CoRE
Internet-Draft
Intended status: Standards Track
Expires: 4 September 2022

C. Amsüss

3 March 2022

CoAP Protocol Indication
draft-amsuess-core-transport-indication-03

Abstract

The Constrained Application Protocol (CoAP, [RFC7252]) is available over different transports (UDP, DTLS, TCP, TLS, WebSockets), but lacks a way to unify these addresses. This document provides terminology and provisions based on Web Linking [RFC8288] to express alternative transports available to a device, and to optimize exchanges using these.

Discussion Venues

This note is to be removed before publishing as an RFC.

Discussion of this document takes place on the Constrained RESTful Environments Working Group mailing list (core@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/core/.

Source for this draft and an issue tracker can be found at https://gitlab.com/chrysn/transport-indication.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction 3
  1.1. Terminology 3
    1.1.1. Using URIs to identify protocol endpoints 4
  1.2. Goals 5
2. Indicating alternative transports 6
  2.1. Example 7
  2.2. Security context propagation 8
  2.3. Choice of transports 8
  2.4. Selection of a canonical origin 9
  2.5. Advertisement through a Resource Directory 9
3. Elision of Proxy-Scheme and Uri-Host 9
  3.1. Impact on caches 11
  3.2. Using unique proxies securely 12
4. Third party proxy services 13
  4.1. Generic proxy advertisements 14
5. Client picked proxies 15
6. Security considerations 16
  6.1. Security context propagation 16
  6.2. Traffic misdirection 16
  6.3. Protecting the proxy 17
7. IANA considerations 17
  7.1. Link Relation Types 17
  7.2. Resource Types 17
8. References 18
  8.1. Normative References 18
  8.2. Informative References 18
Appendix A. Change log 20
Appendix B. Related work and applicability to related fields . . 21
  B.1. On HTTP 22
  B.2. Using DNS 22
  B.3. Using names outside regular DNS 23
  B.4. Multipath TCP 23

Appendix C. Open Questions / further ideas 24
Appendix D. Acknowledgements 25
Author's Address 25

1. Introduction

The Constrained Application Protocol (CoAP) provides transports mechanisms (UDP and DTLS since [RFC7252], TCP, TLS and WebSockets since [RFC8323]), with some additional being used in LwM2M [lwm2m] and even more being explored ([I-D.bormann-t2trg-slipmux], [I-D.amsuess-core-coap-over-gatt]). These are mutually incompatible on the wire, but CoAP implementations commonly support several of them, and proxies can translate between them.

CoAP currently lacks a way to indicate which transports are available for a given resource, and to indicate that a device is prepared to serve as a proxy; this document solves both by introducing the "has-proxy" terminology to Web Linking [RFC8288] that expresses the former through the latter. The additional "has-unique-proxy" term is introduced to negate any per-request overhead that would otherwise be introduced in the course of this.

CoAP also lacks a unified scheme to label a resource in a transport-independent way. This document does _not_ attempt to introduce any new scheme here, or raise a scheme to be the canonical one. Instead, each host or application can pick a canonical address for its resources, and advertise other transports in addition.

1.1. Terminology

Readers are expected to be familiar with the terms and concepts described in CoAP [RFC7252] and link format ([RFC6690] (or, equivalently, web links as described in [RFC8288]).

Same-host proxy: A CoAP server that accepts forward proxy requests (i.e., requests carrying the Proxy-Scheme option) exclusively for URIs that it is also the authoritative server for is defined as a "same-host proxy".

The distinction between a same-host and any other proxy is only relevant on a practical, server-implementation and illustrative level; this specification does not use the distinction in normative requirements, and clients need not make the distinction at all.

hosts: The verb "to host" is used here in the sense of the link relation of the same name defined in [RFC6690].

For resources discovered via CoAP's discovery interface, a hosting statement is typically provided by the defaults implied by [RFC6690] where a link like </sensor/temp> is implied to have the relation "hosts" and the anchor /, such that a statement "coap://hostname hosts coap://hostname/sensor/temp" is implied in the link.

The link relation has been occasionally used with different interpretations, which ascribe more meaning to the term than it has in its definition. In particular,

* the "hosts" relation can not be inferred merely by two URIs having the same scheme, host and port (and vice versa), and

* the "hosts" relation on its own does not make any statement about the physical devices that hold the resource's representation.

[TBD: The former could probably still be used without too many ill effects; but things might also get weird when a dynamic resource created with one transport from use with another transport unless explicitly cleared.]

When talking of proxy requests, this document only talks of the Proxy-Scheme option. Given that all URIs this is usable with can be expressed in decomposed CoAP URIs, the need for using the Proxy-URI option should never arise. The Proxy-URI option is still equivalent to the decomposed options, and can be used if the server supports it.

1.1.1. Using URIs to identify protocol endpoints

The URI coap://device.example.com identifies a particular resource, possibly a "welcome" text. It is, colloquially, also used to identify the combination of a host (identified through a name), the default port, and the CoAP method of sending requests to the host.

For precision, this document uses the term "the transport address indicated by (a URI)" to refer to the host / port / protocol combination, but otherwise no big deal is made of it.

For the CoAP schemes (coap, coaps, coap+tcp, coaps+tcp, coap+ws, coaps+ws), URIs indicating a transport are always given with an empty path (which under their URI normalization rules is equivalent to a path containing a single slash). For the coap and coap+tcp schemes, URIs with different host names can indicate the same transport as long as the names resolve to the same addresses. For the other protocols, the given host name informs the name set in TLS's Server Name Indication (SNI) and/or the host sent in the "Host" header of the underlying HTTP request.

If an update to this document extends the list, for new schemes it might be allowed to have paths, queries or fragment identifiers present in the URI indicating the transport address. No guidance can be given here for these, as no realistic example is known. (Note that while the coap+ws scheme does use the well-known path /.well-known/coap internally, that is used purely on the HTTP side, and not part of the CoAP URI, not even for indicating the transport address).

URIs indicating a transport are especially useful when talking about proxies; this use is aligned with the way they are exprssed in the conventional environment variables http_proxy etc. [cite https://about.gitlab.com/blog/2021/01/27/we-need-to-talk-no-proxy/]. Furthermore, URIs processing is widespread in CoAP systems, and when that changes (e.g. through the introduction of [I-D.ietf-core-href]), URIs indicating a transport will still be efficient to encode. And last but not least, it lines up well with the colloquial identity mentioned above. (An alternative would be using a dedicated naming scheme, say, transport:coap:device.example.com:port, but that would needlessly introduce implementation complexity).

Note that this mechanism can only used with proxies that use CoAP's native address indication mechanisms. Proxies that perform URI mapping (as described in Section 5 of [RFC8075], especially using URI templates) are not supported in this document.

[TBD: Do we want to extend this to HTTP proxies? Probably just not, and if so, only to those that can just take coap://... for a URI.]

1.2. Goals

This document introduces provisions for the seamless use of different transport mechanisms for CoAP. Combined, these provide:

* Enablement: Inform clients of the availability of other transports of servers.

* No Aliasing: Any URI aliasing must be opt-in by the server. Any defined mechanisms must allow applications to keep working on the canonical URIs given by the server.

* Optimization: Do not incur per-request overhead from switching protocols. This may depend on the server's willingness to create aliased URIs.

* Proxy usability: All information provided must be usable by aware proxies to reduce the need for duplicate cache entries.

* Proxy announcement: Allow third parties to announce that they provide alternative transports to a host.

For all these functions, security policies must be described that allow the client to use them as securely as the original transport.

This document will not concern itself with changes in transport availability over time, neither in causing them ("Please take up your TCP interface, I'm going to send a firmware update") nor in advertising their availability in advance. Hosts whose transport's availability changes over time can utilize any suitable mechanism to keep client updated, such as placing a suitable Max-Age value on their resources or having them observable.

2. Indicating alternative transports

While CoAP can set the authority component of the requested URI in all requests (by means of Uri-Host and Uri-Port), setting the scheme of a requested URI (by means of Proxy-Scheme) makes the request implicitly a proxy request. However, this needs to be of only little practical concern: Any device can serve as a proxy for itself (a "same-host proxy") by accepting requests that carry the Proxy-Scheme option. If it is to be a well-behaved proxy, the device should then check whether it recognizes the name set in Uri-Host as one of its own (as it should if no Proxy-Scheme option accompanied it). If the name is not recognized, it should reject the request with 5.05 (Proxying Not Supported) -- unless, of course, it implements forward proxy functionality exceeding the same-host proxy. If the name is recognized, it should process the request as it would process a request coming in on the given protocol (which, for many hosts, is the same as if the option were absent completely).

A server can advertise a recommended proxy by serving a Web Link with the "has-proxy" relation to a URI indicating its transport address. In particular (and that is a typical case), it can indicate its own transport address on an alternative transport when implementing same-host proxy functionality.

The semantics of a link from S to P with relations has-proxy ("S has-proxy P", <P>;rel=has-proxy;anchor="S") are that for any resource R hosted on S ("S hosts R"), the proxy with the transport address indicated by P can be used to obtain R.

2.1. Example

A constrained device at the address 2001:db8::1 that supports CoAP over TCP in addition to CoAP can self-describe like this:

```
Req: to [ff02::fd]:5683 on UDP
Code: GET
Uri-Path: /.well-known/core
Uri-Query: if=tag:example.com,sensor

Res: from [2001:db8::1]:5683
Content-Format: application/link-format
Payload:
</sensors/temp>;if="tag:example.com,sensor",
<coap+tcp://[2001:db8::1]>;rel=has-proxy;anchor="/"


Req: to [2001:db8::1]:5683 on TCP
Code: GET
Proxy-Scheme: coap
Uri-Path: /sensors/temp
Observe: 0

Res: 2.05 Content
Observe: 0
Payload:
39.1°C
```

Figure 1: Discovery and follow-up request through a has-proxy relation

Note that generating this discovery file needs to be dynamic based on its available addresses; only if queried using a link-local source address, the server may also respond with a link-local address in the authority component of the proxy URI.

Unless the device makes resources discoverable at coap+tcp://[2001:db8::1]/.well-known/core or another discovery mechanism, clients may not assume that coap+tcp://[2001:db8::1]/sensors/temp is a valid resource (let alone is equivalent to the other resource on the same path). The server advertising itself like this may reject any request on CoAP-over-TCP unless it contains a Proxy-Scheme option.

Clients that want to access the device using CoAP-over-TCP would send a request by connecting to 2001:db8::1 TCP port 5683 and sending a GET with the options Proxy-Scheme: coap, no Uri-Host or -Port options (utilizing their default values), and the Uri-Paths "sensors" and "temp".

2.2. Security context propagation

If the originally requested URI R or the application requirements demand a security mechanism is used, the client MUST only use the proxy P if the proxy can provide suitable credentials. (The hosting URI S is immaterial to these considerations).

For example, if the application uses the host name and a public key infrastructure and R is coap://example.com/ the proxy accessed as coap+tcp://[2001:db8::1] still needs to provide a certificate chain for the name example.com to one of the system's trust anchors. If, on the other hand, the application is doing a firmware update and requires any certificate from its configured firmware update issuer, the proxy needs to provide such a firmware update certificate.

Some applications have requirements exceeding the requirements of a secure connection, e.g., (explicitly or implicitly) requiring that name resolution happen through a secure process and packets are only routed into networks where it trusts that they will not be intercepted on the path to the server. Such applications need to extend their requirements to the source of the has-proxy statement; a sufficient (but maybe needlessly strict) requirement is to only follow has-proxy statements that are part of the same resource that advertises the link currently being followed. Section Section 6.2 adds further considerations.

2.3. Choice of transports

It is up to the client whether to use an advertised proxy transport, or (if multiple are provided) which to pick.

Links to proxies may be annotated with additional metadata that may help guide such a choice; defining such metadata is out of scope for this document.

Clients MAY switch between advertised transports as long as the document describing them is fresh; they may even do so per request. (For example, they may perform individual requests using CoAP-over-UDP, but choose CoAP-over-TCP for requests with large expected responses). When the describing document approaches expiry, the client can use the representation's ETag to efficiently renew its justification for using the alternative transport.

2.4. Selection of a canonical origin

While a server is at liberty to provide the same resource independently on different transports (i.e. to create aliases), it may make sense for it to pick a single scheme and authority under which it announces its resources. Using only one address helps proxies keep their caches efficient, and makes it easier for clients to avoid exploring the same server twice from different angles.

When there is a predominant scheme and authority through which an existing service is discovered, it makes sense to use these for the canonical addresses.

Otherwise, it is suggested to use the coap or coaps scheme (given that these are the most basic and widespread ones), and the most stable usable name the host has.

2.5. Advertisement through a Resource Directory

In the Resource Directory specification [I-D.ietf-core-resource-directory], protocol negotiation was anticipated to use multiple base values. This approach was abandoned since then, as it would incur heavy URI aliasing.

Instead, devices can submit their has-proxy links to the Resource Directory like all their other metadata.

A client performing resource lookup can ask the RD to provide available (same-host-)proxies in a follow-up request by asking for ?anchor=<the-discovered-host>&rel=has-proxy. The RD may also volunteer that information during resource lookups even though the has-proxy link itself does not match the search criteria.

[It may be useful to define RD parameters for use with lookup here, which'd guide which available proxies to include. For example, asking ?if=tag:example.com,sensor&proxy-links=tcp could give as a result: <coap://[2001:db8::1]/s>;rt=tag:example.com,sensor,<coap+tcp://[2001:db8::1]/>;rel=has-proxy;anchor="coap://[2001:db8::1]/"]

3. Elision of Proxy-Scheme and Uri-Host

A CoAP server may publish and accept multiple URIs for the same resource, for example when it accepts requests on different IP addresses that do not carry a Uri-Host option, or when it accepts requests both with and without the Uri-Host option carrying a registered name. Likewise, the server may serve the same resources on different transports. This makes for efficient requests (with no Proxy-Scheme or Uri-Host option), but in general is discouraged

[aliases].

To make efficient requests possible without creating URI aliases that propagate, the "has-unique-proxy" specialization of the has-proxy relation is defined.

If a proxy is unique, it means that requests arriving at the proxy are treated the same no matter whether the scheme, authority and port of the link context are set in the Proxy-Scheme, Uri-Host and Uri-Port options, respectively, or whether all of them are absent.

[The following two paragraphs are both true but follow different approaches to explaining the observable and implementable behavior; it may later be decided to focus on one or the other in this document.]

While this creates URI aliasing in the requests as they are sent over the network, applications that discover a proxy this way should not "think" in terms of these URIs, but retain the originally discovered URIs (which, because Cool URIs Don't Change[cooluris], should be long-term usable). They use the proxy for as long as they have fresh knowledge of the has-(unique-)proxy statement.

In a way, advertising has-unique-proxy can be viewed as a description of the link target in terms of SCHC [I-D.ietf-lpwan-coap-static-context-hc]: In requests to that target, the link source's scheme and host are implicitly present.

While applications retain knowledge of the originally requested URI (even if it is not expressed in full on the wire), the original URI is not accessible to caches both within the host and on the network (for the latter, see Section 5). Thus, cached responses to the canonical and any aliased URI are mutually interchangeable as long as both the response and the proxy statement are fresh.

A client MAY use a unique-proxy like a proxy and still send the Proxy-Scheme and Uri-Host option; such a client needs to recognize both relation types, as relations of the has-unique-proxy type are a specialization of has-proxy and typically don't carry the latter (redundant) annotation. [To be evaluated -- on one hand, supporting it this way means that the server needs to identify all of its addresses and reject others. Then again, is a server that (like many now do) fully ignore any set Uri-Host correct at all?]

Example:

```
Req: to [ff02::fd]:5683 on UDP
Code: GET
Uri-Path: /.well-known/core
Uri-Query: if=tag:example.com,sensor

Res: from [2001:db8::1]:5683
Content-Format: application/link-format
Payload:
</sensors/>;if="tag:example.com,collection",
<coaps+ws://[2001:db8::1]>;rel=has-unique-proxy;anchor="/"


Req: to [2001:db8::1] via WebSockets over HTTPS
Code: GET
Uri-Path: /sensors/

Res: 2.05 Content
Content-Format: application/link-format
Payload:
</sensors/temperature>;if="tag:example.com,sensor"
```

Figure 2: Follow-up request through a has-unique-proxy relation. Compared to the last example, 5 bytes of scheme indication are saved during the follow-up request.

It is noteworthy that when the URI reference /sensors/temperature is resolved, the base URI is coap://device0815.example.com and not its coaps+ws counterpart -- as the request is still for that URI, which both the client and the server are aware of. However, this detail is of little practical importance: A simplistic client that uses coaps+ws://device0815.proxy.rd.example.com as a base URI will still arrive at an identical follow-up request with no ill effect, as long as it only uses the wrongly assembled URI for dereferencing resources, the security context is the same, the state is kept no longer than the has-unique-proxy statement is fresh, and it does not (for example) pass the URI on to other devices.

