the multicast notification above have `'req_kid' = 5` and `'req_iv' = 501`, as indicated in the `Override-AAD` option to the two clients. Thus, the two clients can build the two same `external_aad` for decrypting and verifying this multicast notification and the following ones.

8. Security Considerations

The same security considerations from [RFC7252][RFC7390][RFC7641][I-D.dijk-core-groupcomm-bis][I-D.ietf-core-object-security][I-D.ietf-core-oscore-groupcomm] hold for this document.

The `Override-Token` option is of class U for OSCORE, hence intermediaries and on-path active adversaries are able to modify its value. This yields the same effects of altering the `Token` value of CoAP messages.

If multicast notifications are protected using Group OSCORE, the original registration requests and related unicast notification responses MUST also be secured. This prevents on-path active adversaries from altering the `Override-AAD` option, and thus ensures secure binding between every multicast notification for a same observed resource and the first notification response sent to each client observing that resource.

To this end, clients and servers MUST use OSCORE or Group OSCORE, for which the `Override-AAD` option is of class E and would thus be hidden also from intermediaries such as CoAP proxies. This ensures that the secure binding above is enforced end-to-end between the server and each observing client.

9. IANA Considerations

This document has the following actions for IANA.

9.1. CoAP Option Numbers Registry

IANA is asked to enter the following option numbers to the "CoAP Option Numbers" registry defined in [RFC7252] within the "CoRE Parameters" registry.

| Number | Name | Reference |
|--------|----------------|-------------------|
| TBD1 | Override-Token | [[this document]] |
| TBD2 | Override-AAD | [[this document]] |

10. References

10.1. Normative References

- [I-D.dijk-core-groupcomm-bis]
Dijk, E., Wang, C., and M. Tiloca, "Group Communication for the Constrained Application Protocol (CoAP)", draft-dijk-core-groupcomm-bis-00 (work in progress), March 2019.
- [I-D.ietf-core-object-security]
Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security for Constrained RESTful Environments (OSCORE)", draft-ietf-core-object-security-16 (work in progress), March 2019.
- [I-D.ietf-core-oscore-groupcomm]
Tiloca, M., Selander, G., Palombini, F., and J. Park, "Group OSCORE - Secure Group Communication for CoAP", draft-ietf-core-oscore-groupcomm-05 (work in progress), July 2019.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://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, <<https://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, <<https://www.rfc-editor.org/info/rfc7252>>.
- [RFC7641] Hartke, K., "Observing Resources in the Constrained Application Protocol (CoAP)", RFC 7641, DOI 10.17487/RFC7641, September 2015, <<https://www.rfc-editor.org/info/rfc7641>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.

10.2. Informative References

- [I-D.ietf-ace-key-groupcomm-oscore]
Tiloca, M., Park, J., and F. Palombini, "Key Management for OSCORE Groups in ACE", draft-ietf-ace-key-groupcomm-oscore-02 (work in progress), July 2019.
- [I-D.ietf-ace-oauth-authz]
Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "Authentication and Authorization for Constrained Environments (ACE) using the OAuth 2.0 Framework (ACE-OAuth)", draft-ietf-ace-oauth-authz-24 (work in progress), March 2019.
- [I-D.ietf-core-coap-pubsub]
Koster, M., Keranen, A., and J. Jimenez, "Publish-Subscribe Broker for the Constrained Application Protocol (CoAP)", draft-ietf-core-coap-pubsub-08 (work in progress), March 2019.
- [I-D.ietf-core-resource-directory]
Shelby, Z., Koster, M., Bormann, C., Stok, P., and C. Amsuess, "CoRE Resource Directory", draft-ietf-core-resource-directory-22 (work in progress), July 2019.
- [I-D.tiloca-core-oscore-discovery]
Tiloca, M., Amsuess, C., and P. Stok, "Discovery of OSCORE Groups with the CoRE Resource Directory", draft-tiloca-core-oscore-discovery-03 (work in progress), July 2019.
- [RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <<https://www.rfc-editor.org/info/rfc6347>>.
- [RFC7390] Rahman, A., Ed. and E. Dijk, Ed., "Group Communication for the Constrained Application Protocol (CoAP)", RFC 7390, DOI 10.17487/RFC7390, October 2014, <<https://www.rfc-editor.org/info/rfc7390>>.

Acknowledgments

The authors sincerely thank John Mattsson, Ludwig Seitz and Goeran Selander for their comments and feedback.

The work on this document has been partly supported by VINNOVA and the Celtic-Next project CRITISEC.

Authors' Addresses

Marco Tiloca
RISE AB
Isafjordsgatan 22
Kista SE-16440 Stockholm
Sweden

Email: marco.tiloca@ri.se

Rikard Hoeglund
RISE AB
Isafjordsgatan 22
Kista SE-16440 Stockholm
Sweden

Email: rikard.hoglund@ri.se

Christian Amsuess
Hollandstr. 12/4
Vienna 1020
Austria

Email: christian@amsuess.com

Francesca Palombini
Ericsson AB
Torshamnsgatan 23
Kista SE-16440 Stockholm
Sweden

Email: francesca.palombini@ericsson.com

CoRE Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 6, 2020

M. Tiloca
RISE AB
C. Amsuess

P. van der Stok
Consultant
July 05, 2019

Discovery of OSCORE Groups with the CoRE Resource Directory
draft-tiloca-core-oscore-discovery-03

Abstract

Group communication over the Constrained Application Protocol (CoAP) can be secured by means of Object Security for Constrained RESTful Environments (OSCORE). At deployment time, devices may not know the exact OSCORE groups to join, the respective Group Manager, or other information required to perform the joining process. This document describes how a CoAP endpoint can use the CoRE Resource Directory to discover OSCORE groups and acquire information to join them through the respective Group Manager. A given OSCORE group may protect multiple application groups, which are separately announced in the Resource Directory as sets of endpoints sharing a pool of resources. This approach is consistent with, but not limited to, the joining of OSCORE groups based on the ACE framework for Authentication and Authorization in constrained environments.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on January 6, 2020.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

| | |
|---|----|
| 1. Introduction | 2 |
| 1.1. Terminology | 4 |
| 2. Registration Resource for Group Managers | 5 |
| 3. Registration of Group Manager Endpoints | 6 |
| 4. Addition and Update of OSCORE Groups | 8 |
| 5. Discovery of OSCORE Groups | 9 |
| 5.1. Discovery Example #1 | 10 |
| 5.2. Discovery Example #2 | 11 |
| 6. Use Case Example With Full Discovery | 12 |
| 7. Security Considerations | 17 |
| 8. IANA Considerations | 17 |
| 8.1. Resource Types | 17 |
| 9. References | 17 |
| 9.1. Normative References | 17 |
| 9.2. Informative References | 18 |
| Acknowledgments | 19 |
| Authors' Addresses | 19 |

1. Introduction

A set of CoAP endpoints constitutes an application group by sharing a common pool of resources. The members of an application group may be members of a given security group, by sharing a common set of keying material to secure group communication.

The Constrained Application Protocol (CoAP) [RFC7252] supports group communication over IP multicast [RFC7390][I-D.dijk-core-groupcomm-bis] to improve efficiency and latency of communication and reduce bandwidth requirements. The document Object Security for Constrained RESTful Environments (OSCORE) [I-D.ietf-core-object-security] describes how to achieve

end-to-end security for CoAP messages through CBOR Object Signing and Encryption (COSE) [RFC8152].