3.1. Impact on caches

[This section is written with the "there is implied URI aliasing" mindset; it should be possible to write it with the "compression" mindset as well (but there is no point in having both around in the document at this time).

It is also slightly duplicating, but also more detailed than, the brief note on the topic in Section 5]

When a node that performs caching learns of a has-unique-proxy statement, it can utilize the information about the implied URI aliasing: Requests to resources hosted by S can be answered with cached entries from P (because by the rules of has-unique-proxy a request can be crafted that is sent to P for which a fresh response is available). The inverse direction (serving resources whose URI "starts with" P from a cached request that was sent to S) is harder to serve because it additionaly requires a fresh statement that "S hosts R" for the matching resource R.

3.2. Using unique proxies securely

[This section is work in progress, it is more a flow of considerations turning back on each other. This is all made a bit trickier by not applying to OSCORE which is usually the author's go-to example, because OSCORE's requirements already preclude all these troubles.]

The use of unique proxies requires slightly more care in terms of security.

No requirements are necessary on the client side; those of {#secctx-propagation} suffice. (In particular, it is not necessary for the statement to originate from the original server unless that were already a requirement without the uniqueness property).

The extra care is necessary on the side of servers that are commissioned with wide ranging authorization [or is it?]: These may now be tricked into serving a resource of which the client assumes a different name. For example, if the desired resource is coaps://high-security.example.org/configuration, and there exists a "home page" style service for employees with patterns of coaps+tcp://user-${username}.example.org/ at which they can store files, and the server operating that service is commissioned with a wild-card certificate "*.example.org", then a device that receives the (malicious) information <coaps+tcp://user-mave.example.org>;rel=has-unique-proxy;anchor="coaps://high-security.example.org" might use this statement to contact the transport address indicated by coaps+tcp://user-mave.example.org and ask for /config (which, to the server, is indistinguishable from coaps+tcp://user-mave.example.org/config) and obtain a malicious configuration.

In a non-unique proxy situation, the error would have been caught by the server, which would have seen the request for coaps://high-security.example.com and refused to serve a request containing critical options it can not adaequately process.

In the unique proxy situation, ... [TBD: now whose fault is it? Can only be the client's ... because it looked at the wildcard certificate rather than whether the host-name it was narrowing it down to is authorized to speak for high-security.example.com? The server (operator) can barely be blamed, for while the certificate is needlessly wide, to the server it did look precisely like a good request.]

4. Third party proxy services

A server that is aware of a suitable cross proxy may use the has-proxy relation to advertise that proxy. If the protocol used towards the proxy provides name indication (as CoAP over TLS or WebSockets does), or by using a large number of addresses or ports, it can even advertise a (more efficient) has-unique-proxy relation. This is particularly interesting when the advertisements are made available across transports, for example in a Resource Directory.

How the server can discover and trust such a proxy is out of scope for this document, but generally involves the same kind of links. In particular, a server may obtain a link to a third party proxy from an administrator as part of its configuration.

The proxy may advertise itself without the origin server's involvement; in that case, the client needs to take additional care (see Section 6.2).

```
Req: GET http://rd.example.com/rd-lookup?if=tag:example.com,sensor

Res:
Content-Format: application/link-format
Payload:
<coap://device0815.example.com/sensors/>;if="tag:example.com,collection",
<coap+wss://device0815.proxy.rd.example.com>;rel=has-unique-proxy;anchor="coap://
device0815.example.com/"


Req: to device0815.proxy.rd.example.com on WebSocket
Host (indicated during upgrade): device0815.proxy.rd.example.com
Code: GET
Uri-Path: /sensors/

Res: 2.05 Content
Content-Format: application/link-format
Payload:
</sensors/temperature>;if="tag:example.com,sensor"
```

Figure 3: HTTP based discovery and CoAP-over-WS request to a CoAP resource through a has-unique-proxy relation

4.1. Generic proxy advertisements

A third party proxy may advertise its availability to act as a proxy for arbitrary CoAP requests. This use is not directly related to the protocol indication in other parts of this document, but sufficiently similar to warrant being described in the same document.

The resource type "TBDcore.proxy" can be used to describe such a proxy. The link target attribute "proxy-schemes" can be used to indicate the scheme(s) supported by the proxy, separated by the space character.

```
Req: GET coap://[fe80::1]/.well-known/core?rt=TBDcore.proxy

Res:
Content-Format: application/link-format
Payload:
<>;rt=TBDcore.proxy;proxy-schemes="coap coap+tcp coap+ws http"

Req: to [fe80::1] via CoAP
Code: GET
Proxy-Scheme: http
Uri-Host: example.com
Uri-Path: /motd
Accept: text/plain

Res: 2.05 Content
Content-Format: text/plain
Payload:
On Monday, October 25th 2021, there is no special message of the day.
```

Figure 4: A CoAP client discovers that its border router can also serve as a proxy, and uses that to access a resource on an HTTP server.

The considerations of Section 6.2 apply here.

A generic advertised proxy is always a forward proxy, and can not be advertised as a "unique" proxy as it would lack information about where to forward. (A proxy limited to a single outbound protocol might in theory work as a unique proxy when using a transport in which the full default Uri-Host value is configured at setup time, but these are considered impractical and thus not assigned a resource type here.)

The use of a generic proxy can be limited to a set of devices that have permission to use it. Clients can be allowed by their network address if they can be verified, or by using explicit client authentication using the methods of [I-D.tiloca-core-oscore-capable-proxies].

5. Client picked proxies

This section is purely informative, and serves to illustrate that the mechanisms introduced in this document do not hinder the continued use of existing proxies.

When a resource is accessed through an "actual" proxy (i.e., a host between the client and the server, which itself may have a same-host proxy in addition to that), the proxy's choice of the upstream server is originally (i.e., without the mechanisms of this document) either configured (as in a "chain" of proxies) or determined by the request URI (where a proxy picks CoAP over TCP and resolves the given name for a request aimed at a coap+tcp URI).

A proxy that has learned, by active solicitation of the information or by consulting links in its cache, that the requested URI is available through a (possibly same-host) proxy, may use that information in choosing the upstream transport, to correct the URI associated with a cached response, and to use responses obtained through one transport to satisfy requests on another.

For example, if a host at coap://h1.example.com has advertised </res>,<coap+tcp://h1.example.com>;rel=has-proxy;anchor="/", then a proxy that has an active CoAP-over-TCP connection to h1.example.com can forward an incoming request for coap://h1.example.com/res through that CoAP-over-TCP connection with a suitable Proxy-Scheme on that connection.

If the host had marked the proxy point as <coap+tcp://h1.example.com>;rel=has-unique-proxy instead, then the proxy could elide the Proxy-Scheme and Uri-Host options, and would (from the original CoAP caching rules) also be allowed to use any fresh cache representation of coap+tcp://h1.example.com/res to satisfy requests for coap://h1.example.com/res.

A client that uses a forward proxy and learns of a different proxy advertised to access a particular resource will not change its behavior if its original proxy is part of its configuration. If the forward proxy was only used out of necessity (e.g., to access a resource on the protocol not supported by the client) it can be practical for the client to use the advertised proxy instead.

6. Security considerations

6.1. Security context propagation

Clients need to strictly enforce the rules of Section 2.2. Failure to do so, in particular using a thusly announced proxy based on a certificate that attests the proxy's name, would allow attackers to circumvent the client's security expectation.

When security is terminated at proxies (as is in DTLS and TLS), a third party proxy can usually not satisfy this requirement; these transports are limited to same-host proxies.

6.2. Traffic misdirection

Accepting arbitrary proxies, even with security context propagation performed properly, would allow attackers to redirect traffic through systems under their control. Not only does that impact availability, it also allows an attacker to observe traffic patterns.

This affects both OSCORE and (D)TLS, as neither protect the participants' network addresses.

Other than the security context propagation rules, there are no hard and general rules about when an advertised proxy is a suitable candidate. Aspects for consideration are:

* When no direct connection is possible (e.g. because the resource to be accessed is served as coap+tcp and TCP is not implemented in the client, or because the resource's host is available on IPv6 while the client has no default IPv6 route), using a proxy is necessary if complete service disruption is to be avoided.

  While an adversary can cause such a situation (e.g. by manipulating routing or DNS entries), such an adversary is usually already in a position to observe traffic patterns.

* A proxy advertised by the device hosting the resource to be accessed is less risky to use than one advertised by a third party.

  Note that in some applications, servers produce representations based on unverified user input. In such cases, and more so when multiple applications share a security context, the advertisements' provenance may need to be considered.

6.3. Protecting the proxy

A widely published statement about a host's availability as a proxy can cause many clients to attempt to use it.

This is mitigated in well-behaved clients by observing the rate limits of [RFC7252], and by ceasing attempts to reach a proxy for the Max-Age of received errors.

Operators can further limit ill-effects by ensuring that their client systems do not needlessly use proxies advertised in an unsecured way, and by providing own proxies when their clients need them.

7. IANA considerations

7.1. Link Relation Types

IANA is asked to add two entries into the Link Relation Type Registry last updated in [RFC8288]:

Relation Name	Description	Reference
has-proxy	The link target can be used as a proxy to reach the link context.	RFCthis
has-unique-proxy	Like has-proxy, and using this proxy implies scheme and host of the target.	RFCthis

Table 1: New Link Relation types

7.2. Resource Types

IANA is asked to add an entry into the "Resource Type (rt=) Link Target Attribute Values" registry under the Constrained RESTful Environments (CoRE) Parameters:

[The RFC Editor is asked to replace any occurrence of TBDcore.proxy with the actually registered attribute value.]

Attribute Value: core.proxy

Description: Forward proxying services

Reference: [this document]

Notes: The schemes for which the proxy is usable may be indicated using the proxy-schemes target attribute as per Section 4.1 of [this document].

8. References

8.1. Normative 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/rfc/rfc7252>.

[RFC8288] Nottingham, M., "Web Linking", RFC 8288, DOI 10.17487/RFC8288, October 2017, <https://www.rfc-editor.org/rfc/rfc8288>.

8.2. Informative References

[aliases] W3C, "Architecture of the World Wide Web, Section 2.3.1 URI aliases", n.d., <https://www.w3.org/TR/webarch/#uri-aliases>.

[cooluris] BL, T., "Cool URIs don’t change", n.d., <https://www.w3.org/Provider/Style/URI>.

[I-D.amsuess-core-coap-over-gatt] Amsüss, C., "CoAP over GATT (Bluetooth Low Energy Generic Attributes)", Work in Progress, Internet-Draft, draft-amsuess-core-coap-over-gatt-01, 2 November 2020, <https://datatracker.ietf.org/doc/html/draft-amsuess-core-coap-over-gatt-01>.

[I-D.amsuess-t2trg-rdlink] Amsüss, C., "rdlink: Robust distributed links to constrained devices", Work in Progress, Internet-Draft, draft-amsuess-t2trg-rdlink-01, 23 September 2019, <https://datatracker.ietf.org/doc/html/draft-amsuess-t2trg-rdlink-01>.

[I-D.bormann-t2trg-slipmux] Bormann, C. and T. Kaupat, "Slipmux: Using an UART interface for diagnostics, configuration, and packet transfer", Work in Progress, Internet-Draft, draft-bormann-t2trg-slipmux-03, 4 November 2019, <https://datatracker.ietf.org/doc/html/draft-bormann-t2trg-slipmux-03>.

[I-D.ietf-core-href]
Bormann, C. and H. Birkholz, "Constrained Resource Identifiers", Work in Progress, Internet-Draft, draft-ietf-core-href-09, 15 January 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-core-href-09>.

[I-D.ietf-core-resource-directory]
Amsüss, C., Shelby, Z., Koster, M., Bormann, C., and P. V. D. Stok, "CoRE Resource Directory", Work in Progress, Internet-Draft, draft-ietf-core-resource-directory-28, 7 March 2021, <https://datatracker.ietf.org/doc/html/draft-ietf-core-resource-directory-28>.

[I-D.ietf-lpwan-coap-static-context-hc]
Minaburo, A., Toutain, L., and R. Andreasen, "Static Context Header Compression (SCHC) for the Constrained Application Protocol (CoAP)", Work in Progress, Internet-Draft, draft-ietf-lpwan-coap-static-context-hc-19, 8 March 2021, <https://datatracker.ietf.org/doc/html/draft-ietf-lpwan-coap-static-context-hc-19>.

[I-D.silverajan-core-coap-protocol-negotiation]
Silverajan, B. and M. Ocak, "CoAP Protocol Negotiation", Work in Progress, Internet-Draft, draft-silverajan-core-coap-protocol-negotiation-09, 2 July 2018, <https://datatracker.ietf.org/doc/html/draft-silverajan-core-coap-protocol-negotiation-09>.

[I-D.tiloca-core-oscore-capable-proxies]
Tiloca, M. and R. Höglund, "OSCORE-capable Proxies", Work in Progress, Internet-Draft, draft-tiloca-core-oscore-capable-proxies-01, 25 October 2021, <https://datatracker.ietf.org/doc/html/draft-tiloca-core-oscore-capable-proxies-01>.

[lwm2m] OMA SpecWorks, "White Paper Lightweight M2M 1.1", n.d., <https://omaspecworks.org/white-paper-lightweight-m2m-1-1/>.

[RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <https://www.rfc-editor.org/rfc/rfc6690>.

[RFC6763] Cheshire, S. and M. Krochmal, "DNS-Based Service Discovery", RFC 6763, DOI 10.17487/RFC6763, February 2013, <https://www.rfc-editor.org/rfc/rfc6763>.

[RFC7838] Nottingham, M., McManus, P., and J. Reschke, "HTTP Alternative Services", RFC 7838, DOI 10.17487/RFC7838, April 2016, <https://www.rfc-editor.org/rfc/rfc7838>.

[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/rfc/rfc8075>.

[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/rfc/rfc8323>.

[RFC8613] Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security for Constrained RESTful Environments (OSCORE)", RFC 8613, DOI 10.17487/RFC8613, July 2019, <https://www.rfc-editor.org/rfc/rfc8613>.

Appendix A. Change log

Since -02 (mainly processing reviews from Marco and Klaus):

* Acknowledge that ’coap://hostname/’ is not the proxy but a URI that (in a particular phrasing) is used to stand in for the proxy’s address (while it regularly identifies a resurce on the server)

* Security: Referencing traffic misdirection already in the first security block.

* Security: Add (incomplete) considerations for unique-proxy case.

* Narrow down "unique" proxy semantics to those properties used by the client, allowing unique proxies to be co-hosted with forward proxies.

* "Client picked proxies" clarified to merely illustrate how this is compatible with them.

* Use of "hosts" relation sharpened.

* Precision on how this does and does not consider changing transports.

* "Related work" section demoted to appendix.

* Add note on DTLS session resumption.

* Variable renaming.

* Various editorial fixes.

Since -01:

* Removed suggestion for generally trusted proxies; now stating that with (D)TLS, "a third party proxy can usually not satisfy [the security context propagation requirement]".

* State more clearly that valid cache entries for resources aliased through has-unique-proxy can be used.

* Added considerations for Multipath TCP.

* Added concrete suggestion and example for advertisement of general proxies.

* Added concrete suggestion for RD lookup extension that provides proxies.

* Minor editorial and example changes.

Since -00:

* Added introduction

* Added examples

* Added SCHC analogy

* Expanded security considerations

* Added guidance on choice of transport, and canonical addresses

* Added subsection on interaction with a Resource Directory

* Added comparisons with related work, including rdlink and DNS-SD sketches

* Added IANA considerations

* Added section on open questions

Appendix B. Related work and applicability to related fields

B.1. On HTTP

The mechanisms introduced here are similar to the Alt-Svc header of [RFC7838] in that they do not create different application-visible addresses, but provide dispatch through lower transport implementations.

Unlike in HTTP, the variations of CoAP protocols each come with their unique URI schemes and thus enable the "transport address indicated by a URI" concept. Thus, there is no need for a distinction between protocol-id and scheme.

To accommodate the message size constraints typical of CoRE environments, and accounting for the differences between HTTP headers and CoAP options, information is delivered once at discovery time.

Using the has-proxy and has-unique-proxy with HTTP URIs as the context is NOT RECOMMENDED; the HTTP provisions of the Alt-Svc header and ALPN are preferred.

B.2. Using DNS

As pointed out in [RFC7838], DNS can already serve some of the applications of Alt-Svc and has-unique-proxy by providing different CNAME records. These cover cases of multiple addresses, but not different ports or protocols.

While not specified for CoAP yet (and neither being specified here),

[which is an open discussion point for CoRE -- should we? Here? In a separate DNS-SD document?]

DNS SRV records (possibly in combination with DNS Service Discovery [RFC6763]) can provide records that could be considered equivalent to has-unique-proxy relations. If _coap._tcp, _coaps._tcp, _coap._udp, _coap+ws._tcp etc. were defined with suitable semantics, these can be equivalent:

```
_coap._udp.device.example.com SRV 0 0 device.example.com 61616
device.example.com AAAA 2001:db8::1

<coap://[2001:db8::1]>;rel=has-unique-proxy;anchor="coap://device.example.com"
```

It would be up to such a specification to give details on what the link’s context is; unlike the link based discovery of this document, it would either need to pick one distinguished context scheme for which these records are looked up, or would introduce aliasing on its own.

B.3. Using names outside regular DNS

Names that are resolved through different mechanisms than DNS, or names which are defined within the scope of DNS but have no universally valid answers to A/AAAA requests, can be advertised using the relation types defined here and CoAP discovery.

In Figure 5, a server using a cryptographic name as described in [I-D.amsuess-t2trg-rdlink] is discovered and used.

```
Req: to [ff02::fd]:5683 on UDP
Code: GET
Uri-Path: /.well-known/core
Uri-Query: rel=has-proxy
Uri-Query: anchor=coap://nbswy3dpo5xxe3denbswy3dpo5xxe3de.ab.rdlink.arpa

Res: from [2001:db8::1]:5683
Content-Format: application/link-format
Payload:
<coap+tcp://[2001:db8::1]>;rel=has-unique-proxy;
  anchor="coap://nbswy3dpo5xxe3denbswy3dpo5xxe3de.ab.rdlink.arpa"


Req: to [2001:db8::1]:5683 on TCP
Code: GET
OSCORE: ...
Uri-Path: /sensors/temp
Observe: 0

Res: 2.05 Content
OSCORE: ...
Observe: 0
Payload:
39.1°C
```

Figure 5: Obtaining a sensor value from a local device with a global name

B.4. Multipath TCP

When CoAP-over-TCP is used over Multipath TCP and no Uri-Host option is sent, the implicit assumption is that there is aliasing between URIs containing any of the endpoints' addresses.

As these are negotiated within MPTCP, this works independently of this document's mechanisms. As long as all the server's addresses are equally reachable, there is no need to advertise multiple addresses that can later be discovered through MPTCP anyway. When

advertisements are long-lived and there is no single more stable address, several available addresses can be advertised (independently of whether MPTCP is involved or not). If a client uses an address that is merely a proxy address (and not a unique proxy address), but during MPTCP finds out that the network location being accessed is actually an MPTCP alternative address of the used one, the client MAY forego sending of the Proxy-Scheme and Uri-Path option.

[This follows from multiple addresses being valid for that TCP connection; at some point we may want to say something about what that means for the default value of the Uri-Host option -- maybe something like "has the default value of any of the associated addresses, but the server may only enable MPTCP if there is implicit aliasing between all of them" (similar to OSCORE's statement)?]

[TBD: Do we need a section analog to this that deals with (D)TLS session resumption in absence of SNI?]

Appendix C. Open Questions / further ideas

* OSCORE interaction: [RFC8613] Section 4.1.3.2 requirements place OSCORE use in a weird category between has-proxy and has-unique-proxy (because if routing still works, the result will be correct). Not sure how to write this down properly, or whether it's actionable at all.

  Possibly there is an inbetween category of "The host needs the Uri-Host etc. when accessed through CoAP, but because the host does not use the same OSCORE KID across different virtual hosts, it's has-unique-proxy as soon as you talk OSCORE".

* Self-uniqueness:

  A host that wants to indicate that it doesn't care about Uri-Host can probably publish something like </>;rel=has-unique-proxy to do so.

  This'd help applications justify when they can elide the Uri-Host, even when no different protocols are involved.

* Advertising under a stable name:

  If a host wants to advertise its host name rather than its IP address during multicast, how does it best do that?

  Options, when answering from 2001:db8::1 to a request to ff02::fd are:

```
<coap://myhostname/foo>,...,
<coap://[2001:db8::1]>;rel=has-unique-proxy;anchor="coap://myhostname"
```

which is verbose but formally clear, and

```
</foo>,...,
<coap://[2001:db8::1]>;rel=has-unique-proxy;anchor="coap://myhostname"
```

which is compatible with unaware clients, but its correctness with respect to canonical URIs needs to be argued by the client, in this sequence

- understanding the has-unique-proxy line,

- understanding that the request that went to 2001:db8::1 was really a Proxy-Scheme/Uri-Host-elided version of a request to coap://myhostname, and then

- processing any relative reference with this new base in mind.

(Not that it'd need to happen in software in that sequence, but that's the sequence needed to understand how the /foo here is really coap://myhostname/foo).

If CoRAL is used during discovery, a base directive or reverse relation to has-unique-proxy would make this easier.

Appendix D. Acknowledgements

This document heavily builds on concepts explored by Bill Silverajan and Mert Ocak in [I-D.silverajan-core-coap-protocol-negotiation], and together with Ines Robles and Klaus Hartke inside T2TRG.

[TBD: reviewers Marco Klaus]

Author's Address

Christian Amsüss
Austria
Email: christian@amsuess.com

CoRE Working Group
Internet-Draft
Updates: 8613 (if approved)
Intended status: Standards Track
Expires: 28 April 2022

R. Höglund
M. Tiloca
RISE AB

25 October 2021

Key Update for OSCORE (KUDOS)
draft-hoeglund-core-oscore-key-limits-02

Abstract

Object Security for Constrained RESTful Environments (OSCORE) uses AEAD algorithms to ensure confidentiality and integrity of exchanged messages. Due to known issues allowing forgery attacks against AEAD algorithms, limits should be followed on the number of times a specific key is used for encryption or decryption. This document defines how two OSCORE peers must follow these limits and what steps they must take to preserve the security of their communications. Therefore, this document updates RFC8613. Furthermore, this document specifies Key Update for OSCORE (KUDOS), a lightweight procedure that two peers can use to update their keying material and establish a new OSCORE Security Context.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction 2
  1.1. Terminology 3
2. AEAD Key Usage Limits in OSCORE 3
  2.1. Problem Overview 3
    2.1.1. Limits for 'q' and 'v' 4
  2.2. Additional Information in the Security Context 7
    2.2.1. Common Context 7
    2.2.2. Sender Context 7
    2.2.3. Recipient Context 8
  2.3. OSCORE Messages Processing 8
    2.3.1. Protecting a Request or a Response 8
    2.3.2. Verifying a Request or a Response 9
3. Current methods for Rekeying OSCORE 9
4. Key Update for OSCORE (KUDOS) 11
  4.1. Extensions to the OSCORE Option 12
  4.2. Function for Security Context Update 13
  4.3. Establishment of the New OSCORE Security Context 15
    4.3.1. Client-Initiated Key Update 16
    4.3.2. Server-Initiated Key Update 18
  4.4. Retention Policies 21
  4.5. Discussion 21
5. Security Considerations 21
6. IANA Considerations 22
  6.1. OSCORE Flag Bits Registry 22
7. References 22
  7.1. Normative References 22
  7.2. Informative References 23
Acknowledgments 24
Authors' Addresses 24

1. Introduction

Object Security for Constrained RESTful Environments (OSCORE) [RFC8613] provides end-to-end protection of CoAP [RFC7252] messages at the application-layer, ensuring message confidentiality and integrity, replay protection, as well as binding of response to request between a sender and a recipient.

In particular, OSCORE uses AEAD algorithms to provide confidentiality and integrity of messages exchanged between two peers. Due to known issues allowing forgery attacks against AEAD algorithms, limits should be followed on the number of times a specific key is used to perform encryption or decryption [I-D.irtf-cfrg-aead-limits].

Should these limits be exceeded, an adversary may break the security properties of the AEAD algorithm, such as message confidentiality and integrity, e.g. by performing a message forgery attack. The original OSCORE specification [RFC8613] does not consider such limits.

This document updates [RFC8613] as follows.

* It defines when a peer must stop using an OSCORE Security Context shared with another peer, due to the reached key usage limits. When this happens, the two peers have to establish a new Security Context with new keying material, in order to continue their secure communication with OSCORE.

* It specifies KUDOS, a lightweight key update procedure that the two peers can use in order to update their current keying material and establish a new OSCORE Security Context. This deprecates and replaces the procedure specified in Appendix B.2 of [RFC8613].

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 related to the CoAP [RFC7252] and OSCORE [RFC8613] protocols.

2. AEAD Key Usage Limits in OSCORE

The following sections details how key usage limits for AEAD algorithms must be considered when using OSCORE. It covers specific limits for common AEAD algorithms used with OSCORE; necessary additions to the OSCORE Security Context, updates to the OSCORE message processing, and existing methods for rekeying OSCORE.

2.1. Problem Overview

The OSCORE security protocol [RFC8613] uses AEAD algorithms to provide integrity and confidentiality of messages, as exchanged between two peers sharing an OSCORE Security Context.

When processing messages with OSCORE, each peer should follow specific limits as to the number of times it uses a specific key. This applies separately to the Sender Key used to encrypt outgoing messages, and to the Recipient Key used to decrypt and verify incoming protected messages.

Exceeding these limits may allow an adversary to break the security properties of the AEAD algorithm, such as message confidentiality and integrity, e.g. by performing a message forgery attack.

The following refers to the two parameters 'q' and 'v' introduced in [I-D.irtf-cfrg-aead-limits], to use when deploying an AEAD algorithm.

* 'q': this parameter has as value the number of messages protected with a specific key, i.e. the number of times the AEAD algorithm has been invoked to encrypt data with that key.

* 'v': this parameter has as value the number of alleged forgery attempts that have been made against a specific key, i.e. the amount of failed decryptions that has been done with the AEAD algorithm for that key.

When a peer uses OSCORE:

* The key used to protect outgoing messages is its Sender Key, in its Sender Context.

* The key used to decrypt and verify incoming messages is its Recipient Key, in its Recipient Context.

Both keys are derived as part of the establishment of the OSCORE Security Context, as defined in Section 3.2 of [RFC8613].

As mentioned above, exceeding specific limits for the 'q' or 'v' value can weaken the security properties of the AEAD algorithm used, thus compromising secure communication requirements.

Therefore, in order to preserve the security of the used AEAD algorithm, OSCORE has to observe limits for the 'q' and 'v' values, throughout the lifetime of the used AEAD keys.

2.1.1. Limits for 'q' and 'v'

Formulas for calculating the security levels as Integrity Advantage (IA) and Confidentiality Advantage (CA) probabilities, are presented in [I-D.irtf-cfrg-aead-limits]. These formulas take as input specific values for 'q' and 'v' (see section Section 2.1) and for 'l', i.e., the maximum length of each message (in cipher blocks).

For the algorithms that can be used as AEAD Algorithm for OSCORE shows in Figure 1, the key property to achieve is having IA and CA values which are no larger than p = 2^-64, which will ensure a safe security level for the AEAD Algorithm. This can be entailed by using the values q = 2^20, v = 2^20, and l = 2^10, that this document recommends to use for these algorithms.

Figure 1 shows the resulting IA and CA probabilities enjoyed by the considered algorithms, when taking the value of 'q', 'v' and 'l' above as input to the formulas defined in [I-D.irtf-cfrg-aead-limits].

Algorithm name	IA probability	CA probability
AEAD_AES_128_CCM	2^-64	2^-66
AEAD_AES_128_GCM	2^-97	2^-89
AEAD_AES_256_GCM	2^-97	2^-89
AEAD_CHACHA20_POLY1305	2^-73	-

Figure 1: Probabilities for algorithms based on chosen q, v and l values.

For the AEAD_AES_128_CCM_8 algorithm when used as AEAD Algorithm for OSCORE, larger IA and CA values are achieved, depending on the value of 'q', 'v' and 'l'. Figure 2 shows the resulting IA and CA probabilities enjoyed by AEAD_AES_128_CCM_8, when taking different values of 'q', 'v' and 'l' as input to the formulas defined in [I-D.irtf-cfrg-aead-limits].

As shown in Figure 2, it is especially possible to achieve the lowest IA = 2^-54 and a good CA = 2^-70 by considering the largest possible value of the (q, v, l) triplet equal to (2^20, 2^10, 2^8), while still keeping a good security level. Note that the value of 'l' does not impact on IA, while CA displays good values for every considered value of 'l'.

When AEAD_AES_128_CCM_8 is used as AEAD Algorithm for OSCORE, this document recommends to use the triplet (q, v, l) = (2^20, 2^10, 2^8) and to never use a triplet (q, v, l) such that the resulting IA and CA probabilities are higher than 2^-54.

'q', 'v' and 'l'	IA probability	CA probability
q=2^20, v=2^20, l=2^8	2^-44	2^-70
q=2^15, v=2^20, l=2^8	2^-44	2^-80
q=2^10, v=2^20, l=2^8	2^-44	2^-90
q=2^20, v=2^15, l=2^8	2^-49	2^-70
q=2^15, v=2^15, l=2^8	2^-49	2^-80
q=2^10, v=2^15, l=2^8	2^-49	2^-90
q=2^20, v=2^14, l=2^8	2^-50	2^-70
q=2^15, v=2^14, l=2^8	2^-50	2^-80
q=2^10, v=2^14, l=2^8	2^-50	2^-90
q=2^20, v=2^10, l=2^8	2^-54	2^-70
q=2^15, v=2^10, l=2^8	2^-54	2^-80
q=2^10, v=2^10, l=2^8	2^-54	2^-90
q=2^20, v=2^20, l=2^6	2^-44	2^-74
q=2^15, v=2^20, l=2^6	2^-44	2^-84
q=2^10, v=2^20, l=2^6	2^-44	2^-94
q=2^20, v=2^15, l=2^6	2^-49	2^-74
q=2^15, v=2^15, l=2^6	2^-49	2^-84
q=2^10, v=2^15, l=2^6	2^-49	2^-94
q=2^20, v=2^14, l=2^6	2^-50	2^-74
q=2^15, v=2^14, l=2^6	2^-50	2^-84
q=2^10, v=2^14, l=2^6	2^-50	2^-94
q=2^20, v=2^10, l=2^6	2^-54	2^-74
q=2^15, v=2^10, l=2^6	2^-54	2^-84
q=2^10, v=2^10, l=2^6	2^-54	2^-94

Figure 2: Probabilities for AEAD_AES_128_CCM_8 based on chosen q, v and l values.

The algorithms using AES presented in this draft all use a block size of 16 bytes (128 bits), while AEAD_CHACHA20_POLY1305 uses a block size of 64 bytes (512 bits). As 'l' is defined as the maximum size of each message in blocks, different block sizes will result in different maximum messages sizes for the same value of 'l'. Figure 3 presents the resulting maximum message size in bytes for the different algorithms and values of 'l' presented in this document.

Algorithm name	l=2^6 in bytes	l=2^8 in bytes	l=2^10 in bytes
AEAD_AES_128_CCM	1024	4096	16384
AEAD_AES_128_GCM	1024	4096	16384
AEAD_AES_256_GCM	1024	4096	16384
AEAD_AES_128_CCM_8	1024	4096	16384
AEAD_CHACHA20_POLY1305	4096	16384	65536

Figure 3: Maximum length of each message (in bytes)

2.2. Additional Information in the Security Context

In addition to what defined in Section 3.1 of [RFC8613], the OSCORE Security Context MUST also include the following information.

2.2.1. Common Context

The Common Context is extended to include the following parameter.

* 'exp': with value the expiration time of the OSCORE Security Context, as a non-negative integer. The parameter 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 for NumericDate in Section 2 of [RFC7519].

  At the time indicated in this field, a peer MUST stop using this Security Context to process any incoming or outgoing message, and is required to establish a new Security Context to continue OSCORE-protected communications with the other peer.

2.2.2. Sender Context

The Sender Context is extended to include the following parameters.

* 'count_q': a non-negative integer counter, keeping track of the current 'q' value for the Sender Key. At any time, 'count_q' has as value the number of messages that have been encrypted using the Sender Key. The value of 'count_q' is set to 0 when establishing the Sender Context.

* 'limit_q': a non-negative integer, which specifies the highest value that 'count_q' is allowed to reach, before stopping using the Sender Key to process outgoing messages.

The value of 'limit_q' depends on the AEAD algorithm specified in the Common Context, considering the properties of that algorithm. The value of 'limit_q' is determined according to Section 2.1.1.

2.2.3. Recipient Context

The Recipient Context is extended to include the following parameters.

* 'count_v': a non-negative integer counter, keeping track of the current 'v' value for the Recipient Key. At any time, 'count_v' has as value the number of failed decryptions occurred on incoming messages using the Recipient Key. The value of 'count_v' is set to 0 when establishing the Recipient Context.

* 'limit_v': a non-negative integer, which specifies the highest value that 'count_v' is allowed to reach, before stopping using the Recipient Key to process incoming messages.

  The value of 'limit_v' depends on the AEAD algorithm specified in the Common Context, considering the properties of that algorithm. The value of 'limit_v' is determined according to Section 2.1.1.