In particular, [I-D.ietf-core-oscore-groupcomm] specifies how OSCORE protects CoAP messages in group communication contexts, so enabling OSCORE groups as security groups. Typically, one application group relies on exactly one OSCORE group, while a same OSCORE group may be used by multiple application groups at the same time.

A CoAP endpoint joins an OSCORE group via a Group Manager (GM), in order to get the necessary group keying material. As in [I-D.ietf-ace-key-groupcomm-oscore], the joining process can be based on the ACE framework for Authentication and Authorization in constrained environments [I-D.ietf-ace-oauth-authz], with the joining endpoint and the GM acting as ACE Client and Resource Server, respectively. That is, the joining endpoint accesses the join resource associated with the OSCORE group of interest and exported by the GM.

Typically, devices are equipped with a static X509 IDevID certificate installed at manufacturing time. This certificate is used at deployment time during an enrollment process that provides the device with an Operational Certificate, possibly updated during the device lifetime. In the presence of secure group communication for CoAP, such an Operational Certificate may be accompanied by information required to join OSCORE groups. This especially includes a reference to the join resources to access at the respective GMs.

However, it is usually impossible to provide such precise information to freshly deployed devices as part of their (early) Operational Certificate. This can be due to a number of reasons: (1) the OSCORE group(s) to join and the responsible GM(s) are generally unknown at manufacturing time; (2) an OSCORE group of interest is created, or the responsible GM is deployed, only after the device is enrolled and fully operative in the network; and (3) information related to existing OSCORE groups or to their GMs has been changed. This requires a method for CoAP endpoints to dynamically discover OSCORE groups and their GM, and to retrieve valid information about deployed groups.

This specification describes how CoAP endpoints can use the CoRE Resource Directory (RD) [I-D.ietf-core-resource-directory] for discovering an OSCORE group and retrieving the information required to join that group through a given GM. In principle, the GM registers as an endpoint with the RD. The corresponding registration resource includes one link for each OSCORE group under that GM, specifying the path to the related join resource.

More information about the OSCORE group is stored in the target attributes of the respective link. This especially includes the identifiers of the application groups which use that OSCORE group. This enables a lookup of those application groups at the Resource Directory, where they are separately announced by a Commissioning Tool (see Appendix A of [I-D.ietf-core-resource-directory]).

When querying the RD for OSCORE groups, a CoAP endpoint can further benefit of the CoAP Observe Option [RFC7641]. This enables the reception of notifications about the creation of new OSCORE groups or the updates concerning existing groups. Thus, it facilitates the early deployment of CoAP endpoints, i.e. even before the GM is deployed and OSCORE groups are created.

The approach in this document is consistent with, but not limited to, the joining of OSCORE groups in [I-D.ietf-ace-key-groupcomm-oscore].

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 BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

This specification requires readers to be familiar with the terms and concepts discussed in [I-D.ietf-core-resource-directory] and [RFC6690]. Readers should also be familiar with the terms and concepts discussed in [RFC7252], [I-D.ietf-core-oscore-groupcomm] and [I-D.ietf-ace-key-groupcomm-oscore].

Terminology for constrained environments, such as "constrained device" and "constrained-node network", is defined in [RFC7228].

This document also refers to the following terminology.

- o OSCORE group: a set of CoAP endpoints that share one OSCORE Common Security Context to protect group communication as described in [I-D.ietf-core-oscore-groupcomm]. Consequently, an OSCORE group acts as security group for all its members.
- o Application group: a set of CoAP endpoints that share a set of common resources. Application groups are announced in the RD by a Commissioning Tool, according to the RD-Groups usage pattern (see Appendix A of [I-D.ietf-core-resource-directory]). An application group can be associated with a single OSCORE group, while multiple application groups can use the same OSCORE group. Application groups share resources by definition. Any two application groups

associated to the same OSCORE group do not share any same resource.

- o Zeroed-epoch Group ID: this refers to the Group ID of an OSCORE group as stored in the RD. The structure of such a stored Group ID is as per Appendix C of [I-D.ietf-core-oscore-groupcomm], with the "Group Epoch" part immutable and set to zero.

2. Registration Resource for Group Managers

With reference to Figure 3 of [I-D.ietf-core-resource-directory], a Group Manager (GM) registers as an endpoint with the CoRE Resource Directory (RD). The registration includes the links to the "join resources" located at the GM, and associated to the OSCORE groups administrated by that GM.

In particular, each link to a join resource includes:

- o "target": URI of the join resource at the GM.
- o target attributes, including:
 - * Resource Type (rt) with the value "core.osc.j" defined in Section 8.1 of this specification.
 - * The zeroed-epoch Group ID of the OSCORE group.
 - * One target attribute for each application group associated with the OSCORE group, specifying the name of that application group.
 - * The algorithm used to countersign messages in the OSCORE group.
 - * The elliptic curve (if applicable) for the algorithm used to countersign messages in the OSCORE group.
 - * The key type of countersignature keys used to countersign messages in the OSCORE group.
 - * The encoding of public keys used in the OSCORE group.
 - * The AEAD algorithm used in the OSCORE group.
 - * The HKDF algorithm used in the OSCORE group.

3. Registration of Group Manager Endpoints

During deployment, a GM finds the RD as described in Section 4 of [I-D.ietf-core-resource-directory]. Afterwards, the GM registers as an endpoint with the RD, as described in Section 5 of [I-D.ietf-core-resource-directory].

When doing so, the GM MUST also register all the join resources it has at that point in time, i.e. one for each of its OSCORE groups.

For each registered join resource, the GM MUST specify the following parameters in the payload of the registration request.

- o 'rt' = "core.osc.j" (see Section 8.1).
- o 'oscore-gid', specifying the zeroed-epoch Group ID of the OSCORE group of interest. This parameter MUST specify a single value.
- o 'app-gp', specifying the name(s) of the application group(s) associated to the OSCORE group of interest. This parameter MAY be included multiple times, and each occurrence MUST specify the name of one application group. A same application group MUST NOT be specified multiple times.

Also, for each registered join resource, the GM MAY specify the following parameters in the payload of the registration request.

- o 'cs_alg', specifying the algorithm used to countersign messages in the OSCORE group. If present, this parameter MUST specify a single value, which is taken from the 'Name' column of the "COSE Algorithms" Registry defined in [RFC8152].
- o 'cs_crv', specifying the elliptic curve (if applicable) for the algorithm used to countersign messages in the OSCORE group. If present, this parameter MUST specify a single value, which is taken from the 'Name' column of the "COSE Elliptic Curve" Registry defined in [RFC8152].
- o 'cs_kty', specifying the key type of countersignature keys used to countersign messages in the OSCORE group. If present, this parameter MUST specify a single value, which is taken from the 'Name' column of the "COSE Key Types" Registry defined in [RFC8152].
- o 'cs_kenc', specifying the encoding of the public keys used in the OSCORE group. If present, this parameter MUST specify a single value, which is taken from the 'Name' column of Figure 2 in [I-D.ietf-ace-key-groupcomm-oscore], as registered in the "ACE

Public Key Encoding" Registry defined in [I-D.ietf-ace-key-groupcomm].

- o 'alg', specifying the AEAD algorithm used in the OSCORE group. If present, this parameter MUST specify a single value, which is taken from the 'Name' column of the "COSE Algorithms" Registry defined in [RFC8152].
- o 'hkdf', specifying the HKDF algorithm used in the OSCORE group. If present, this parameter MUST specify a single value, which is taken from the 'Name' column of the "COSE Algorithms" Registry defined in [RFC8152].