2.3. OSCORE Messages Processing

In order to keep track of the 'q' and 'v' values and ensure that AEAD keys are not used beyond reaching their limits, the processing of OSCORE messages is extended as defined in this section. A limitation that is introduced is that, in order to not exceed the selected value for 'l', the total size of the COSE plaintext, authentication Tag, and possible cipher padding for a message may not exceed the block size for the selected algorithm multiplied with 'l'.

In particular, the processing of OSCORE messages follows the steps outlined in Section 8 of [RFC8613], with the additions defined below.

2.3.1. Protecting a Request or a Response

Before encrypting the COSE object using the Sender Key, the 'count_q' counter MUST be incremented.

If 'count_q' exceeds the 'limit_q' limit, the message processing MUST be aborted. From then on, the Sender Key MUST NOT be used to encrypt further messages.

2.3.2. Verifying a Request or a Response

If an incoming message is detected to be a replay (see Section 7.4 of [RFC8613]), the 'count_v' counter MUST NOT be incremented.

If the decryption and verification of the COSE object using the Recipient Key fails, the 'count_v' counter MUST be incremented.

After 'count_v' has exceeded the 'limit_v' limit, incoming messages MUST NOT be decrypted and verified using the Recipient Key, and their processing MUST be aborted.

3. Current methods for Rekeying OSCORE

Before the limit of 'q' or 'v' defined in Section 2.1.1 has been reached for an OSCORE Security Context, the two peers have to establish a new OSCORE Security Context, in order to continue using OSCORE for secure communication.

In practice, the two peers have to establish new Sender and Recipient Keys, as the keys actually used by the AEAD algorithm. When this happens, both peers reset their 'count_q' and 'count_v' values to 0 (see Section 2.2).

Other specifications define a number of ways to accomplish this, as summarized below.

* The two peers can run the procedure defined in Appendix B.2 of [RFC8613]. That is, the two peers exchange three or four messages, protected with temporary Security Contexts adding randomness to the ID Context.

  As a result, the two peers establish a new OSCORE Security Context with new ID Context, Sender Key and Recipient Key, while keeping the same OSCORE Master Secret and OSCORE Master Salt from the old OSCORE Security Context.

  This procedure does not require any additional components to what OSCORE already provides, and it does not provide perfect forward secrecy.

  The procedure defined in Appendix B.2 of [RFC8613] is used in 6TiSCH networks [RFC7554][RFC8180] when handling failure events. That is, a node acting as Join Registrar/Coordinator (JRC) assists new devices, namely "pledges", to securely join the network as per the Constrained Join Protocol [RFC9031]. In particular, a pledge exchanges OSCORE-protected messages with the JRC, from which it obtains a short identifier, link-layer keying material and other

configuration parameters. As per Section 8.3.3 of [RFC9031], a JRC that experiences a failure event may likely lose information about joined nodes, including their assigned identifiers. Then, the reinitialized JRC can establish a new OSCORE Security Context with each pledge, through the procedure defined in Appendix B.2 of [RFC8613].

* The two peers can run the OSCORE profile [I-D.ietf-ace-oscore-profile] of the Authentication and Authorization for Constrained Environments (ACE) Framework [I-D.ietf-ace-oauth-authz].

  When a CoAP client uploads an Access Token to a CoAP server as an access credential, the two peers also exchange two nonces. Then, the two peers use the two nonces together with information provided by the ACE Authorization Server that issued the Access Token, in order to derive an OSCORE Security Context.

  This procedure does not provide perfect forward secrecy.

* The two peers can run the EDHOC key exchange protocol based on Diffie-Hellman and defined in [I-D.ietf-lake-edhoc], in order to establish a pseudo-random key in a mutually authenticated way.

  Then, the two peers can use the established pseudo-random key to derive external application keys. This allows the two peers to securely derive especially an OSCORE Master Secret and an OSCORE Master Salt, from which an OSCORE Security Context can be established.

  This procedure additionally provides perfect forward secrecy.

* If one peer is acting as LwM2M Client and the other peer as LwM2M Server, according to the OMA Lightweight Machine to Machine Core specification [LwM2M], then the LwM2M Client peer may take the initiative to bootstrap again with the LwM2M Bootstrap Server, and receive again an OSCORE Security Context. Alternatively, the LwM2M Server can instruct the LwM2M Client to initiate this procedure.

  If the OSCORE Security Context information on the LwM2M Bootstrap Server has been updated, the LwM2M Client will thus receive a fresh OSCORE Security Context to use with the LwM2M Server.

In addition to that, the LwM2M Client, the LwM2M Server as well as the LwM2M Bootstrap server are required to use the procedure defined in Appendix B.2 of [RFC8613] and overviewed above, when they use a certain OSCORE Security Context for the first time [LwM2M-Transport].

Manually updating the OSCORE Security Context at the two peers should be a last resort option, and it might often be not practical or feasible.

Even when any of the alternatives mentioned above is available, it is RECOMMENDED that two OSCORE peers update their Security Context by using the KUDOS procedure as defined in Section 4 of this document.

It is RECOMMENDED that the peer initiating the key update procedure starts it before reaching the 'q' or 'v' limits. Otherwise, the AEAD keys possibly to be used during the key update procedure itself may already be or become invalid before the rekeying is completed, which may prevent a successful establishment of the new OSCORE Security Context altogether.

4. Key Update for OSCORE (KUDOS)

This section defines KUDOS, a lightweight procedure that two OSCORE peers can use to update their keying material and establish a new OSCORE Security Context.

KUDOS relies on the support function updateCtx() defined in Section 4.2 and the message exchange defined in Section 4.3. The following properties are fulfilled.

* KUDOS can be initiated by either peer. In particular, the client or the server may start KUDOS by sending the first rekeying message.

* The new OSCORE Security Context enjoys Perfect Forward Secrecy.

* The same ID Context value used in the old OSCORE Security Context is preserved in the new Security Context. Furthermore, the ID Context value never changes throughout the KUDOS execution.

* KUDOS is robust against a peer rebooting, and it especially avoids the reuse of AEAD (nonce, key) pairs.

* KUDOS completes in one round trip. The two peers achieve mutual proof-of-possession in the following exchange, which is protected with the newly established OSCORE Security Context.

4.1. Extensions to the OSCORE Option

In order to support the message exchange for establishing a new OSCORE Security Context as defined in Section 4.3, this document extends the use of the OSCORE option originally defined in [RFC8613] as follows.

* This document defines the usage of the seventh least significant bit, called "Extension-1 Flag", in the first byte of the OSCORE option containing the OSCORE flag bits. This flag bit is specified in Section 6.1.

  When the Extension-1 Flag is set to 1, the second byte of the OSCORE option MUST include the set of OSCORE flag bits 8-15.

* This document defines the usage of the first least significant bit "ID Detail Flag", 'd', in the second byte of the OSCORE option containing the OSCORE flag bits. This flag bit is specified in Section 6.1.

  When it is set to 1, the compressed COSE object contains an 'id detail', to be used for the steps defined in Section 4.3. In particular, the 1 byte following 'kid context' (if any) encodes the length x of 'id detail', and the following x bytes encode 'id detail'.

* The second-to-eighth least significant bits in the second byte of the OSCORE option containing the OSCORE flag 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 of [RFC8613].

Figure 4 shows the OSCORE option value including also 'id detail'.

```
 0 1 2 3 4 5 6 7  8   9   10  11  12  13  14  15 <----- n bytes ----->
+-+-+-+-+-+-+-+-+---+---+---+---+---+---+---+---+---------------------+
|0|1|0|h|k|  n  | 0 | 0 | 0 | 0 | 0 | 0 | 0 | d | Partial IV (if any) |
+-+-+-+-+-+-+-+-+---+---+---+---+---+---+---+---+---------------------+

 <- 1 byte -> <----- s bytes ------> <- 1 byte -> <----- x bytes ---->
+------------+----------------------+-----------------------------------+
| s (if any) | kid context (if any) | x (if any) | id detail (if any) |
+------------+----------------------+------------+--------------------+

+------------------+
| kid (if any) ... |
+------------------+
```

Figure 4: The OSCORE option value, including 'id detail'

4.2. Function for Security Context Update

The updateCtx() function shown in Figure 5 takes as input a nonce N as well as an OSCORE Security Context CTX_IN, and returns as output a new OSCORE Security Context CTX_OUT.

As a first step, the updateCtx() function derives the new values of the Master Secret and Master Salt for CTX_OUT, according to one of the two following methods. The used method depends on how the two peers established their original Security Context, i.e., the Security Context that they shared before performing KUDOS with one another for the first time.

* If the original Security Context was established by running the EDHOC protocol [I-D.ietf-lake-edhoc], the following applies.

  First, the EDHOC key PRK_4x3m shared by the two peers is updated using the EDHOC-KeyUpdate() function defined in Section 4.4 of [I-D.ietf-lake-edhoc], which takes the nonce N as input.

  After that, the EDHOC-Exporter() function defined in Section 4.3 of [I-D.ietf-lake-edhoc] is used to derive the new values for the Master Secret and Master Salt, consistently with what is defined in Appendix A.2 of [I-D.ietf-lake-edhoc]. In particular, the context parameter provided as second argument to the EDHOC-Exporter() function is the empty CBOR byte string (0x40) [RFC8949], which is denoted as h''.

  Note that, compared to the compliance requirements in Section 7 of [I-D.ietf-lake-edhoc], a peer MUST support the EDHOC-KeyUpdate() function, in case it establishes an original Security Context through the EDHOC protocol and intends to perform KUDOS.

* If the original Security Context was established through other means than the EDHOC protocol, the new Master Secret is derived through an HKDF-Expand() step, which takes as input N as well as the Master Secret value from the Security Context CTX_IN. Instead, the new Master Salt takes N as value.

In either case, the derivation of new values follows the same approach used in TLS 1.3, which is also based on HKDF-Expand (see Section 7.1 of [RFC8446]) and used for computing new keying material in case of key update (see Section 4.6.3 of [RFC8446]).

After that, the new Master Secret and Master Salt parameters are used to derive a new Security Context CTX_OUT as per Section 3.2 of [RFC8613]. Any other parameter required for the derivation takes the same value as in the Security Context CTX_IN. Finally, the function returns the newly derived Security Context CTX_OUT.

```
updateCtx(N, CTX_IN) {

  CTX_OUT       // The new Security Context
  MSECRET_NEW   // The new Master Secret
  MSALT_NEW     // The new Master Salt

  if <the original Security Context was established through EDHOC> {

    EDHOC-KeyUpdate(N)
    // This results in updating the key PRK_4x3m of the
    // EDHOC session, i.e., PRK_4x3m = Extract(N, PRK_4x3m)

    MSECRET_NEW = EDHOC-Exporter("OSCORE_Master_Secret",
                                 h'', key_length)
      = EDHOC-KDF(PRK_4x3m, TH_4,
                  "OSCORE_Master_Secret", h'', key_length)

    MSALT_NEW = EDHOC-Exporter("OSCORE_Master_Salt",
                               h'', salt_length)
      = EDHOC-KDF(PRK_4x3m, TH_4,
                  "OSCORE_Master_Salt", h'', salt_length)

  }
  else {
    Master Secret Length = < Size of CTX_IN.MasterSecret in bytes >

    MSECRET_NEW = HKDF-Expand-Label(CTX_IN.MasterSecret, Label,
                                    N, Master Secret Length)
                = HKDF-Expand(CTX_IN.MasterSecret, HkdfLabel,
                              Master Secret Length)

    MSALT_NEW = N;
  }

  < Derive CTX_OUT using MSECRET_NEW and MSALT_NEW,
    together with other parameters from CTX_IN >

  Return CTX_OUT;

}
```

Where HkdfLabel is defined as

```
struct {
    uint16 length = Length;
    opaque label<7..255> = "oscore " + Label;
    opaque context<0..255> = Context;
} HkdfLabel;
```

Figure 5: Function for deriving a new OSCORE Security Context

4.3. Establishment of the New OSCORE Security Context

This section defines the actual KUDOS procedure performed by two peers to update their OSCORE keying material. Before starting KUDOS, the two peers share the OSCORE Security Context CTX_OLD. Once completed the KUDOS execution, the two peers agree on a newly established OSCORE Security Context CTX_NEW.

In particular, each peer contributes by generating a fresh value R1 or R2, and providing it to the other peer. The byte string concatenation of the two values, hereafter denoted as R1 | R2, is used as input N by the updateCtx() function, in order to derive the new OSCORE Security Context CTX_NEW. As for any new OSCORE Security Context, the Sender Sequence Number and the replay window are re-initialized accordingly (see Section 3.2.2 of [RFC8613]).

Once a peer has successfully derived the new OSCORE Security Context CTX_NEW, that peer MUST terminate all the ongoing observations it has with the other peer as protected with the old Security Context CTX_OLD.

Once a peer has successfully decrypted and verified an incoming message protected with CTX_NEW, that peer MUST discard the old Security Context CTX_OLD.

KUDOS can be started by the client or the server, as defined in Section 4.3.1 and Section 4.3.2, respectively. The following properties hold for both the client- and server-initiated version of KUDOS.

* The initiator always offers the fresh value R1.

* The responder always offers the fresh value R2.

* The responder is always the first one deriving the new OSCORE Security Context CTX_NEW.

* The initiator is always the first one achieving key confirmation, hence able to safely discard the old OSCORE Security Context CTX_OLD.

* Both the initiator and the responder use the same respective OSCORE Sender ID and Recipient ID. Also, they both preserve and use the same OSCORE ID Context from CTX_OLD.

The length of the nonces R1, and R2 is application specific. The application needs to set the length of each nonce such that the probability of its value being repeated is negligible; typically, at least 8 bytes long.

4.3.1. Client-Initiated Key Update

Figure 6 shows the KUDOS workflow with the client acting as initiator.

```
                     Client               Server
                  (initiator)          (responder)
                             |                    |
  Generate R1                |                    |
                             |                    |
  CTX_1 =                    |                    |
    updateCtx(R1,            |                    |
              CTX_OLD)       |                    |
                             |                    |
                             |     Request #1     |
  Protect with CTX_1         |------------------->|
                             | OSCORE Option:     | CTX_1 =
                             |   ...              |   update(R1,
                             |   d flag: 1        |          CTX_OLD)
                             |   ...              |
                             |   ID Detail: R1    | Verify with CTX_1
                             |   ...              |
                             |                    | Generate R2
                             |                    |
                             |                    | CTX_NEW =
                             |                    |   update(R1|R2,
                             |                    |          CTX_OLD)
                             |                    |
                             |     Response #1    |
                             |<-------------------| Protect with CTX_NEW
  CTX_NEW =                  | OSCORE Option:     |
    updateCtx(R1|R2,         |   ...              |
              CTX_OLD)       |   d flag: 1        |
                             |   ...              |
  Verify with CTX_NEW        |   ID Detail: R2    |
                             |   ...              |
  Discard CTX_OLD            |                    |
                             |                    |
```

```
     // The actual key update process ends here.
     // The two peers can use the new Security Context CTX_NEW.

                             |                    |
                             |     Request #2     |
     Protect with CTX_NEW    |------------------->|
                             |                    | Verify with CTX_NEW
                             |                    |
                             |                    | Discard CTX_OLD
                             |                    |
                             |     Response #2    |
                             |<-------------------| Protect with CTX_NEW
     Verify with CTX_NEW     |                    |
                             |                    |
```

Figure 6: Client-Initiated KUDOS Workflow

First, the client generates a random value R1, and uses the nonce N = R1 together with the old Security Context CTX_OLD, in order to derive a temporary Security Context CTX_1. Then, the client sends an OSCORE request to the server, protected with the Security Context CTX_1. In particular, the request has the 'd' flag bit set to 1 and specifies R1 as 'id detail' (see Section 4.1).

Upon receiving the OSCORE request, the server retrieves the value R1 from the 'id detail' of the request, and uses the nonce N = R1 together with the old Security Context CTX_OLD, in order to derive the temporary Security Context CTX_1. Then, the server verifies the request by using the Security Context CTX_1.

After that, the server generates a random value R2, and uses the nonce N = R1 | R2 together with the old Security Context CTX_OLD, in order to derive the new Security Context CTX_NEW. Then, the server sends an OSCORE response to the client, protected with the new Security Context CTX_NEW. In particular, the response has the 'd' flag bit set to 1 and specifies R2 as 'id detail'.

Upon receiving the OSCORE response, the client retrieves the value R2 from the 'id detail' of the response. Since the client has received a response to an OSCORE request it made with the 'd' flag bit set to 1, the client uses the nonce N = R1 | R2 together with the old Security Context CTX_OLD, in order to derive the new Security Context CTX_NEW. Finally, the client verifies the response by using the Security Context CTX_NEW and deletes the old Security Context CTX_OLD.

After that, the client can send a new OSCORE request protected with the new Security Context CTX_NEW. When successfully verifying the request using the Security Context CTX_NEW, the server deletes the old Security Context CTX_OLD and can reply with an OSCORE response protected with the new Security Context CTX_NEW.

From then on, the two peers can protect their message exchanges by using the new Security Context CTX_NEW.

4.3.2. Server-Initiated Key Update

Figure 7 shows the KUDOS workflow with the server acting as initiator.

```
                        Client               Server
                     (responder)          (initiator)
                          |                    |
                          |     Request #1     |
     Protect with CTX_OLD |------------------->|
                          |                    | Verify with CTX_OLD
                          |                    |
                          |                    | Generate R1
                          |                    |
                          |                    | CTX_1 =
                          |                    |   updateCtx(R1,
                          |                    |             CTX_OLD)
                          |                    |
                          |     Response #1    |
                          |<-------------------| Protect with CTX_1
     CTX_1 =              | OSCORE Option:     |
       updateCtx(R1,      |   ...              |
                 CTX_OLD) |   d flag: 1        |
                          |   ...              |
     Verify with CTX_1    |   ID Detail: R1    |
                          |   ...              |
     Generate R2          |                    |
                          |                    |
     CTX_NEW =            |                    |
       updateCtx(R1|R2,   |                    |
                 CTX_OLD) |                    |
                          |                    |
                          |     Request #2     |
     Protect with CTX_NEW |------------------->|
                          | OSCORE Option:     | CTX_NEW =
                          |   ...              |   updateCtx(R1|R2,
                          |   d flag: 1        |             CTX_OLD)
                          |   ...              |
                          |   ID Detail: R1|R2  | Verify with CTX_NEW
                          |   ...              |
                          |                    | Discard CTX_OLD
                          |                    |

     // The actual key update process ends here.
     // The two peers can use the new Security Context CTX_NEW.

                          |     Response #2    |
                          |<-------------------| Protect with CTX_NEW
     Verify with CTX_NEW  |                    |
                          |                    |
     Discard CTX_OLD      |                    |
                          |                    |
```

Figure 7: Server-Initiated KUDOS Workflow

First, the client sends a normal OSCORE request to the server, protected with the old Security Context CTX_OLD and with the 'd' flag bit set to 0.

Upon receiving the OSCORE request and after having verified it with the old Security Context CTX_OLD as usual, the server generates a random value R1 and uses the nonce N = R1 together with the old Security Context CTX_OLD, in order to derive a temporary Security Context CTX_1. Then, the server sends an OSCORE response to the client, protected with the Security Context CTX_1. In particular, the response has the 'd' flag bit set to 1 and specifies R1 as 'id detail' (see Section 4.1).

Upon receiving the OSCORE response, the client retrieves the value R1 from the 'id detail' of the response, and uses the nonce N = R1 together with the old Security Context CTX_OLD, in order to derive the temporary Security Context CTX_1. Then, the client verifies the response by using the Security Context CTX_1.

After that, the client generates a random value R2, and uses the nonce N = R1 | R2 together with the old Security Context CTX_OLD, in order to derive the new Security Context CTX_NEW. Then, the client sends an OSCORE request to the server, protected with the new Security Context CTX_NEW. In particular, the request has the 'd' flag bit set to 1 and specifies R1 | R2 as 'id detail'.

Upon receiving the OSCORE request, the server retrieves the value R1 | R2 from the request. Then, the server verifies that: i) the value R1 is identical to the value R1 specified in a previous OSCORE response with the 'd' flag bit set to 1; and ii) the value R1 | R2 has not been received before in an OSCORE request with the 'd' flag bit set to 1. If the verification succeeds, the server uses the nonce N = R1 | R2 together with the old Security Context CTX_OLD, in order to derive the new Security Context CTX_NEW. Finally, the server verifies the request by using the Security Context CTX_NEW and deletes the old Security Context CTX_OLD.

After that, the server can send an OSCORE response protected with the new Security Context CTX_NEW. When successfully verifying the response using the Security Context CTX_NEW, the client deletes the old Security Context CTX_OLD.

From then on, the two peers can protect their message exchanges by using the new Security Context CTX_NEW.

4.4. Retention Policies

Applications MAY define policies that allows a peer to also temporarily keep the old Security Context CTX_OLD, rather than simply overwriting it to become CTX_NEW. This allows the peer to decrypt late, still on-the-fly incoming messages protected with CTX_OLD.

When enforcing such policies, the following applies.

* Outgoing messages MUST be protected by using only CTX_NEW.

* Incoming messages MUST first be attempted to decrypt by using CTX_NEW. If decryption fails, a second attempt can use CTX_OLD.

* When an amount of time defined by the policy has elapsed since the establishment of CTX_NEW, the peer deletes CTX_OLD.

4.5. Discussion

KUDOS is intended to deprecate and replace the procedure defined in Appendix B.2 of [RFC8613], as fundamentally achieving the same goal, while displaying a number of improvements and advantages.

In particular, it is especially convenient for the handling of failure events concerning the JRC node in 6TiSCH networks (see Section 3). That is, among its intrinsic advantages compared to the procedure defined in Appendix B.2 of [RFC8613], KUDOS preserves the same ID Context value, when establishing a new OSCORE Security Context.

Since the JRC uses ID Context values as identifiers of network nodes, namely "pledge identifiers", the above implies that the JRC does not have anymore to perform a mapping between a new, different ID Context value and a certain pledge identifier (see Section 8.3.3 of [RFC9031]). It follows that pledge identifiers can remain constant once assigned, and thus ID Context values used as pledge identifiers can be employed in the long-term as originally intended.

5. Security Considerations

This document mainly covers security considerations about using AEAD keys in OSCORE and their usage limits, in addition to the security considerations of [RFC8613].

Depending on the specific key update procedure used to establish a new OSCORE Security Context, the related security considerations also apply.

TODO: Add more considerations.

6. IANA Considerations

This document has the following actions for IANA.

6.1. OSCORE Flag Bits Registry

IANA is asked to add the following entries to the "OSCORE Flag Bits" registry within the "Constrained RESTful Environments (CoRE) Parameters" registry group.

Bit Position	Name	Description	Reference
1	Extension-1 Flag	Set to 1 if the OSCORE Option specifies a second byte of OSCORE flag bits	[This Document]
15	ID Detail Flag	Set to 1 if the compressed COSE object contains 'id detail'	[This Document]

7. References

7.1. Normative References

[I-D.ietf-lake-edhoc]
Selander, G., Mattsson, J. P., and F. Palombini, "Ephemeral Diffie-Hellman Over COSE (EDHOC)", Work in Progress, Internet-Draft, draft-ietf-lake-edhoc-12, 20 October 2021, <https://www.ietf.org/archive/id/draft-ietf-lake-edhoc-12.txt>.

[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>.

[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>.

[RFC8613] Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security for Constrained RESTful Environments (OSCORE)", RFC 8613, DOI 10.17487/RFC8613, July 2019, <https://www.rfc-editor.org/info/rfc8613>.

[RFC8949] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", STD 94, RFC 8949, DOI 10.17487/RFC8949, December 2020, <https://www.rfc-editor.org/info/rfc8949>.

7.2. Informative 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)", Work in Progress, Internet-Draft, draft-ietf-ace-oauth-authz-45, 29 August 2021, <https://www.ietf.org/archive/id/draft-ietf-ace-oauth-authz-45.txt>.

[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", Work in Progress, Internet-Draft, draft-ietf-ace-oscore-profile-19, 6 May 2021, <https://www.ietf.org/archive/id/draft-ietf-ace-oscore-profile-19.txt>.

[I-D.irtf-cfrg-aead-limits]
Günther, F., Thomson, M., and C. A. Wood, "Usage Limits on AEAD Algorithms", Work in Progress, Internet-Draft, draft-irtf-cfrg-aead-limits-03, 12 July 2021, <https://www.ietf.org/archive/id/draft-irtf-cfrg-aead-limits-03.txt>.

[LwM2M] Open Mobile Alliance, "Lightweight Machine to Machine Technical Specification - Core, Approved Version 1.2, OMA-TS-LightweightM2M_Core-V1_2-20201110-A", November 2020, <http://www.openmobilealliance.org/release/LightweightM2M/V1_2-20201110-A/OMA-TS-LightweightM2M_Core-V1_2-20201110-A.pdf>.

[LwM2M-Transport]
Open Mobile Alliance, "Lightweight Machine to Machine Technical Specification - Transport Bindings, Approved Version 1.2, OMA-TS-LightweightM2M_Transport-V1_2-20201110-A", November 2020, <http://www.openmobilealliance.org/release/LightweightM2M/V1_2-20201110-A/OMA-TS-LightweightM2M_Transport-V1_2-20201110-A.pdf>.

[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>.

[RFC7554] Watteyne, T., Ed., Palattella, M., and L. Grieco, "Using IEEE 802.15.4e Time-Slotted Channel Hopping (TSCH) in the Internet of Things (IoT): Problem Statement", RFC 7554, DOI 10.17487/RFC7554, May 2015, <https://www.rfc-editor.org/info/rfc7554>.

[RFC8180] Vilajosana, X., Ed., Pister, K., and T. Watteyne, "Minimal IPv6 over the TSCH Mode of IEEE 802.15.4e (6TiSCH) Configuration", BCP 210, RFC 8180, DOI 10.17487/RFC8180, May 2017, <https://www.rfc-editor.org/info/rfc8180>.

[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>.

[RFC9031] Vuini, M., Ed., Simon, J., Pister, K., and M. Richardson, "Constrained Join Protocol (CoJP) for 6TiSCH", RFC 9031, DOI 10.17487/RFC9031, May 2021, <https://www.rfc-editor.org/info/rfc9031>.

Acknowledgments

The authors sincerely thank Christian Amsuess, John Mattsson and Goeran Selander 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 H2020 project SIFIS-Home (Grant agreement 952652).

Authors' Addresses

Rikard Höglund
RISE AB
Isafjordsgatan 22
SE-16440 Stockholm Kista
Sweden

Email: rikard.hoglund@ri.se

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

Email: marco.tiloca@ri.se

CoRE Working Group
Internet-Draft
Intended status: Informational
Expires: 18 July 2023

M. Koster
Dogtiger Labs
A. Soloway
Qualcomm Technologies, Inc.
B. Silverajan, Ed.
Tampere University
14 January 2023

Conditional Attributes for Constrained RESTful Environments
draft-ietf-core-conditional-attributes-06

Abstract

This specification defines Conditional Notification and Control Attributes that work with CoAP Observe (RFC7641).

Editor note

The git repository for the draft is found at https://github.com/core-wg/conditional-attributes/

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction 2
2. Terminology 3
3. Conditional Attributes 3
3.1. Conditional Notification Attributes 3
3.1.1. Greater Than (c.gt) 4
3.1.2. Less Than (c.lt) 4
3.1.3. Change Step (c.st) 5
3.1.4. Notification Band (c.band) 5
3.1.5. Edge (c.edge) 6
3.2. Conditional Control Attributes 7
3.2.1. Minimum Period (c.pmin) 7
3.2.2. Maximum Period (c.pmax) 8
3.2.3. Minimum Evaluation Period (c.epmin) 8
3.2.4. Maximum Evaluation Period (c.epmax) 8
3.2.5. Confirmable Notification (c.con) 8
3.3. Server processing of Conditional Attributes 9
4. Implementation Considerations 9
5. Security Considerations 10
6. IANA Considerations 10
7. Acknowledgements 11
8. Contributors 11
9. Changelog 11
10. Normative References 12
Appendix A. Pseudocode: Processing Conditional Attributes . . . 13
Appendix B. Examples 14
B.1. Minimum Period (c.pmin) example 15
B.2. Maximum Period (c.pmax) example 15
B.3. Greater Than (c.gt) example 17
B.4. Greater Than (c.gt) and Period Max (c.pmax) example . . . 17
Authors' Addresses 19

1. Introduction

IETF Standards for machine-to-machine communication in constrained environments describe the Constrained Application Protocol (CoAP) [RFC7252], a RESTful application protocol, as well as a set of related information standards that may be used to represent machine data and machine metadata in REST interfaces.

This specification defines Conditional Notification and Control Attributes for use with CoAP 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 [RFC7252] and [RFC7641]. This specification makes use of the following additional terminology:

Notification Band: A resource value range that may be bounded by a minimum and maximum value or may be unbounded having either a minimum or maximum value.

3. Conditional Attributes

This specification defines conditional attributes for use with CoRE Observe [RFC7641]. Conditional attributes provide fine-grained control of notification and synchronization of resource states. When observing a resource, a CoAP client conveys conditional attributes as metadata using the query component of a CoAP URI. A conditional attribute can be represented as a "name=value" query parameter or simply a "name" without a value. Multiple conditional attributes in a query component are separated with an ampersand "&". A resource marked as Observable in its link description SHOULD support these conditional attributes.

Note: In this draft, we assume that there are finite quantization effects in the internal or external updates to the value representing the state of a resource; specifically, that a resource state may be updated at any time with any valid value. We therefore avoid any continuous-time assumptions in the description of the conditional 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. Conditional Notification Attributes

Conditional Notification Attributes define the conditions that trigger a notification. 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.

The set of Conditional Notification Attributes defined here allows a client to control how often a notification is received and how much a representation state should change in order to trigger a notification. One or more Conditional Notification Attributes MAY be included in an Observe request.

Conditional Notification Attributes are defined below:

Attribute	Name	Value
Greater Than	c.gt	xs:decimal
Less Than	c.lt	xs:decimal
Change Step	c.st	xs:decimal (>0)
Notification Band	c.band	(none)
Edge	c.edge	xs:boolean

Table 1: Conditional Notification Attributes

3.1.1. Greater Than (c.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 "c.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 "c.gt". If the value drops below the upper limit value then a notification is sent, subject again to the "c.pmin" time.

The Greater Than parameter can only be supported on resources with a scalar numeric value.

3.1.2. Less Than (c.lt)

When present, Less Than indicates the lower limit value the resource value SHOULD cross before triggering a notification. A notification is sent whenever the sampled value crosses the specified lower limit value, relative to the last reported value, and the time for "c.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 "c.lt". If the value rises above the

lower limit value then a new notification is sent, subject to the "c.pmin" time.

The Less Than parameter can only be supported on resources with a scalar numeric value.

3.1.3. Change Step (c.st)

When present, Change step indicates how much the value representing a resource state SHOULD change before triggering a notification, compared to the previous resource state. Upon reception of a query including the "c.st" attribute, the current resource state representing the most recently sampled value is reported, and then set as the last reported value (last_rep_v). When a subsequent sampled value or update of the resource state differs from the last reported state by an amount, positive or negative, greater than or equal to st, and the time for "c.pmin" has elapsed since the last notification, a notification is sent and the last reported value is updated to the new resource state 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. "c.pmin", the change in resource states received in two sequential notifications may differ by more than "c.st".

3.1.4. Notification Band (c.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 result in differing behaviour. For example, in monitoring the temperature of machinery, whilst the temperature is in the normal operating range, only periodic updates are needed. However as the temperature moves to more abnormal ranges more frequent state updates may be sent to clients.

Without a notification band, a transition across a Less Than (c.lt), or Greater Than (c.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 "c.band" attribute works as a modifier to the behaviour of "c.gt" and "c.lt". Its use is determined only by its presence, and not its value. Therefore, if "c.band" is present in a query, "c.gt", "c.lt" or both, MUST be included.

When "c.band" is present with "c.lt" but without "c.gt", the lower bound for the notification band (notification band minimum) is defined. Notifications occur when the resource value is equal to or above the notification band minimum. No maximum values exist for the band.

When "c.band" is present with "c.gt" but without "c.lt", the upper bound for the notification band (notification band maximum) is defined. Notifications occur when the resource value is equal to or below the notification band maximum. No minimum values exist for the band.

If "c.band" is specified in which the value of "c.gt" is less than that of "c.lt", in-band notification occurs. That is, notification occurs whenever the resource value is between the "c.gt" and "c.lt" values, including equal to "c.gt" or "c.lt".

If "c.band" is specified in which the value of "c.gt" is greater than that of "c.lt", out-of-band notification occurs. That is, notification occurs when the resource value not between the "c.gt" and "c.lt" values, excluding equal to "c.gt" and "c.lt".

The Notification Band parameter can only be supported on resources with a scalar numeric value.

3.1.5. Edge (c.edge)

When present, the Edge attribute indicates interest for receiving notifications of either the falling edge or the rising edge transition of a boolean resource state. When the value of the "c.edge" attribute is 0 (False), the server notifies the client each time a resource state changes from True to False. When the value of the "c.edge" attribute is 1 (True), the server notifies the client each time a resource state changes from False to True.

The "c.edge" attribute can only be supported on resources with a boolean value.

3.2. Conditional Control Attributes

Conditional Control Attributes define the time intervals between consecutive notifications as well as the cadence of the evaluation of the conditions that trigger a notification. Conditional Control Attributes can be used to configure the internal server-driven sampling process for performing evaluations of the conditions of a resource. One or more Conditional Control Attributes MAY be included in an Observe request.