A CoAP endpoint that queries the RD to discover OSCORE groups and their join resource to access (see Section 5) would benefit from the link target attributes above as follows.

- o The values of 'cs_alg', 'cs_crv', 'cs_kty' and 'cs_kenc' related to a join resource provide an early knowledge of the format and encoding of public keys used in the OSCORE group. Thus, the CoAP endpoint does not need to ask the GM for this information as a preliminary step before the join process, or to perform a trial-and-error exchange with the GM. Hence, the CoAP endpoint is able to provide the GM with its own public key in the correct expected format and encoding at the very first step of the join process.
- o The values of 'cs_alg', 'alg' and 'hkdf' related to a join resource provide an early knowledge of the algorithms used in the OSCORE group. Thus, the CoAP endpoint is able to decide whether to actually proceed with the join process, depending on its support for the indicated algorithms.

The GM SHOULD NOT use the Simple Registration approach described in Section 5.1 of [I-D.ietf-core-resource-directory].

The example below shows a GM with endpoint name "gm1" and address 2001:db8::ab that registers with the RD. The GM specifies the value of the 'oscore-gid' parameter for accessing the OSCORE group with zeroed-epoch Group ID "feedca570000" and used by the application group with name "group1" specified with the value of the 'app-gp' parameter. The countersignature algorithm used in the OSCORE group is EdDSA, with elliptic curve Ed25519 and keys of type OKP. Public keys used in the OSCORE group are encoded as COSE Keys [RFC8152].

Request: GM -> RD


```
Req: POST coap://rd.example.com/rd?ep=gml
Content-Format: 40
Payload:
</join/feedca570000>;ct=41;rt="core.osc.j";oscore-gid="feedca570000";
    app-gp="group1";cs_alg="EdDSA";cs_crv="Ed25519";
    cs_kty="OKP";cs_kenc="COSE_Key"
```

Response: RD -> GM

```
Res: 2.01 Created
Location-Path: /rd/4521
```

4. Addition and Update of OSCORE Groups

The GM is responsible to refresh the registration of all its join resources in the RD. This means that the GM has to update the registration within its lifetime as per Section 5.3.1 of [I-D.ietf-core-resource-directory], and has to change the content of the registration when a join resource is added/removed or if its target attributes have to be changed, such as in the following cases.

- o The GM creates a new OSCORE group and starts exporting the related join resource.
- o The GM dismisses an OSCORE group and stops exporting the related join resource.
- o Information related to an existing OSCORE group changes, e.g. the list of associated application groups.

To perform an update of its registrations, the GM can re-register with the RD and fully specify all links to its join resources with their target attributes.

The example below shows how the GM from Section 3 re-registers with the RD. When doing so, it specifies:

- o The same previous join resource associated to the OSCORE group with zeroed-epoch Group ID "feedca570000".
- o An additional join resource associated to the OSCORE group with zeroed-epoch Group ID "ech0ech00000" and used by the application group "group2".
- o A third join resource associated with the OSCORE group with zeroed-epoch Group ID "abcdef120000" and used by two application groups, namely "group3" and "group4".

Request: GM -> RD

Req: POST coap://rd.example.com/rd?ep=gml

Content-Format: 40

Payload:

```
</join/feedca570000>;ct=41;rt="core.osc.j";oscore-gid="feedca570000";
    app-gp="group1";cs_alg="EdDSA";cs_crv="Ed25519";
    cs_kty="OKP";cs_kenc="COSE_Key",
</join/ech0ech00000>;ct=41;rt="core.osc.j";oscore-gid="ech0ech00000";
    app-gp="group2";cs_alg="EdDSA";cs_crv="Ed25519";
    cs_kty="OKP";cs_kenc="COSE_Key",
</join/abcdef120000>;ct=41;rt="core.osc.j";oscore-gid="abcdef120000";
    app-gp="group3";app-gp="group4";cs_alg="EdDSA";
    cs_crv="Ed25519";cs_kty="OKP";cs_kenc="COSE_Key"
```

Response: RD -> GM

Res: 2.04 Changed

Location-Path: /rd/4521

Alternatively, the GM can perform a PATCH/iPATCH [RFC8132] request to the RD, as per Section 5.3.3 of [I-D.ietf-core-resource-directory]. This requires new media-types to be defined in future standards, to apply a link-format document as a patch to an existing stored document.

5. Discovery of OSCORE Groups

A CoAP endpoint that wants to join an OSCORE group, hereafter called the joining node, might not have all the necessary information at deployment time. Also, it might want to know about possible new OSCORE groups created afterwards by the respective Group Managers.

To this end, the joining node can perform a resource lookup at the RD as per Section 6.1 of [I-D.ietf-core-resource-directory], to retrieve the missing pieces of information needed to join the OSCORE group(s) of interest. The joining node can find the RD as described in Section 4 of [I-D.ietf-core-resource-directory].

The joining node MUST use the following parameter values for the lookup filtering.

- o 'rt' = "core.osc.j" (see Section 8.1).

The joining node MAY additionally consider the following parameters for the lookup filtering, depending on the information it has already available.

- o 'oscore-gid', specifying the zeroed-epoch Group ID of the OSCORE group of interest. This parameter MUST specify a single value.
- o 'ep', specifying the registered endpoint of the GM.
- o 'app-gp', specifying the name(s) of the application group(s) associated with the OSCORE group of interest. This parameter MAY be included multiple times, and each occurrence MUST specify the name of one application group. An application group MUST be specified only once.

5.1. Discovery Example #1

Consistently with the examples in Section 3 and Section 4, the example below considers a joining node that wants to join the OSCORE group associated with the application group "group1", but that does not know the zeroed-epoch Group ID of the OSCORE group, the responsible GM and the join resource to access.

Request: Joining node -> RD

Req: GET coap://rd.example.com/rd-lookup/res
?rt=core.osc.j&app-gp=group1

Response: RD -> Joining node

Res: 2.05 Content

Payload:

```
<coap://[2001:db8::ab]/join/feedca570000>;rt="core.osc.j";  
  oscore-gid="feedca570000";app-gp="group1";  
  cs_alg="EdDSA";cs_crv="Ed25519";cs_kty="OKP";  
  cs_kenc="COSE_Key";anchor="coap://[2001:db8::ab]"
```

To retrieve the multicast IP address used in "group1", the joining node performs an endpoint lookup as shown below. The following assumes that the application group "group1" had been previously registered as per Appendix A of [I-D.ietf-core-resource-directory], with ff35:30:2001:db8::23 as associated multicast IP address.

Request: Joining node -> RD

Req: GET coap://rd.example.com/rd-lookup/ep
?et=core.rd-group&ep=group1

Response: RD -> Joining node

```
Res: 2.05 Content
Payload:
</rd/501>;ep="group1";et="core.rd-group";
      base="coap://[ff35:30:2001:db8::23]"
```

5.2. Discovery Example #2

Consistently with the examples in Section 3 and Section 4, the example below considers a joining node that wants to join the OSCORE group with zeroed-epoch Group ID "feedca570000", but that does not know the responsible GM, the join resource to access, and the associated application groups.

The example also shows how the joining node uses CoAP observation [RFC7641], in order to be notified of possible changes in the join resource's target attributes. This is also useful to handle the case where the OSCORE group of interest has not been created yet, so that the joining node can receive the requested information when it becomes available.