Conditional Control Attributes are defined below:

Attribute	Name	Value
Minimum Period (s)	c.pmin	xs:decimal (>0)
Maximum Period (s)	c.pmax	xs:decimal (>0)
Minimum Evaluation Period (s)	c.epmin	xs:decimal (>0)
Maximum Evaluation Period (s)	c.epmax	xs:decimal (>0)
Confirmable Notification	c.con	xs:boolean

Table 2: Conditional Control Attributes

3.2.1. Minimum Period (c.pmin)

When present, Minimum Period indicates the minimum time, in seconds, between two consecutive notifications (whether or not the resource state has changed). In the absence of this parameter, the minimum period is up to the server. 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 update the resource state with the last sampled value that occurred during the "c.pmin" interval, after the "c.pmin" interval expires.

Note: due to finite quantization effects, the time between notifications may be greater than "c.pmin" even when the sampled value changes within the "c.pmin" interval. "c.pmin" may or may not be used to drive the internal sampling process.

3.2.2. Maximum Period (c.pmax)

When present, Maximum Period indicates the maximum time, in seconds, between two consecutive notifications (regardless of whether or not the resource state has changed). In the absence of this parameter, the maximum period is up to the server. Maximum Period MUST be greater than zero and MUST be greater than or equal to Minimum Period (if present), otherwise the receiver MUST return a CoAP error code 4.00 "Bad Request" (or equivalent).

3.2.3. Minimum Evaluation Period (c.epmin)

When present, Minimum Evaluation Period indicates the minimum time, in seconds, the client recommends to the server to wait between two consecutive evaluations of the conditions of a resource, since the client has no interest in the server doing more frequent evaluations. When the value of Minimum Evaluation Period expires after the previous evaluation, the server MAY immediately perform a new evaluation. In the absence of this parameter, the minimum evaluation period is not defined and thus not used by the server. The server MAY use "c.pmin", if defined, as a guidance on the desired evaluation cadence. Minimum Evaluation Period MUST be greater than zero, otherwise the receiver MUST return a CoAP error code 4.00 "Bad Request" (or equivalent).

3.2.4. Maximum Evaluation Period (c.epmax)

When present, Maximum Evaluation Period indicates the maximum time, in seconds, the server MAY wait between two consecutive evaluations of the conditions of a resource. When the value of Maximum Evaluation Period expires after the previous evaluation, the server MUST immediately perform a new evaluation. In the absence of this parameter, the maximum evaluation period is not defined and thus not used by the server. Maximum Evaluation Period MUST be greater than zero and MUST be greater than Minimum Evaluation Period (if present), otherwise the receiver MUST return a CoAP error code 4.00 "Bad Request" (or equivalent).

3.2.5. Confirmable Notification (c.con)

When present with a value of 1 (True), Confirmable Notification indicates a notification MUST be confirmable, i.e., the server MUST send the notification in a confirmable CoAP message, to request an acknowledgement from the client. When present with a value of 0 (False), Confirmable Notification indicates a notification can be confirmable or non-confirmable, i.e., it can be sent in a confirmable or a non-confirmable CoAP message.

3.3. Server processing of Conditional Attributes

Conditional Notification Attributes and Conditional Control Attributes 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. As a general example, the pseudocode illustrated in Appendix A shows one way to determine when a notification is to be sent.

4. Implementation Considerations

When "c.pmax" and "c.pmin" are equal, the expected behaviour is that notifications will be sent every (c.pmin == c.pmax) seconds. However, these notifications can only be fulfilled by the server on a best effort basis. Because "c.pmin" and "c.pmax" are designed as acceptable tolerance bounds for sending state updates, a query from an interested client containing equal "c.pmin" and "c.pmax" values must not be seen as a hard real-time scheduling contract between the client and the server.

The use of the notification band minimum and maximum allows for a synchronization whenever a change in the resource value occurs. Theoretically this could occur in-line with the server internal sample period or as defined by the "c.epmin" and "c.epmax" values for 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.

When a server has multiple observations with different measurement cadences as defined by the "c.epmin" and "c.epmax" values, the server MAY evaluate all observations when performing the measurement of any one observation.

This specification defines conditional attributes that can be used with CoAP Observe relationships between CoAP clients and CoAP servers. However, it is recognised that the presence of one or more proxies between a client and a server can interfere with clients receiving resource updates, if a proxy does not supply resource representations when the value remains unchanged (e.g. if "c.pmax" is set, and the server sends multiple updates when the resource state contains the same value). A server SHOULD use the Max-Age option to mitigate this by setting Max-Age to be less than or equal to "c.pmax".

5. Security Considerations

The security considerations in Section 11 of [RFC7252] apply. Additionally, the security considerations in Section 7 of [RFC7641] also apply.

6. IANA Considerations

This specification requests a new Conditional Attributes registry to ensure attributes map uniquely to parameter names.

Note to IANA: Please replace "RFC XXXX" with the assigned RFC number in the table below.

Attribute	Parameter	Value	Reference
Minimum Period (s)	c.pmin	xs:decimal (>0)	RFC XXXX
Maximum Period (s)	c.pmax	xs:decimal (>0)	RFC XXXX
Minimum Evaluation Period (s)	c.epmin	xs:decimal (>0)	RFC XXXX
Maximum Evaluation Period (s)	c.epmax	xs:decimal (>0)	RFC XXXX
Confirmable Notification	c.con	xs:boolean	RFC XXXX
Greater Than	c.gt	xs:decimal	RFC XXXX
Less Than	c.lt	xs:decimal	RFC XXXX
Change Step	c.st	xs:decimal (>0)	RFC XXXX
Notification Band	c.band	(none)	RFC XXXX
Edge	c.edge	xs:boolean	RFC XXXX

Table 3

7. Acknowledgements

Hannes Tschofenig and Mert Ocak highlighted syntactical corrections in the usage of pmax and pmin in a query. David Navarro proposed allowing for pmax to be equal to pmin. Marco Tiloca provided an extensive review.

8. Contributors

Christian Groves
Australia
email: cngroves.std@gmail.com

Zach Shelby
ARM
Vuokatti
FINLAND
phone: +358 40 7796297
email: zach.shelby@arm.com

Matthieu Vial
Schneider-Electric
Grenoble
France
phone: +33 (0)47657 6522
eMail: matthieu.vial@schneider-electric.com

Jintao Zhu
Huawei
Xian, Shaanxi Province
China
email: jintao.zhu@huawei.com

9. Changelog

draft-ietf-core-conditional-attributes-06

* Removed code block from Section 3.5

* Added an appendix containing pseudocode for server processing.

draft-ietf-core-conditional-attributes-05

* Multiple (mostly editorial) clarifications and updates based on review comments on mailing list from Marco Tiloca.

draft-ietf-core-conditional-attributes-04

* Reference code updated to include behaviour for edge attribute.

draft-ietf-core-conditional-attributes-03

* Attribute names updated to create uniqueness for use as conditional observe attributes.

draft-ietf-core-conditional-attributes-02

* Clarifications on usage and value of the band parameter

* Implementation considerations for proxies added

* Security considerations added

* IANA considerations added

draft-ietf-core-conditional-attributes-01

* Clarifications on True and False values for Edge and Con Attributes

* Alan Soloway added as author

draft-ietf-core-conditional-attributes-00

* Conditional Atttributes section from draft-ietf-core-dynlink-13 separated into own WG draft

10. 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>.

[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>.

Appendix A. Pseudocode: Processing Conditional Attributes

This appendix is informative. It describes the possible logic of how a server processes conditional attributes to determine when to send a notification to a client.

Note: The pseudocode is not exhaustive nor should it be treated as reference code. It depicts a subset of the conditional attributes described in this document.

```
// struct Resource {
//
//  bool band;
//  int pmin;
//  int pmax;
//  int epmin;
//  int epmax;
//  int st;
//  int gt;
//  int lt;
//
//  time_t last_sampled_time;
//  time_t last_rep_time;

//  int curr_state;
//  int prev_state;
//
//  ...
//
// };


boolean is_notifiable( Resource * r ) {

    time_t curr_time = get_current_time();

    #define BAND_EXISTS ( r->band )

    #define LT_EXISTS ( r->lt )
    #define GT_EXISTS ( r->gt )

    #define EPMIN_TRUE ( curr_time - r->last_sampled_time >= r->epmin )
    #define EPMAX_TRUE ( curr_time - r->last_sampled_time > r->epmax )
```

```
    #define PMIN_TRUE ( curr_time - r->last_reported_time >= r->pmin )
    #define PMAX_TRUE ( curr_time - r->last_reported_time > r->pmax )

    #define LT_TRUE ( r->curr_state < r->lt ^ r->prev_state < r->lt )
    #define GT_TRUE ( r->curr_state > r->gt ^ r->prev_state > r->gt )

    #define ST_TRUE ( abs( r->curr_state - r->prev_state ) >= r->st )

    #define INBAND_TRUE ( gt < lt && (gt <= curr_state && curr_state <= lt ))
    #define OUTOFBAND_TRUE ( lt < gt && (gt < curr_state || curr_state < lt ))

    #define BANDMIN_TRUE ( r->lt <= r->curr_state)
    #define BANDMAX_TRUE (r->curr_state <= r->gt)


    if PMAX_TRUE {
        return true;
    }

    if PMIN_TRUE {
        if !BAND_EXISTS {
            if LT_TRUE || GT_TRUE || ST_TRUE {
                return true;
            }
        }
        else {
         if ( ( BANDMIN_TRUE && !GT_EXISTS) || (BANDMAX_TRUE && !LT_EXISTS) || IN
BAND_TRUE || OUTOFBAND_TRUE ) {
             return true;
         }
        }
    }

    return false;

}
```

Figure 1: Pseudocode showing the logic for processing conditional attributes

Appendix B. Examples

This appendix is informative. It provides some examples of the use of Conditional Attributes.

Note: For brevity only the method or response code is shown in the header field.

B.1. Minimum Period (c.pmin) example

```
           Observed   CLIENT  SERVER     Actual
      t    State          |       |         State
           ____________   |       |   ____________
      1                   |       |
      2     unknown       |       |      18.5 Cel
      3                  +------>|                   Header: GET
      4                  | GET   |                    Token: 0x4a
      5                  |       |                Uri-Path: temperature
      6                  |       |               Uri-Query: c.pmin="10"
      7                  |       |                 Observe: 0 (register)
      8                  |       |
      9   ____________   |<------+                   Header: 2.05
     10                  | 2.05  |                    Token: 0x4a
     11    18.5 Cel      |       |                 Observe: 9
     12                  |       |                 Payload: "18.5 Cel"
     13                  |       |   ____________
     14                  |       |
     15                  |       |      23 Cel
     16                  |       |
     17                  |       |
     18                  |       |
     19                  |       |   ____________
     20   ____________   |<------+                   Header: 2.05
     21                  | 2.05  |      26 Cel        Token: 0x4a
     22    26 Cel        |       |                 Observe: 20
     23                  |       |                 Payload: "26 Cel"
     24                  |       |
     25                  |       |
```

Figure 2: Client registers and receives one notification of the current state and one of a new state state when c.pmin time expires.

B.2. Maximum Period (c.pmax) example

```
           Observed   CLIENT  SERVER     Actual
      t    State          |       |         State
           _____________  |       |  _____________
      1                   |       |
      2     unknown       |       |     18.5 Cel
      3                  +------>|                   Header: GET
      4                   | GET   |                    Token: 0x4a
      5                   |       |                 Uri-Path: temperature
      6                   |       |               Uri-Query: c.pmax="20"
      7                   |       |                 Observe: 0 (register)
      8                   |       |
      9   _____________   |<------+                  Header: 2.05
     10                   | 2.05  |                   Token: 0x4a
     11    18.5 Cel       |       |                 Observe: 9
     12                   |       |                 Payload: "18.5 Cel"
     13                   |       |
     14                   |       |
     15                   |       |  _____________
     16   _____________   |<------+                  Header: 2.05
     17                   | 2.05  |     23 Cel        Token: 0x4a
     18    23 Cel         |       |                 Observe: 16
     19                   |       |                 Payload: "23 Cel"
     20                   |       |
     21                   |       |
     22                   |       |
     23                   |       |
     24                   |       |
     25                   |       |
     26                   |       |
     27                   |       |
     28                   |       |
     29                   |       |
     30                   |       |
     31                   |       |
     32                   |       |
     33                   |       |
     34                   |       |
     35                   |       |
     36                   |       |  _____________
     37   _____________   |<------+                  Header: 2.05
     38                   | 2.05  |     23 Cel        Token: 0x4a
     39    23 Cel         |       |                 Observe: 37
     40                   |       |                 Payload: "23 Cel"
     41                   |       |
     42                   |       |
```

Figure 3: Client registers and receives one notification of the current state, one of a new state and one of an unchanged state when c.pmax time expires.

B.3. Greater Than (c.gt) example

```
        Observed   CLIENT  SERVER     Actual
    t   State          |       |         State
        _____________  |       |  _____________
    1                  |       |
    2    unknown       |       |      18.5 Cel
    3                 +----->|                   Header: GET
    4                 | GET   |                    Token: 0x4a
    5                 |       |                Uri-Path: temperature
    6                 |       |               Uri-Query: c.gt=25
    7                 |       |                 Observe: 0 (register)
    8                 |       |
    9   _____________ |<-----+                   Header: 2.05
   10                 | 2.05  |                   Token: 0x4a
   11    18.5 Cel     |       |                 Observe: 9
   12                 |       |                 Payload: "18.5 Cel"
   13                 |       |
   14                 |       |
   15                 |       |  _____________
   16   _____________ |<-----+                   Header: 2.05
   17                 | 2.05  |      26 Cel       Token: 0x4a
   18    26 Cel       |       |                 Observe: 16
   29                 |       |                 Payload: "26 Cel"
   20                 |       |
   21                 |       |
```

Figure 4: 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.

B.4. Greater Than (c.gt) and Period Max (c.pmax) example

```
        Observed   CLIENT  SERVER     Actual
   t    State          |       |         State
        _____________  |       |  _____________
   1                   |       |
   2     unknown       |       |     18.5 Cel
   3                  +----->|                  Header: GET
   4                  | GET  |                   Token: 0x4a
   5                  |       |                Uri-Path: temperature
   6                  |       |         Uri-Query: c.pmax=20;c.gt=25
   7                  |       |                 Observe: 0 (register)
   8                  |       |
   9   _____________  |<-----+                   Header: 2.05
  10                  | 2.05 |                    Token: 0x4a
  11    18.5 Cel      |       |                 Observe: 9
  12                  |       |                 Payload: "18.5 Cel"
  13                  |       |
  14                  |       |
  15                  |       |
  16                  |       |
  17                  |       |
  18                  |       |
  19                  |       |
  20                  |       |
  21                  |       |
  22                  |       |
  23                  |       |
  24                  |       |
  25                  |       |
  26                  |       |
  27                  |       |
  28                  |       |
  29                  |       |  _____________
  30   _____________  |<-----+                   Header: 2.05
  31                  | 2.05 |     23 Cel         Token: 0x4a
  32    23 Cel        |       |                 Observe: 30
  33                  |       |                 Payload: "23 Cel"
  34                  |       |
  35                  |       |
  36                  |       |  _____________
  37   _____________  |<-----+                   Header: 2.05
  38                  | 2.05 |     26 Cel         Token: 0x4a
  39    26 Cel        |       |                 Observe: 37
  40                  |       |                 Payload: "26 Cel"
  41                  |       |
  42                  |       |
```

Figure 5: Client registers and receives one notification of the current state, one when c.pmax time expires and one of a new state when it passes through the greater than threshold of 25.

Authors' Addresses

Michael Koster
Dogtiger Labs
524 H Street
Antioch, CA, 94509
United States of America
Email: michaeljohnkoster@gmail.com

Alan Soloway
Qualcomm Technologies, Inc.
5775 Morehouse Drive
San Diego, 92121
United States of America
Email: asoloway@qti.qualcomm.com

Bilhanan Silverajan (editor)
Tampere University
Kalevantie 4
FI-33100 Tampere
Finland
Email: bilhanan.silverajan@tuni.fi

CoRE Working Group
Internet-Draft
Intended status: Informational
Expires: January 13, 2022

M. Koster
SmartThings
B. Silverajan, Ed.
Tampere University

July 12, 2021

Dynamic Resource Linking for Constrained RESTful Environments
draft-ietf-core-dynlink-14

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).

Editor note

The git repository for the draft is found at https://github.com/core-wg/dynlink

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction 2
2. Terminology 3
3. Link Bindings 3
3.1. The "bind" attribute and Binding Methods 4
3.1.1. Polling 5
3.1.2. Observe 5
3.1.3. Push 5
3.1.4. Execute 6
3.2. Link Relation 6
4. Binding Table 6
5. Implementation Considerations 8
6. Security Considerations 8
7. IANA Considerations 8
7.1. Resource Type value ’core.bnd’ 8
7.2. Link Relation Type 8
8. Acknowledgements 9
9. Contributors 9
10. Changelog 10
11. References 12
11.1. Normative References 12
11.2. Informative References 13
Authors’ Addresses 13

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.