Request: Joining node -> RD

```
Req: GET coap://rd.example.com/rd-lookup/res
      ?rt=osc.j&oscore-gid=feedca570000
Observe: 0
```

Response: RD -> Joining node

```
Res: 2.05 Content
Observe: 24
Payload:
<coap://[2001:db8::ab]/join/feedca570000>;rt="osc.j";
  oscore-gid="feedca570000";app-gp="group1";
  cs_alg="EdDSA";cs_crv="Ed25519";cs_kty="OKP";
  cs_kenc="COSE_Key";anchor="coap://[2001:db8::ab]"
```

Depending on the search criteria, the joining node performing the resource lookup can get large responses. This can happen, for instance, when the lookup request targets all the join resources at a specified GM, or all the join resources of all the registered GMs, as in the example below.

Request: Joining node -> RD

```
Req: GET coap://rd.example.com/rd-lookup/res?rt=osc.j
```

Response: RD -> Joining node

Res: 2.05 Content

Payload:

```
<coap://[2001:db8::ab]/join/feedca570000>;rt="osc.j";
  oscore-gid="feedca570000";app-gp="group1";
  cs_alg="EdDSA";cs_crv="Ed25519";cs_kty="OKP";
  cs_kenc="COSE_Key";anchor="coap://[2001:db8::ab]",
<coap://[2001:db8::ab]/join/ech0ech00000>;rt="osc.j";
  oscore-gid="ech0ech00000";app-gp="group2";
  cs_alg="EdDSA";cs_crv="Ed25519";cs_kty="OKP";
  cs_kenc="COSE_Key";anchor="coap://[2001:db8::ab]",
<coap://[2001:db8::ab]/join/abcdef120000>;rt="osc.j";
  oscore-gid="abcdef120000";app-gp="group3";
  app-gp="group4";cs_alg="EdDSA";cs_crv="Ed25519";
  cs_kty="OKP";cs_kenc="COSE_Key";anchor="coap://[2001:db8::ab]"
```

Therefore, it is RECOMMENDED that a joining node which performs a resource lookup with the CoAP Observe option specifies the value of the parameter 'oscore-gid' in its GET request sent to the RD.

6. Use Case Example With Full Discovery

In this section, the discovery of security groups is described to support the installation process of a lighting installation in an office building. The described process is a simplified version of one of many processes.

Assume the existence of four luminaires that are members of two application groups. In the first application group, the four luminaires receive presence messages and light intensity messages from sensors or their proxy. In the second application group, the four luminaires and several other pieces of equipment receive building state schedules.

Each of the two application groups is associated to a different security group and uses its own dedicated multicast IP address.

The Fairhair Alliance describes how a new device is accepted and commissioned in the network [Fairhair], by means of its certificate stored during the manufacturing process. When commissioning the new device in the installation network, the new device gets a new identity defined by a newly allocated certificate, following the BRSKI specification.

Section 7.3 of [I-D.ietf-core-resource-directory] describes how the Commissioning Tool (CT) assigns an endpoint name based on the CN field, (CN=ACME) and the serial number of the certificate (serial number = 123x, with $3 < x < 8$). Corresponding ep-names ACME-1234, ACME-1235, ACME-1236 and ACME-1237 are also assumed.

It is common practice that locations in the building are specified according to a coordinate system. After the acceptance of the luminaires into the installation network, the coordinate of each device is communicated to the CT. This can be done manually or automatically.

The mapping between location and ep-name is calculated by the CT. For instance, on the basis of grouping criteria, the CT assigns: i) group "grp_R2-4-015" to the four luminaires; and ii) group "grp_schedule" to all schedule requiring devices. Also, the device with ep name ACME-123x has been assigned IP address: [2001:db8:4::x]. The RD is assigned IP address: [2001:db8:4:ff]. The used multicast addresses are: [ff05::5:1] and [ff05::5:2].

*** **

The CT defines the application group "grp_R2-4-015", with resource /light and base address [ff05::5:1], as follows.

Request: CT -> RD

Req: POST coap://[2001:db8:4::ff]/rd
?ep=grp_R2-4-015&et=core.rd-group&base=coap://[ff05::5:1]
Payload:
</light>;rt="oic.d.light"

Response: RD -> CT

Res: 2.01 Created
Location-Path: /rd/501

Also, the CT defines a second application group "grp_schedule", with resource /schedule and base address [ff05::5:2], as follows.

Request: CT -> RD

Req: POST coap://[2001:db8:4::ff]/rd
?ep=grp_schedule&et=core.rd-group&base=coap://[ff05::5:2]
Payload:
</schedule>;rt="oic.r.time.period"

Response: RD -> CT

Res: 2.01 Created
Location-Path: /rd/502

*** **

Consecutively, the CT registers the four devices in the RD (IP address: 2001:db8:4::ff), with their endpoint names and application groups.

For group "grp_R2-4-015", four endpoints are specified as follows, with x = 4, 5, 6, 7.

Request: CT -> RD

```
Req: POST coap://[2001:db8:4::ff]/rd
      ?ep=ACME-123x&base=coap://[2001:db8:4::x]&app-gp=grp_R2-4-015
Payload:
</light>;rt="oic.d.light"
```

Response: RD -> CT

```
Res: 2.01 Created
Location-Path: /rd/452x
```

For group "grp_schedule", four other endpoints are specified as follows, with x = 4, 5, 6, 7.

Request: CT -> RD

```
Req: POST coap://[2001:db8:4::ff]/rd
      ?ep=ACME-123x&base=coap://[2001:db8:4::x]&app-gp=grp_schedule
Payload:
</schedule>;rt="oic.r.time.period"
```

Response: RD -> CT

```
Res: 2.01 Created
Location-Path: /rd/456x
```

*** **

Finally, the CT defines the corresponding security groups. In particular, assuming a Group Manager responsible for both security groups and with address [2001:db8::ab], the CT specifies:

Request: CT -> RD

```
Req: POST coap://[2001:db8:4::ff]/rd?ep=gml&base=coap://[2001:db8::ab]
Payload:
</join/feedca570000>;ct=41;rt="core.osc.j";
      oscore-gid="feedca570000";app-gp="grp_R2-4-015",
</join/feedsc590000>;ct=41;rt="core.osc.j";
      oscore-gid="feedsc590000";app-gp="grp_schedule"
```

Response: RD -> CT

Res: 2.01 Created
Location-Path: /rd/4521

*** **

The device with IP address [2001:db8:4::x] can consequently learn the groups to which it belongs. In particular, it first does an endpoint lookup to the RD to learn the application groups to which it belongs.

Request: Joining node -> RD

Req: GET coap://[2001:db8:4::ff]/rd-lookup/ep
?base=coap://[2001:db8:4::x]

Response: RD -> Joining node

Res: 2.05 Content

Payload:

```
<rd/452x>;base=coap://[2001:db8:4::x]&ep=ACME-123x&\n    app-gp=grp_R2-4-015,\n<rd/456x>;base=coap://[2001:db8:4::x]&ep=ACME-123x&\n    app-gp=grp_schedule
```

To retrieve the multicast IP address used in "grp_R2-4-015", the device performs an endpoint lookup as shown below.

Request: Joining node -> RD

Req: GET coap://[2001:db8:4::ff]/rd-lookup/ep
?et=core.rd-group&ep=grp_R2-4-015

Response: RD -> Joining node

Res: 2.05 Content

Payload:

```
</rd/501>;ep="grp_R2-4-015";et="core.rd-group";\n    base="coap://[ff05::5:1]"
```

Similarly, to retrieve the multicast IP address used in "grp_schedule", the device performs an endpoint lookup as shown below.

Request: Joining node -> RD

Req: GET coap://[2001:db8:4::ff]/rd-lookup/ep
?et=core.rd-group&ep=grp_schedule

Response: RD -> Joining node

Res: 2.05 Content

Payload:

```
</rd/502>;ep="grp_schedule";et="core.rd-group";
    base="coap://[ff05::5:2]"
```

*** **

Having learnt the application groups to which the device belongs, the device learns the security groups to which it belongs. In particular, it does the following for app-gp="grp_R2-4-015".

Request: Joining node -> RD

```
Req: GET coap://[2001:db8:4::ff]/rd-lookup/res
    ?rt=core.osc.j&app-gp=grp_R2-4-015
```

Response: RD -> Joining Node

Res: 2.05 Content

Payload:

```
<coap://[2001:db8::ab]/join/feedca570000>;
    rt="core.osc.j";oscore-gid="feedca570000";
    app-gp="grp_R2-4-015";anchor="coap://[2001:db8::ab]"
```

Similarly, the device does the following for app-gp="grp_schedule".

```
Req: GET coap://[2001:db8:4::ff]/rd-lookup/res
    ?rt=core.osc.j&app-gp=grp_schedule
```

Response: RD -> Joining Node

Res: 2.05 Content

Payload:

```
<coap://[2001:db8::ab]/join/feedsc590000>;
    rt="core.osc.j";oscore-gid="feedsc590000";
    app-gp="grp_schedule";anchor="coap://[2001:db8::ab]"
```

*** **

After this last discovery step, the device can ask permission to join the security groups, and effectively join them through the Group Manager, e.g. according to [I-D.ietf-ace-key-groupcomm-oscore].

7. Security Considerations

The security considerations as described in Section 8 of [I-D.ietf-core-resource-directory] apply here as well.

8. IANA Considerations

This document has the following actions for IANA.

8.1. Resource Types

IANA is asked to enter the following value into the Resource Type (rt=) Link Target Attribute Values subregistry within the Constrained Restful Environments (CoRE) Parameters registry defined in [RFC6690].

| Value | Description | Reference |
|------------|--|-------------------|
| core.osc.j | Join resource of an OSCORE Group Manager | [[this document]] |

9. References

9.1. Normative References

- [I-D.ietf-ace-key-groupcomm]
Palombini, F. and M. Tiloca, "Key Provisioning for Group Communication using ACE", draft-ietf-ace-key-groupcomm-02 (work in progress), July 2019.
- [I-D.ietf-ace-key-groupcomm-oscore]
Tiloca, M., Park, J., and F. Palombini, "Key Management for OSCORE Groups in ACE", draft-ietf-ace-key-groupcomm-oscore-02 (work in progress), July 2019.
- [I-D.ietf-core-oscore-groupcomm]
Tiloca, M., Selander, G., Palombini, F., and J. Park, "Group OSCORE - Secure Group Communication for CoAP", draft-ietf-core-oscore-groupcomm-05 (work in progress), July 2019.
- [I-D.ietf-core-resource-directory]
Shelby, Z., Koster, M., Bormann, C., Stok, P., and C. Amsuess, "CoRE Resource Directory", draft-ietf-core-resource-directory-22 (work in progress), July 2019.

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <<https://www.rfc-editor.org/info/rfc6690>>.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<https://www.rfc-editor.org/info/rfc7252>>.
- [RFC8152] Schaad, J., "CBOR Object Signing and Encryption (COSE)", RFC 8152, DOI 10.17487/RFC8152, July 2017, <<https://www.rfc-editor.org/info/rfc8152>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.

9.2. Informative References

- [Fairhair] FairHair Alliance, "Security Architecture for the Internet of Things (IoT) in Commercial Buildings", White Paper, ed. Piotr Polak, March 2018, <https://www.fairhair-alliance.org/data/downloadables/1/9/fairhair_security_wp_march-2018.pdf>.
- [I-D.dijk-core-groupcomm-bis] Dijk, E., Wang, C., and M. Tiloca, "Group Communication for the Constrained Application Protocol (CoAP)", draft-dijk-core-groupcomm-bis-00 (work in progress), March 2019.
- [I-D.ietf-ace-oauth-authz] Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "Authentication and Authorization for Constrained Environments (ACE) using the OAuth 2.0 Framework (ACE-OAuth)", draft-ietf-ace-oauth-authz-24 (work in progress), March 2019.
- [I-D.ietf-core-object-security] Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security for Constrained RESTful Environments (OSCORE)", draft-ietf-core-object-security-16 (work in progress), March 2019.

- [RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, DOI 10.17487/RFC7228, May 2014, <<https://www.rfc-editor.org/info/rfc7228>>.
- [RFC7390] Rahman, A., Ed. and E. Dijk, Ed., "Group Communication for the Constrained Application Protocol (CoAP)", RFC 7390, DOI 10.17487/RFC7390, October 2014, <<https://www.rfc-editor.org/info/rfc7390>>.
- [RFC7641] Hartke, K., "Observing Resources in the Constrained Application Protocol (CoAP)", RFC 7641, DOI 10.17487/RFC7641, September 2015, <<https://www.rfc-editor.org/info/rfc7641>>.
- [RFC8132] van der Stok, P., Bormann, C., and A. Sehgal, "PATCH and FETCH Methods for the Constrained Application Protocol (CoAP)", RFC 8132, DOI 10.17487/RFC8132, April 2017, <<https://www.rfc-editor.org/info/rfc8132>>.

Acknowledgments

The authors sincerely thank Carsten Bormann, Francesca Palombini, Dave Robin and Jim Schaad for their comments and feedback.

The work on this document has been partly supported by VINNOVA and the Celtic-Next project CRITISEC, and by the EIT-Digital High Impact Initiative ACTIVE.

Authors' Addresses

Marco Tiloca
RISE AB
Isafjordsgatan 22
Kista SE-16440 Stockholm
Sweden

Email: marco.tiloca@ri.se

Christian Amsuess
Hollandstr. 12/4
Vienna 1020
Austria

Email: christian@amsuess.com

Peter van der Stok
Consultant

Phone: +31-492474673 (Netherlands), +33-966015248 (France)
Email: consultancy@vanderstok.org
URI: www.vanderstok.org

CORE
Internet-Draft
Updates: 8428 (if approved)
Intended status: Standards Track
Expires: December 20, 2019

H. Tschofenig
Arm Ltd.
June 18, 2019

Introducing the Local Base Name in SenML
draft-tschofenig-core-senml-lbn-00

Abstract

The Sensor Measurement Lists (SenML) specification defines a format for representing simple sensor measurements and device parameters. This specification defines a new label to relax the requirement for global identification of every measurement.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on December 20, 2019.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

| | |
|--|---|
| 1. Introduction | 2 |
| 2. Terminology | 4 |
| 3. Local Base Name SenML Structure and Semantics | 4 |
| 4. CDDL | 4 |
| 5. Security Considerations | 4 |
| 6. IANA Considerations | 4 |
| 7. Acknowledgements | 5 |
| 8. References | 5 |
| 8.1. Normative References | 5 |
| 8.2. Informative References | 5 |
| Author's Address | 6 |

1. Introduction

The Sensor Measurement Lists (SenML) specification, see RFC 8428 [RFC8428], defines a format for representing simple sensor measurements and device parameters.