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], [RFC6690] and [RFC7641]. 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.

3. 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 [I-D.ietf-core-conditional-attributes] can be used with Link Bindings in order to customize the notification behavior and timing.

3.1. 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 1: 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
Execute	exec	Source	POST

Table 2: 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 defined in [I-D.ietf-core-conditional-attributes] are used in the binding.

The binding methods are described in more detail below.

3.1.1. 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 defined in [I-D.ietf-core-conditional-attributes]).

3.1.2. 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 [I-D.ietf-core-conditional-attributes]).

3.1.3. Push

The Push method can be used to allow a source endpoint to replace an outdated resource state at the destination with a newer representation. 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.

3.1.4. Execute

An alternative means for a source endpoint to deliver change-of-state notifications to a destination resource is to use the Execute Method. While the Push method simply updates the state of the destination resource with the representation of the source resource, Execute can be used when the destination endpoint wishes to receive all state changes from a source. This allows, for example, the existence of a resource collection consisting of all the state changes at the destination endpoint. When the Execute method is assigned to a binding, the source endpoint sends POST 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.

Note: Both the Push and the Execute methods are examples of Server Push mechanisms that are being researched in the Thing-to-Thing Research Group (T2TRG) [I-D.irtf-t2trg-rest-iot].

3.2. 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.

4. 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 3: 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 rquest 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 1: 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.

5. Implementation Considerations

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.

6. 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.

7. IANA Considerations

7.1. 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 4. 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

7.2. 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

8. 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. Discussions with Ari Keraenen led to the addition of an extra binding method supporting POST operations.

9. Contributors

Christian Groves
Australia
email: cngroves.std@gmail.com

Zach Shelby
ARM
Vuokatti
FINLAND
phone: +358 40 7796297
email: zach.shelby@arm.com

Matthieu Vial
Schneider-Electric
Grenoble
France
phone: +33 (0)47657 6522
eMail: matthieu.vial@schneider-electric.com

Jintao Zhu
Huawei
Xi'an, Shaanxi Province
China
email: jintao.zhu@huawei.com

10. Changelog

draft-ietf-core-dynlink-14

- o Conditional Atttributes section removed and submitted as draft-ietf-core-conditional-attributes-00

draft-ietf-core-dynlink-13

- o Conditional Atttributes section restructured
- o "edge" and "con" attributes added
- o Implementation considerations, clarifications added when pmax == pmin
- o rewritten to remove talk of server reporting values to clients

draft-ietf-core-dynlink-12

- o Attributes epmin and epmax included
- o pmax now can be equal to pmin

draft-ietf-core-dynlink-11

- o Updates to author list

draft-ietf-core-dynlink-10

- o Binding methods now support both POST and PUT operations for server push.

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 conlict 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".

- Added Appendix A for examples.

draft-ietf-core-dynlink-01

- General: The term state synchronization has been introduced to describe the process of synchronization between destination and source resources.

- General: The document has been restructured the make the information flow better.

- Clause 3.1: The descriptions of the binding attributes have been updated to clarify their usage.

- Clause 3.1: A new clause has been added to discuss the interactions between the resources.

- Clause 3.4: Has been simplified to refer to the descriptions in 3.1. As the text was largely duplicated.

- Clause 4.1: Added a clarification that individual resources may be removed from the binding table.

- Clause 6: Formailised the IANA considerations.

draft-ietf-core-dynlink Initial Version 00:

- This is a copy of draft-groves-core-dynlink-00

draft-groves-core-dynlink Draft Initial Version 00:

- This initial version is based on the text regarding the dynamic linking functionality in I.D.ietf-core-interfaces-05.

- The WADL description has been dropped in favour of a thorough textual description of the REST API.

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>.

[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>.

11.2. Informative References

[I-D.ietf-core-conditional-attributes] Koster, M. and B. Silverajan, "Conditional Attributes for Constrained RESTful Environments", draft-ietf-core-conditional-attributes-00 (work in progress), July 2021.

[I-D.irtf-t2trg-rest-iot] Keranen, A., Kovatsch, M., and K. Hartke, "RESTful Design for Internet of Things Systems", draft-irtf-t2trg-rest-iot-07 (work in progress), February 2021.

[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>.

Authors' Addresses

Michael Koster
SmartThings
665 Clyde Avenue
Mountain View 94043
USA

Email: michael.koster@smartthings.com

Bilhanan Silverajan (editor)
Tampere University
Kalevantie 4
Tampere FI-33100
Finland

Email: bilhanan.silverajan@tuni.fi

CoRE Working Group
Internet-Draft
Obsoletes: 7390 (if approved)
Updates: 7252, 7641 (if approved)
Intended status: Standards Track
Expires: 26 October 2024

E. Dijk
IoTconsultancy.nl
C. Wang
InterDigital
M. Tiloca
RISE AB

24 April 2024

Group Communication for the Constrained Application Protocol (CoAP)
draft-ietf-core-groupcomm-bis-11

Abstract

This document specifies the use of the Constrained Application Protocol (CoAP) for group communication, including the use of UDP/IP multicast as the default underlying data transport. 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. The target application area of this specification is any group communication use cases that involve resource-constrained devices or networks that support CoAP. This document replaces and obsoletes RFC 7390, while it updates RFC 7252 and RFC 7641.

Discussion Venues

This note is to be removed before publishing as an RFC.

Discussion of this document takes place on the CORE Working Group mailing list (core@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/core/ (https://mailarchive.ietf.org/arch/browse/core/).

Source for this draft and an issue tracker can be found at https://github.com/core-wg/groupcomm-bis (https://github.com/core-wg/groupcomm-bis).

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction 4
  1.1. Scope 5
  1.2. Terminology 6
  1.3. Changes to Other Documents 6
2. Group Definition and Group Configuration 7
  2.1. Group Definition 8
    2.1.1. CoAP Group 8
    2.1.2. Application Group 8
    2.1.3. Security Group 9
    2.1.4. Relations Between Group Types 9
  2.2. Group Configuration 12
    2.2.1. Group Naming 12
    2.2.2. Group Creation and Membership 17
    2.2.3. Group Discovery 18
    2.2.4. Group Maintenance 22
3. CoAP Usage in Group Communication 23
  3.1. Request/Response Model 23
    3.1.1. General 23
    3.1.2. Response Suppression 23
    3.1.3. Repeating a Request 24
    3.1.4. Request/Response Matching and Distinguishing Responses 24
    3.1.5. Token Reuse 25
    3.1.6. Client Handling of Multiple Responses With Same Token 26

3.2. Caching 27
3.2.1. Freshness Model 28
3.2.2. Validation Model 28
3.3. URI Path Selection 29
3.4. Port Selection for UDP Transport 30
3.5. Proxy Operation 30
3.5.1. Forward-Proxies 30
3.5.2. Reverse-Proxies 32
3.5.3. Single Group Request to Multiple Proxies 34
3.6. Congestion Control 35
3.7. Observing Resources 37
3.8. Block-Wise Transfer 39
3.9. Transport Protocols 39
3.9.1. UDP/IPv6 Multicast Transport 40
3.9.2. UDP/IPv6 Multicast Transport over 6LoWPAN 40
3.9.3. UDP/IPv4 Multicast Transport 41
3.9.4. TCP, TLS, and WebSockets 41
3.9.5. Other Transports 42
3.10. Interworking with Other Protocols 42
3.10.1. MLD/MLDv2/IGMP/IGMPv3 42
3.10.2. RPL 42
3.10.3. MPL 43
4. Unsecured Group Communication 44
5. Secured Group Communication using Group OSCORE 45
5.1. Group OSCORE 45
5.2. Secure Group Maintenance 47
5.3. Proxy Security 48
6. Security Considerations 49
6.1. CoAP NoSec Mode 49
6.2. Group OSCORE 50
6.2.1. Group Key Management 51
6.2.2. Source Authentication 52
6.2.3. Countering Attacks 52
6.3. Risk of Amplification 55
6.4. Replay of Non-Confirmable Messages 57
6.5. Use of CoAP No-Response Option 57
6.6. 6LoWPAN and MPL 58
6.7. Wi-Fi 59
6.8. Monitoring 59
6.8.1. General Monitoring 59
6.8.2. Pervasive Monitoring 59
7. IANA Considerations 60
8. References 60
8.1. Normative References 60
8.2. Informative References 62
Appendix A. Use Cases 66
A.1. Discovery 66
A.1.1. Distributed Device Discovery 66

A.1.2. Distributed Service Discovery 67
A.1.3. Directory Discovery 67
A.2. Operational Phase 68
A.2.1. Actuator Group Control 68
A.2.2. Device Group Status Request 68
A.2.3. Network-wide Query 68
A.2.4. Network-wide / Group Notification 69
A.3. Software Update 69
Appendix B. Examples of Group Naming for Application Groups . . 69
B.1. Group Naming using the URI Path Component 70
B.2. Group Naming using the URI Query Component 71
B.3. Group Naming using the URI Authority Component 71
B.4. Group Naming using the URI Host Subcomponent 72
B.5. Group Naming using the URI Port Subcomponent 72
B.6. Group Naming using a Custom CoAP Option 73
Appendix C. Examples of Group Discovery from CoAP Servers . . . 73
C.1. Application Groups Associated with a CoAP Group 73
C.2. Members of a Given Application Group 74
C.3. Members of any Application Group of a Given Type 75
C.4. Members of any Application Group in the Network 76
Appendix D. Examples of Message Exchanges 77
Appendix E. Document Updates 84
E.1. Version -10 to -11 84
E.2. Version -09 to -10 84
E.3. Version -08 to -09 85
E.4. Version -07 to -08 85
E.5. Version -06 to -07 85
E.6. Version -05 to -06 86
E.7. Version -04 to -05 86
E.8. Version -03 to -04 87
E.9. Version -02 to -03 87
E.10. Version -01 to -02 88
E.11. Version -00 to -01 88
Acknowledgments 88
Authors' Addresses 89

1. Introduction

This document specifies group communication using the Constrained Application Protocol (CoAP) [RFC7252], together with UDP/IP multicast as the default transport for CoAP group communication messages. 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 that support group communication include "Eclipse Californium" [Californium], "Go-CoAP" [Go-CoAP] as well as "libcoap" [libcoap].

Both unsecured and secured CoAP group communication 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) [RFC8613]. This method provides end-to-end application-layer security protection of CoAP messages, by using CBOR Object Signing and Encryption (COSE) [RFC9052][RFC9053].

This document replaces and obsoletes [RFC7390], while it updates both [RFC7252] and [RFC7641]. A summary of the changes and additions to these documents is provided in Section 1.3.

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 those solutions that use CoAP messages over a "one-to-many" (i.e., non-unicast) transport protocol are in the scope of this document. There are alternative methods to achieve group communication using CoAP, using unicast only. One example is Publish-Subscribe [I-D.ietf-core-coap-pubsub] which uses a central broker server that CoAP clients access via unicast communication. These alternative methods may be usable for the same or similar use cases as the ones targeted in this document.

This document defines UDP/IP multicast as the default transport protocol for CoAP group requests, as in [RFC7252]. Other transport protocols (which may include broadcast, non-IP multicast, geocast, etc.) are not described in detail and are not considered. Although UDP/IP multicast transport is assumed in most of the text in this document, we expect many of the considerations for UDP/IP multicast can be re-used for alternative transport protocols.

Furthermore, this document defines Group OSCORE [I-D.ietf-core-oscore-groupcomm] as the default group communication security solution for CoAP. Security solutions for group

communication and configuration other than Group OSCORE are not considered. General principles for secure group configuration are in scope.

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 terminology [RFC7252]. Terminology related to group communication is defined in Section 2.1.

In addition, the following terms are extensively used.

* Group URI -- This is defined as a CoAP URI that has the "coap" scheme and includes in the authority component either an IP multicast address or a group hostname (e.g., a Group Fully Qualified Domain Name (FQDN)) that can be resolved to an IP multicast address. A group URI also can contain a UDP port number in the authority component. Group URIs follow the regular CoAP URI syntax (see Section 6 of [RFC7252]).

* Security material -- This refers to any security keys, counters, or parameters stored in a device that are required to participate in secure group communication with other devices.

1.3. Changes to Other Documents

This document obsoletes and replaces [RFC7390] as follows.

* It provides separate definitions for CoAP groups, application groups, and security groups, together with high-level guidelines on their configuration (see Section 2).

* It defines the use of Group OSCORE [I-D.ietf-core-oscore-groupcomm] as the security protocol to protect group communication for CoAP, together with high-level guidelines on secure group maintenance (see Section 5).

* It updates all the guidelines about using group communication for CoAP (see Section 3).

* It strongly discourages unsecured group communication for CoAP based on the CoAP NoSec (No Security) mode (see Section 4 and Section 6.1), and highlights the risk of amplification attacks (see Section 6.3).

* It updates all sections on transport protocols and interworking with other protocols based on new IETF work done for these protocols.

This document updates [RFC7252] as follows.

* It updates the request/response model for group communication, as to response suppression (see Section 3.1.2) and token reuse time (see Section 3.1.5).

* It updates the freshness model and validation model to use for cached responses (see Section 3.2.1 and Section 3.2.2).

* It defines the measures against congestion risk specified in [RFC7252] to be applicable also to alternative transports other than IP multicast, and defines additional guidelines to reduce congestion risks (see Section 3.6).

* It explicitly admits the use of the IPv6 multicast address scopes realm-local (3), admin-local (4), and global (E). In particular, it recommends that an IPv6 CoAP server supports at least link-local (2), admin-local (4), and site-local (5) scopes with the "All CoAP Nodes" multicast CoAP group (see Section 3.9.1). Also, it recommends that the realm-local (3) scope is supported by an IPv6 CoAP server on a 6LoWPAN node (see Section 3.9.1).

This document updates [RFC7641] as follows.

* It defines the use of the CoAP Observe Option in group requests, for both the GET method and the FETCH method [RFC8132], together with normative behavior for both CoAP clients and CoAP servers (see Section 3.7).

2. Group Definition and Group Configuration

In the following, different group types are first defined in Section 2.1. Then, Group configuration, including group creation and maintenance by an application, user, or commissioning entity is considered in Section 2.2.

2.1. Group Definition

Three types of groups and their mutual relations are defined in this section: CoAP group, application group, and security group.

2.1.1. CoAP Group

A CoAP group is defined as a set of CoAP endpoints, where each endpoint is configured to receive CoAP group messages that are sent to the group's associated IP multicast address and UDP port. That is, CoAP groups have relevance at the level of IP networks and CoAP endpoints.

An endpoint may be a member of multiple CoAP groups, by subscribing to multiple IP multicast addresses. A node may be a member of multiple CoAP groups, by hosting multiple CoAP server endpoints on different UDP ports. Membership(s) of an endpoint or node to a CoAP group may dynamically change over time. A node or endpoint sending a CoAP group message to a CoAP group is not necessarily itself a member of that CoAP group: it is a member only if it also has a CoAP endpoint listening on the associated IP multicast address and UDP port associated with the CoAP group.

A CoAP group is identified by information encoded within a group URI. Further details on identifying a CoAP group are provided in Section 2.2.1.1.

2.1.2. Application Group

An application group is a set of CoAP server endpoints (hosted on different nodes) that share a common set of CoAP resources. That is, 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.

An endpoint may be a member of multiple application groups. A client endpoint that sends a group communication message to an application group is not necessarily itself a member of this application group.

There can be a one-to-one or a one-to-many relation between a CoAP group and application group(s). Such relations are discussed in more detail in Section 2.1.4.

An application group name may be explicitly encoded in the group URI of a CoAP request, for example in the URI path component. If this is not the case, the application group is implicitly derived by the receiver, e.g., based on information in the CoAP request or other contextual information. Further details on identifying an application group are provided in Section 2.2.1.2.

2.1.3. Security Group

For secure group communication, a security group is required. A security group comprises endpoints storing shared group security material, such that they can use it to protect and verify mutually exchanged messages.

That is, a client endpoint needs to be a member of a security group in order to send a valid secured group communication message to that group. A server endpoint needs to be a member of a security group in order to receive and correctly verify a secured group communication message sent to that group. An endpoint may be a member of multiple security groups.

There can be a many-to-many relation between security groups and CoAP groups, but often it is one-to-one. Also, there can be a many-to-many relation between security groups and application groups, but often it is one-to-one. Such relations are discussed in more detail in Section 2.1.4.

Further details on identifying a security group are provided in Section 2.2.1.3.

If the NoSec mode is used (see Section 4), group communication does not rely on security at the transport layer nor at the CoAP layer, hence the communicating endpoints do not refer to a security group.

2.1.4. Relations Between Group Types

Using the above group type definitions, a CoAP group communication message sent by an endpoint can be associated with a tuple that contains one instance of each group type:

(application group, CoAP group, security group)

A special note is appropriate about the possible relation between security groups and application groups.