Ideally, sensor readings used in the Internet of Things environment should be as small as possible. For this reason the specification also defines an encoding of these sensor measurements and device parameters in CBOR (on top of other serialization formats).

A design decision in SenML was, however, that each measurement transmitted over the network is self-contained and contains information that uniquely identifies and differentiates the sensor from all others – not only locally on the device but globally.

This is accomplished by the combination of two fields, namely the 'Name' and the 'Base Name' values. The specification requires the concatenation of the Name and the Base Name values to yield the name of the sensor and recommends that the concatenated names be represented as URIs or URNs.

Figure 1 is an example taken from RFC 8428.

```
[
  {"bn":"urn:dev:ow:10e2073a01080063:", "n":"temp", "u":"Cel", "v":23.1},
  {"n":"label", "vs":"Machine Room"},
  {"n":"open", "vb":false},
  {"n":"nfc-reader", "vd":"aGkgCg"}
]
```

Figure 1: SenML Example Measurement with Base Name value

The global identification of every measurement as it is traveling through the network and through different systems has its use case and allows easy identification of the source and enables correlation.

Unfortunately, it also has drawbacks:

- o The unique identification of the sensor adds a substantial overhead, particularly when the sensor identification is verbose. Deployed systems, for example, make use of RFC 4122 [RFC4122] Type 5 Universally Unique Identifier (UUIDs).
- o The global identification of every measurements is often unnecessary when the SenML is used as a mechanism to represent data for device-to-cloud communication or cloud-to-gateway communication where data is subsequently processed, aggregated or otherwise modified. In such systems, the identification of the sensor and the device has its origin in the security context rather than in the SenML contained measurement. LwM2M [LWM2M] is an example of such an architecture.
- o Finally, there are privacy implications of globally identifying each measurements and some deployments may prefer better privacy protection over ease of correlation.

This specification therefore updates RFC 8428 and defines a new Local Base Name value (lbn) that can be used instead of the Base Name value defined in RFC 8428.

Figure 2 shows an example based on LwM2M use of SenML.

```
[
  {"lbn":"/3303/", "n":"0/5700/", "v":25.2},
  {"n":"/1/5700/", "v":5}
]
```

Figure 2: SenML Example Measurement with Local Base Name value

Note: In the LwM2M data model objects, object instances, and resources are encoded as numerical values. The value of '3303' refers to the Temperature object, '0' and '1' to the two instances of the resource '5700' (Sensor Value). Hence, this measurement indicates that the two temperature sensors expose their sensor readings.

2. 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 BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

3. Local Base Name SenML Structure and Semantics

This specification defines one new label, the Local Base Name value, which is used instead of the Base Name value. For practical purposes the Local Base Name / Name concatenation does not need to be a URN or a URI because existing data models used in the IoT space do not all make use of URIs/URNs.

| Name | Label | CBOR Label | JSON Type | XML Type |
|-----------------|-------|------------|-----------|----------|
| Local Base Name | lbn | 8 | String | string |

Figure 3: Local Base Name SenML Label

4. CDDL

This document defines a new value to be added to the CDDL defined in Section 11 of RFC 8428.

The new key-value-pair is

```
lbn => tstr          ; Local Base Name
```

5. Security Considerations

This document inherits the security properties of RFC 8428 but improves its privacy features by removing the unique identification of the sensor when the Local Base Name value is used instead of the Name / Base Name combination.

6. IANA Considerations

IANA is asked to register the following new entry in the SenML Labels Registry.

| Name | Label | CL | JSON Type | XML Type | EI | Reference |
|-----------------|-------|----|-----------|----------|----|--------------|
| Local Base Name | lbn | 8 | String | string | a | [[This RFC]] |

Note that CL = CBOR Label and EI = EXI ID.

7. Acknowledgements

The author would like to thank the OMA Device Management and Service Enablement working group for their discussion and their input. In particular, I would to thank Ari Keranen, David Navarro, Mojan Mohajer and Alan Soloway.

8. References

8.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8428] Jennings, C., Shelby, Z., Arkko, J., Keranen, A., and C. Bormann, "Sensor Measurement Lists (SenML)", RFC 8428, DOI 10.17487/RFC8428, August 2018, <<https://www.rfc-editor.org/info/rfc8428>>.

8.2. Informative References

- [LWM2M] Open Mobile Alliance, "Lightweight Machine to Machine Technical Specification Version 1.0", February 2017, <http://www.openmobilealliance.org/release/LightweightM2M/V1_0-20170208-A/OMA-TS-LightweightM2M-V1_0-20170208-A.pdf>.
- [RFC4122] Leach, P., Mealling, M., and R. Salz, "A Universally Unique IDentifier (UUID) URN Namespace", RFC 4122, DOI 10.17487/RFC4122, July 2005, <<https://www.rfc-editor.org/info/rfc4122>>.

Author's Address

Hannes Tschofenig
Arm Ltd.
110 Fulbourn Rd
Cambridge CB1 9NJ
UK

Email: Hannes.tschofenig@arm.com
URI: <http://www.tschofenig.priv.at>

Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: January 9, 2020

M. Veillette, Ed.
Trilliant Networks Inc.
I. Petrov, Ed.
Acklio
July 08, 2019

Constrained YANG Module Library
draft-veillette-core-yang-library-05

Abstract

This document describes a constrained version of the YANG library that provides information about the YANG modules, datastores, and datastore schemas used by a constrained network management server (e.g., a CORECONF server).

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on January 9, 2020.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

| | |
|---|----|
| 1. Introduction | 2 |
| 2. Terminology and Notation | 2 |
| 3. Overview | 3 |
| 3.1. Tree diagram | 3 |
| 3.2. Major differences between ietf-constrained-yang-library and ietf-yang-library | 4 |
| 4. YANG Module "ietf-constrained-yang-library" | 5 |
| 5. IANA Considerations | 13 |
| 5.1. YANG Module Registry | 13 |
| 6. Security Considerations | 13 |
| 7. Acknowledgments | 14 |
| 8. References | 14 |
| 8.1. Normative References | 14 |
| 8.2. Informative References | 14 |
| Authors' Addresses | 15 |

1. Introduction

There is a need for a standard mechanism to expose which YANG modules, datastores and datastore schemas are in use by a constrained network management server. This document defines the YANG module 'ietf-constrained-yang-library' that provides this information.

YANG module 'ietf-constrained-yang-library' shares the same data model and objectives as 'ietf-yang-library', only datatypes and mandatory requirements have been updated to minimize its size to allow its implementation by Constrained Nodes and/or Constrained Networks as defined by [RFC7228]. To review the list of objectives and proposed data model, please refer to [RFC8525] section 2 and 3.

2. Terminology and Notation

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 BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

The following terms are defined in [RFC7950]: client, deviation, feature, module, submodule and server.

The following term is defined in [I-D.ietf-core-sid]: YANG Schema Item Identifier (SID).

The following terms are defined in [RFC8525]: YANG library and YANG library checksum.

3. Overview

The conceptual model of the YANG library is depicted in Figure 1.

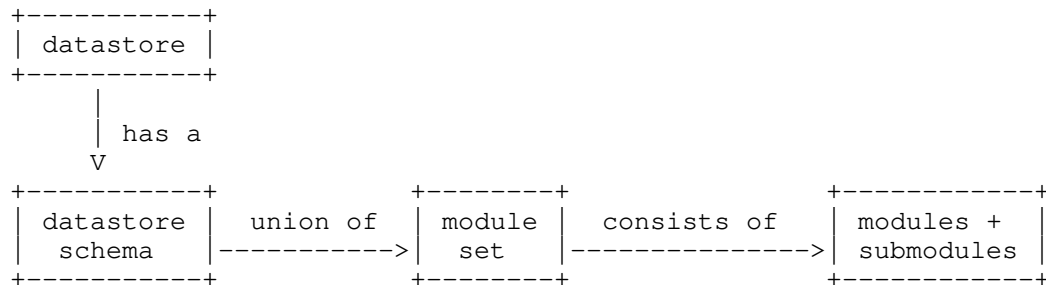


Figure 1: Conceptual model of the YANG library

It's expected that most constrained network management servers have one datastore (e.g. a unified datastore). However, some servers may have multiples datastore as described by NMDA [RFC8342]. The YANG library data model supports both cases.

In this model, every datastore has an associated datastore schema, which is the union of module sets, which is a collection of modules. Multiple datastores may refer to the same datastore schema and individual datastore schemas may share module sets.

For each module, the YANG library provides:

- o the YANG module identifier (i.e. SID)
- o its revision
- o its list of submodules
- o its list of imported modules
- o its set of features and deviations

YANG module namespace and location are also supported, but their implementation is not recommended for constrained servers.

3.1. Tree diagram

The tree diagram of YANG module `ietf-constrained-yang-library` is provided below. This graphical representation of a YANG module is defined in [RFC8340].

```

module: ietf-constrained-yang-library
  +--ro yang-library
    +--ro module-set* [index]
      +--ro index          uint8
      +--ro module* [identifier]
        +--ro identifier    sid:sid
        +--ro revision?     revision-identifier
        +--ro namespace?    inet:uri
        +--ro location*     inet:uri
        +--ro submodule* [identifier]
          +--ro identifier    sid:sid
          +--ro revision?     revision-identifier
          +--ro location*     inet:uri
        +--ro feature*      sid:sid
        +--ro deviation*    -> ../../module/identifier
      +--ro import-only-module* [identifier revision]
        +--ro identifier    sid:sid
        +--ro revision      union
        +--ro namespace?    inet:uri
        +--ro location*     inet:uri
        +--ro submodule* [identifier]
          +--ro identifier    sid:sid
          +--ro revision?     revision-identifier
          +--ro location*     inet:uri
    +--ro schema* [index]
      +--ro index          uint8
      +--ro module-set*    -> ../../module-set/index
    +--ro datastore* [identifier]
      +--ro identifier      ds:datastore-ref
      +--ro schema          -> ../../schema/index
    +--ro checksum          binary

notifications:
  +---n yang-library-update
    +--ro checksum          -> /yang-library/checksum

```

3.2. Major differences between ietf-constrained-yang-library and ietf-yang-library

The changes between the reference data model 'ietf-yang-library' and its constrained version 'ietf-constrained-yang-library' are listed below:

- o module-set 'name' and schema 'name' are implemented using an 8 bits unsigned integer and renamed 'index'.

- o module 'name', submodule 'name' and datastore 'name' are implemented using a SID (i.e. an unsigned integer) and renamed 'identifier'.
- o 'feature' and 'deviation' are implemented using a SID (i.e. an unsigned integer).
- o 'revision' fields are implemented using a 4 bytes binary string.
- o the mandatory requirement of the 'namespace' fields is removed, and implementation is not recommended. SIDs used by constrained devices and protocols don't require namespaces.
- o the implementation of the 'location' fields are not recommended, the use of the module SID as the handle to retrieve the associated YANG module is proposed instead.

4. YANG Module "ietf-constrained-yang-library"

RFC Ed.: update the date below with the date of RFC publication and remove this note.

```
<CODE BEGINS> file "ietf-constrained-yang-library@2019-03-28.yang"
module ietf-constrained-yang-library {
  yang-version 1.1;
  namespace
    "urn:ietf:params:xml:ns:yang:ietf-constrained-yang-library";
  prefix "yanglib";

  // RFC Ed.: update ietf-core-sid reference.

  import ietf-sid-file {
    prefix sid;
    reference "I-D.ietf-core-sid";
  }
  import ietf-inet-types {
    prefix inet;
    reference "RFC 6991: Common YANG Data Types.";
  }
  import ietf-datastores {
    prefix ds;
    reference
      "RFC 8342: Network Management Datastore Architecture (NMDA).";
  }

  organization
    "IETF NETCONF (Network Configuration) Working Group";
```


contact

"WG Web: <<http://datatracker.ietf.org/wg/core/>>
WG List: <<mailto:core@ietf.org>>
WG Chair: Carsten Bormann
<<mailto:cabo@tzi.org>>
WG Chair: Jaime Jimenez
<<mailto:jaime.jimenez@ericsson.com>>
Editor: Michel Veillette
<<mailto:michel.veillette@trilliantinc.com>>";

description

"This module provides information about the YANG modules, datastores, and datastore schemas implemented by a constrained network management server.

Copyright (c) 2018 IETF Trust and the persons identified as authors of the code. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, is permitted pursuant to, and subject to the license terms contained in, the Simplified BSD License set forth in Section 4.c of the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>).

This version of this YANG module is part of RFC XXXX; see the RFC itself for full legal notices.";

// RFC Ed.: update reference.

```
revision 2019-03-28 {  
  description  
    "Second revision.";  
  reference  
    "[I-D.veillette-core-yang-library]";  
}
```

```
revision 2018-09-21 {  
  description  
    "Initial revision.";  
  reference  
    "[I-D.veillette-core-yang-library]";  
}
```

```
/*
 * Typedefs
 */

typedef revision-identifier {
  type binary {
    length "4";
  }
  description
    "Revision date encoded as a binary string, each nibble
    representing a digit of the of revision date. For example,
    revision 2018-09-21 is encoded as 0x20 0x18 0x09 0x21.";
}

/*
 * Groupings
 */

grouping module-identification-leafs {
  description
    "Parameters for identifying YANG modules and submodules.";

  leaf identifier {
    type sid:sid;
    mandatory true;
    description
      "SID assigned to this module or submodule.";
  }
  leaf revision {
    type revision-identifier;
    description
      "The YANG module or submodule revision date. If no
      revision statement is present in the YANG module
      or submodule, this leaf is not instantiated.";
  }
}

grouping location-leaf-list {
  description
    "Common location leaf list parameter for modules and
    submodules.";

  leaf-list location {
    type inet:uri;
    description
      "Contains a URL that represents the YANG schema resource
      for this module or submodule."
  }
}
```

```
        This leaf is present in the model to keep the alignment
        with 'ietf-yang-library'. Support of this leaf in
        constrained devices is not necessarily required, nor
        expected. It is recommended that clients used the module
        or sub-module SID as the handle used to retrieve the
        corresponding YANG module";
    }
}

grouping implementation-parameters {
  description
    "Parameters for describing the implementation of a module.";

  leaf-list feature {
    type sid:sid;
    description
      "List of all YANG feature names from this module that are
      supported by the server, regardless whether they are
      defined in the module or any included submodule.";
  }
  leaf-list deviation {
    type leafref {
      path "../..../module/identifier";
    }
    description
      "List of all YANG deviation modules used by this server to
      modify the conformance of the module associated with this
      entry. Note that the same module can be used for
      deviations for multiple modules, so the same entry MAY
      appear within multiple 'module' entries.

      This reference MUST NOT (directly or indirectly)
      refer to the module being deviated.

      Robust clients may want to make sure that they handle a
      situation where a module deviates itself (directly or
      indirectly) gracefully.";
  }
}

grouping module-set-parameters {
  description
    "A set of parameters that describe a module set.";

  leaf index {
    type uint8;
    description
      "An arbitrary number assigned of the module set.";
  }
}
```

```
}
list module {
  key "identifier";
  description
    "An entry in this list represents a module implemented
    by the server, as per RFC 7950 section 5.6.5, with a
    particular set of supported features and deviations.";
  reference
    "RFC 7950: The YANG 1.1 Data Modeling Language.";

  uses module-identification-leafs;

  leaf namespace {
    type inet:uri;
    description
      "The XML namespace identifier for this module.
      This leaf is present in the model to keep the alignment
      with 'ietf-yang-library'. Support of this leaf in
      constrained devices is not required, nor expected.";
  }

  uses location-leaf-list;

  list submodule {
    key "identifier";
    description
      "Each entry represents one submodule within the parent
      module.";
    uses module-identification-leafs;
    uses location-leaf-list;
  }

  uses implementation-parameters;
}
list import-only-module {
  key "identifier revision";
  description
    "An entry in this list indicates that the server imports
    reusable definitions from the specified revision of the
    module, but does not implement any protocol accessible
    objects from this revision.

    Multiple entries for the same module name MAY exist.
    This can occur if multiple modules import the same
    module, but specify different revision-dates in the
    import statements.";

  leaf identifier {
```

```
    type sid:sid;
    description
        "The YANG module name.";
}
leaf revision {
    type union {
        type revision-identifier;
        type string {
            length 0;
        }
    }
    description
        "The YANG module revision date.";
}
leaf namespace {
    type inet:uri;
    description
        "The XML namespace identifier for this module.
        This leaf is present in the model to keep the alignment
        with 'ietf-yang-library'. Support of this leaf in
        constrained devices is not required, nor expected.";
}

uses location-leaf-list;

list submodule {
    key "identifier";
    description
        "Each entry represents one submodule within the
        parent module.";

    uses module-identification-leafs;
    uses location-leaf-list;
}
}

grouping yang-library-parameters {
    description
        "The YANG library data structure is represented as a grouping
        so it can be reused in configuration or another monitoring
        data structure.";

    list module-set {
        key index;
        description
            "A set of modules that may be used by one or more schemas."
    }
}
```

```

    A module set does not have to be referentially complete,
    i.e., it may define modules that contain import statements
    for other modules not included in the module set.";

    uses module-set-parameters;
}

list schema {
  key "index";
  description
    "A datastore schema that may be used by one or more
    datastores.

    The schema must be valid and referentially complete,
    i.e., it must contain modules to satisfy all used import
    statements for all modules specified in the schema.";

  leaf index {
    type uint8;
    description
      "An arbitrary reference number assigned to the schema.";
  }
  leaf-list module-set {
    type leafref {
      path "../../module-set/index";
    }
    description
      "A set of module-sets that are included in this schema.
      If a non import-only module appears in multiple module
      sets, then the module revision and the associated
      features and deviations must be identical.";
  }
}

list datastore {
  key "identifier";
  description
    "A datastore supported by this server.

    Each datastore indicates which schema it supports.

    The server MUST instantiate one entry in this list
    per specific datastore it supports.

    Each datastore entry with the same datastore schema
    SHOULD reference the same schema.";

  leaf identifier {
```

```
        type ds:datastore-ref;
        description
            "The identity of the datastore.";
    }
    leaf schema {
        type leafref {
            path "../../schema/index";
        }
        mandatory true;
        description
            "A reference to the schema supported by this datastore.
            All non import-only modules of the schema are
            implemented with their associated features and
            deviations.";
    }
}

/*
 * Top-level container
 */

container yang-library {
    config false;
    description
        "Container holding the entire YANG library of this server.";

    uses yang-library-parameters;

    leaf checksum {
        type binary;
        mandatory true;
        description
            "A server-generated checksum or digest of the contents of
            the 'yang-library' tree. The server MUST change the
            value of this leaf if the information represented by
            the 'yang-library' tree, except 'yang-library/checksum',
            has changed.";
    }
}

/*
 * Notifications
 */

notification yang-library-update {
    description
        "Generated when any YANG library information on the
```

```
        server has changed.";

    leaf checksum {
        type leafref {
            path "/yanglib:yang-library/yanglib:checksum";
        }
        mandatory true;
        description
            "Contains the YANG library checksum or digest for the
             updated YANG library at the time the notification is
             generated.";
    }
}
}
<CODE ENDS>
```

5. IANA Considerations

5.1. YANG Module Registry

This document registers one YANG module in the YANG Module Names registry [RFC7950].

name: ietf-constrained-yang-library

namespace: urn:ietf:params:xml:ns:yang:ietf-constrained-yang-library

prefix: lib

reference: RFC XXXX

// RFC Ed.: replace XXXX with RFC number and remove this note

6. Security Considerations

Some of the readable data nodes in this YANG module may be considered sensitive or vulnerable in some network environments. It is thus important to control read access to these data nodes.

Specifically, the 'module' list may help an attacker to identify the server capabilities and server implementations with known bugs. Server vulnerabilities may be specific to particular modules, module revisions, module features, or even module deviations. This information is included in each module entry. For example, if a particular operation on a particular data node is known to cause a server to crash or significantly degrade device performance, then the module list information will help an attacker to identify server

implementations with such a defect, in order to launch a denial of service attack on these devices.

7. Acknowledgments

The YANG module defined by this memo have been derived from an already existing YANG module, `ietf-yang-library` [RFC8525], we will like to thanks to the authors of this YANG module. A special thank also to Andy Bierman for his initial recommendations for the creation of this YANG module.

8. References

8.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <<https://www.rfc-editor.org/info/rfc7950>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8340] Bjorklund, M. and L. Berger, Ed., "YANG Tree Diagrams", BCP 215, RFC 8340, DOI 10.17487/RFC8340, March 2018, <<https://www.rfc-editor.org/info/rfc8340>>.
- [RFC8342] Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K., and R. Wilton, "Network Management Datastore Architecture (NMDA)", RFC 8342, DOI 10.17487/RFC8342, March 2018, <<https://www.rfc-editor.org/info/rfc8342>>.
- [RFC8525] Bierman, A., Bjorklund, M., Schoenwaelder, J., Watsen, K., and R. Wilton, "YANG Library", RFC 8525, DOI 10.17487/RFC8525, March 2019, <<https://www.rfc-editor.org/info/rfc8525>>.

8.2. Informative References

- [I-D.ietf-core-sid] Veillette, M., Pelov, A., and I. Petrov, "YANG Schema Item Identifier (SID)", draft-ietf-core-sid-07 (work in progress), July 2019.

[RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, DOI 10.17487/RFC7228, May 2014, <<https://www.rfc-editor.org/info/rfc7228>>.

Authors' Addresses

Michel Veillette (editor)
Trilliant Networks Inc.
610 Rue du Luxembourg
Granby, Quebec J2J 2V2
Canada

Email: michel.veillette@trilliantinc.com

Ivaylo Petrov (editor)
Acklio
1137A avenue des Champs Blancs
Cesson-Sevigne, Bretagne 35510
France

Email: ivaylo@ackl.io