On one hand, multiple application groups may use the same security group. Thus, the same group security material is used to protect the messages targeting any of those application groups. This has the

benefit that typically less storage, configuration, and updating are required for security material. In this case, a CoAP endpoint is supposed to know the exact application group to refer to for each message that is sent or received, based on, e.g., the server port number used, the targeted resource, or the content and structure of the message payload.

On the other hand, a single application group may use multiple security groups. Thus, different messages targeting the resources of the application group can be protected with different security material. This can be convenient, for example, if the security groups differ with respect to the cryptographic algorithms and related parameters they use. In this case, a CoAP client can join just one of the security groups, based on what it supports and prefers, while a CoAP server in the application group would rather have to join all of them.

Beyond this particular case, applications should be careful in associating a single application group to multiple security groups. In particular, it is NOT RECOMMENDED to use different security groups to reflect different access policies for resources in the same application group.

In fact, being a member of a security group actually grants access only to exchange secured messages and enables authentication of group members, while access control (authorization) to use resources in the application group belongs to a separate security domain. Therefore, access control to use resources in the application group should be separately enforced by leveraging the resource properties or through dedicated access control credentials assessed by separate means.

Figure 1 summarizes the relations between the different types of groups described above in Unified Modeling Language (UML) class diagram notation. The class attributes in square brackets are optionally defined.

```
   +-----------------------------+              +--------------------+
   |   Application group         |              |    CoAP group      |
   |.............................|              |....................|
   |                             |              |                    |
   |  [ - Application group name ] +--------------+ - IP mcast address |
   |                             |  1...N    1  | - UDP port number  |
   |                             |              |                    |
   |                             |              |                    |
   +-------------+----------------+              +---------+----------+
                 |  1...N                                  |  1...N
                 |                                         |
                 |                                         |
                 |                                         |  1...N
                 |                              +------------+----------+
                 |                              |   Security group      |
                 |                              |.......................|
                 |                              |                       |
                 '-----------------------------+ - Security group name |
                                        1...N  | - Security material   |
                                               |                       |
                                               +-----------------------+
```

Figure 1: Relations Among Different Group Types

Figure 2 provides a deployment example of the relations between the different types of groups. It shows six CoAP servers (Srv1-Srv6) and their respective resources hosted (/resX). Although in real-life deployments using group communication the number of servers and resources would usually be higher, only limited numbers are shown here for ease of representation.

There are three application groups (1, 2, 3) and two security groups (1, 2). The Security Group 1 may, for example, include all lighting devices on a floor of an office building, while Security Group 2 includes all Heating, Ventilation, and Air Conditioning (HVAC) devices of that floor. Security Group 1 is used by both Application Group 1 and 2. The Application Group 1 for example may consist of all lights in a hallway, while Application Group 2 includes all lights in a storage room. Three clients (Cli1, Cli2, Cli3) are configured with security material for Security Group 1. These clients may be motion sensors and a control panel (Cli3), that send multicast messages to /resA to inform the lights of any motion or user activity detected. The control panel Cli3 additionally sends a multicast message to /resB to communicate the latest light preset selected by a user. The latter action only influences the lighting in the storage room (Application Group 2). Two clients (Cli2, Cli4) are configured with security material for Security Group 2. These clients may be temperature/humidity sensors that report measurements

periodically to all HVAC devices (Srv5, Srv6) in the Application Group 3, using for example /resC to report temperature and /resD to report humidity.

All the shown application groups may use the same CoAP group (not shown in the figure), for example the CoAP group with site-local, site-specific multicast IP address ff15::3456 and default UDP port number 5683 on which all the shown resources are hosted for each server. Other floors of the same building may replicate the shown structure, but using different security groups and different CoAP groups.

```
 .-----------------------------.   .----------------------------.
|                               | |                              |
|       +---------------------+ | |  +---------------------+     |
|       | Application Group 1 | | |  | Application Group 3 | Cli2 |
|       |                     | | |  |                     |     |
| Cli1  | Srv1   Srv2   Srv3  | | |  | Srv5   Srv6         | Cli4 |
|       | /resA  /resA  /resA | | |  | /resC  /resC        |     |
| Cli2  +---------------------+ | |  | /resD  /resD        |     |
|                               | |  +---------------------+     |
| Cli3     Security Group 1     | |                              |
|                               | |        Security Group 2      |
|       +---------------------+ |  '----------------------------'
|       | Application Group 2 | |
|       |                     | |
|       | Srv1   Srv4         | |
|       | /resB  /resB        | |
|       +---------------------+ |
 '-----------------------------'
```

Figure 2: Deployment Example of Different Group Types

2.2. Group Configuration

The following defines how groups of different types are named, created, discovered, and maintained.

2.2.1. Group Naming

Different types of groups are named as specified below, separately for CoAP groups, application groups, and security groups.

2.2.1.1. CoAP Groups

A CoAP group is always defined by the two properties of IP multicast address and UDP port number (see Section 2.1.1).

However, a CoAP group is for practical purposes identified and named by the authority component in the group URI. This component includes the host subcomponent and an optional UDP port number. The host subcomponent directly defines the IP multicast address of the CoAP group, in case the host consists of an IP literal. The host subcomponent indirectly defines the IP multicast address of the CoAP group, in case the host consists of a hostname: resolving the hostname to an IP address in this case produces the IP multicast address.

It follows that the same CoAP group might have multiple names, which can be simultaneously and interchangeably used. For example, if the two hostnames group1.com and group1.alias.com both resolve to the IP multicast address [ff15::1234], then the following authority components are all names for the same CoAP group.

* group1.example:7700

* group1.alias.example:7700

* [ff15::1234]:7700

Also note that, when using the "coap" scheme, the two authority components <HOST> and <HOST>:5683 both identify the same CoAP group, whose members listen to the CoAP default port number 5683. Therefore, building on the above, the following authority components are all names for the same CoAP group.

* group1.example

* group1.alias.example

* [ff15::1234]

* group1.example:5683

* group1.alias.example:5683

* [ff15::1234]:5683

When configuring a CoAP group membership, it is recommended to configure an endpoint with an IP multicast address literal, instead of a group hostname. This is because DNS infrastructure may not be deployed in many constrained networks. In case a group hostname is configured, it can be uniquely mapped to an IP multicast address via DNS resolution, if DNS client functionality is available in the endpoint being configured and the DNS service is supported in the network.

Examples of hierarchical CoAP group FQDN naming (and scoping) for a building control application were shown in Section 2.2 of [RFC7390].

2.2.1.2. Application Groups

An application group can be named in many ways through different types of identifiers, such as name string, (integer) number, URI, or other types of strings. The decision of whether and how exactly an application group name is encoded and transported is application specific.

The following discusses a number of possible methods to use, while full examples for the different methods are provided in Appendix B.

An application group name can be explicitly encoded in a group URI. In such a case, it can be encoded within one of the following URI components.

* URI path component -- This is the most common and RECOMMENDED method to encode the application group name. When using this method in constrained networks, an application group name APPNAME should be kept short.

  A best practice for doing so is to use a URI path component such that: i) it includes a path segment as delimiter with a designated value, e.g., "gp", followed by ii) a path segment with value the name of the application group, followed by iii) the path segment(s) that identify the targeted resource within the application group. For example, both /gp/APPNAME/res1 and /base/gp/APPNAME/res1/res2 conform to this practice. Just like application group names, the path segment used as delimiter should be kept short in constrained networks.

  Full examples are provided in Appendix B.1.

* URI query component -- This method can use the following formats. In either case, when using this method in constrained networks, an application group name APPNAME should be as short as possible.

  - As a first alternative, the URI query component consists of only one parameter, which has no value and has the name of the application group as its own identifier. That is, the query component ?APPNAME conforms to this format.

- As a second alternative, the URI query component includes a query parameter as designated indicator, e.g., "gp", with value the name of the application group. That is, assuming "gp" to be used as designated indicator, both the query components ?gp=APPNAME and ?par1=v1&gp=APPNAME conform to this format.

Full examples are provided in Appendix B.2.

* URI authority component -- If this method is used, the application group is identified by the authority component as a whole.

  In particular, the application group has the same name of the CoAP group expressed by the group URI (see Section 2.2.1.1). Thus, this method can only be used if there is a one-to-one mapping between CoAP groups and application groups (see Section 2.1.4).

  While the host component of the Group URI can be a group hostname, an implementation would likely rather use an IP address literal, in order to reduce the size of the CoAP request. In particular, the Uri-Host Option can be fully elided in this case.

  A full example is provided in Appendix B.3.

* URI host subcomponent -- If this method is used, the application group is identified solely by the host subcomponent of the authority component.

  Since an application group can be associated with only one CoAP group (see Section 2.1.4), using this method implies that, given any two CoAP groups, the port subcomponent of the URI authority component MUST NOT be the only information distinguishing them.

  Like for the previous case relying on the whole URI authority component, an implementation would likely use an IP address literal rather than the group hostname as host component of the Group URI, in order to reduce the size of the CoAP request. In particular, the Uri-Host Option can be fully elided in this case.

  A full example is provided in Appendix B.4.

* URI port subcomponent -- By using this method, the application group is uniquely identified by the destination port number encoded in the port subcomponent of the authority component.

  Since an application group can be associated with only one CoAP group (see Section 2.1.4), using this method implies that any two CoAP groups cannot differ only by their host subcomponent of the URI authority component.

A full example is provided in Appendix B.5.

Alternatively, there are also methods to encode the application group name within the CoAP request, even though it is not encoded within the group URI. An example of such a method is summarized below.

* The application group name can be encoded in a new (e.g., custom, application-specific) CoAP Option, which the client adds to the CoAP request before sending it out.

  Upon receiving the request as a member of the targeted CoAP group, each CoAP server would, by design, understand this Option, decode it, and treat the result as an application group name.

  A full example is provided in Appendix B.6.

Furthermore, it is possible to encode the application group name neither in the group URI nor within a CoAP request, thus yielding the most compact representation on the wire. In this case, each CoAP server needs to determine the right application group based on contextual information, such as the client identity and/or the target resource. For example, each application group on a server could support a unique set of resources, such that it does not overlap with the set of resources of any other application group.

Finally, Appendix A of [RFC9176] provides an example of an application group registered to a Resource Directory (RD), along with the CoAP group it uses and the resources it supports. In that example, an application group name "lights" is encoded in the "ep" (endpoint) attribute of the RD registration entry, while the CoAP group ff35:30:2001:db8:f1::8000:1 is specified in the authority component of the URI encoded in the "base" attribute.

2.2.1.3. Security Groups

A security group can be named in many ways through different types of identifiers, such as name string, (integer) number, URI, or other types of strings. Such a group name is generally not related to other kinds of group identifiers that may be specific of the used security solution.

The name of a security group is not expected to be used in messages exchanged among its members, unless the application requires otherwise. At the same time, it is useful to identify the security group when performing a number of side tasks related to secure group communication, such as the following ones.

* An administrator may have to request for an authorization to configure security groups at an available Group Manager (see Section 5). During the authorization process, as well as during the interaction between the administrator and the Group Manager, the group name identifies the specific security group that the administrator wishes to configure and is authorized to.

* A CoAP endpoint may have to request for an authorization to join a specific security group through the respective Group Manager, and thus obtain the required group security material (see Section 5). During the authorization process, as well as during the interaction between the CoAP endpoint and the Group Manager, the group name identifies the specific security group that the CoAP endpoint wishes to join and is authorized to.

* A CoAP endpoint may first need to discover the specific security groups to join through the respective Group Manager (see Section 2.2.3.1). Results from the discovery process include the name of the security groups to join, together with additional information such as a pointer to the respective Group Manager.

It is discouraged to use "NoSec" and any of its lowercase/uppercase combinations as name of a security group. Indications that endpoints can use the NoSec mode MUST NOT rely on setting up and advertising a pseudo security group with name "NoSec" or any of its lowercase/uppercase combinations.

2.2.2. Group Creation and Membership

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 number 5683. Then, it configures one or more devices as listeners to that IP multicast address, with a CoAP endpoint listening on the CoAP group's associated UDP port. These endpoints/devices are the group members.

The configuring entity can be, for example, a local application with pre-configuration, a user, a software developer, a cloud service, or a local commissioning tool. Also, the devices sending CoAP requests to the CoAP group in the role of CoAP client 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 group URI.

The IETF does not define a mandatory protocol to accomplish CoAP group creation. [RFC7390] defined an experimental protocol for configuring memberships of CoAP groups for unsecured group communication, based on JSON-formatted configuration resources, and

the experiment is concluded. For IPv6 CoAP groups, common multicast address ranges that are used to configure group addresses from are ff1x::/16 and ff3x::/16.

To create an application group, a configuring entity may configure a resource (name) or a set of resources on CoAP endpoints, such that a CoAP request sent to a group URI by a configured CoAP client will be processed by one or more CoAP servers that have the matching URI path configured. These servers are the members of the application group.

To create a security group, a configuring entity defines an initial subset of the related security material. This comprises a set of group properties including the cryptographic algorithms and parameters used in the security group, as well as additional information relevant throughout the group life-cycle, such as the security group name and description. This task MAY be entrusted to a dedicated administrator, that interacts with a Group Manager as defined in Section 5. After that, further security materials to protect group communications have to be generated, compatible with the configuration specified for the security group.

To participate in a security group, CoAP endpoints have to be configured with the group security material used to protect communications in the associated application/CoAP groups. The part of the process that involves secure distribution of group security material MAY use standardized communication with a Group Manager as defined in Section 5.

For unsecure group communication using the NoSec mode (see Section 4), there is no security material to be provided, hence there is no security group for CoAP endpoints to participate in.

The configuration of groups and membership may be performed at different moments in the life-cycle of a device. For example, it can occur during product (software) creation, in the factory, at a reseller, on-site during first deployment, or on-site during a system reconfiguration operation.

2.2.3. Group Discovery

The following describes how a CoAP endpoint can discover groups by different means, i.e., by using a Resource Directory or directly from the CoAP servers that are members of such groups.

2.2.3.1. Discovery through a Resource Directory

It is possible for CoAP endpoints to discover application groups as well as CoAP groups, by using the RD-Groups usage pattern of the CoRE Resource Directory (RD), as defined in Appendix A of [RFC9176].

In particular, an application group can be registered to the RD, specifying the reference IP multicast address of its associated CoAP group. The registration of groups to the RD is typically performed by a Commissioning Tool. Later on, CoAP endpoints can discover the registered application groups and related CoAP group(s), by using the lookup interface of the RD.

When secure communication is provided with Group OSCORE (see Section 5), the approach described in [I-D.tiloca-core-oscore-discovery] also based on the RD can be used, in order to discover the security group to join.

In particular, the responsible OSCORE Group Manager registers its security groups to the RD, as links to its own corresponding resources for joining the security groups [I-D.ietf-ace-key-groupcomm-oscore]. Later on, CoAP endpoints can discover the names of the registered security groups and related application groups, by using the lookup interface of the RD, and then join the security group through the respective Group Manager.

2.2.3.2. Discovery from the CoAP Servers

It is possible for CoAP endpoints to discover application groups and CoAP groups from the CoAP servers that are members of such groups, by using a GET request targeting the /.well-known/core resource.

As discussed below, such a GET request may be sent to the IP multicast address of an already known CoAP group associated with one or more application groups; or to the "All CoAP Nodes" multicast address, thus targeting all reachable CoAP servers in any CoAP group. Also, the GET request may specify a query component, in order to filter the application groups of interest.

These particular details concerning the GET request depend on the specific discovery action intended by the client and on application-specific means used to encode names of application groups and CoAP groups, e.g., in group URIs and/or CoRE target attributes used with resource links.

The following discusses a number of methods to discover application groups and CoAP groups, building on the following assumptions. First, application group names are encoded in the path component of

Group URIs (see Section 2.2.1.2), using the path segment "gp" as designated delimiter. Second, the type of an application group is encoded in the CoRE Link Format attribute "rt" of a group resource with a value "g.<GROUPTYPE>".

Full examples for the different methods are provided in Appendix C.

* A CoAP client can discover all the application groups associated with a specific CoAP group.

  This is achieved by sending the GET request above to the IP multicast address of the CoAP group, and specifying a wildcarded group type "g.*" as resource type in the URI query parameter "rt". For example, the request can use a Group URI with path and query components "/.well-known/core?rt=g.*", so that the query matches any application group resource type. Alternatively, the request can use a Group URI with path and query components "/.well-known/core?href=/gp/*", so that the query matches any application group resources and also matches any sub-resources of those.

  Through the corresponding responses, the query result is a list of resources at CoAP servers that are members of the specified CoAP group and have at least one application group associated with the CoAP group. That is, the client gains knowledge of: i) the set of servers that are members of the specified CoAP group and member of any of the associated application groups; ii) for each of those servers, the name of the application groups where the server is a member and that are associated with the CoAP group.

  A full example is provided in Appendix C.1.

* A CoAP client can discover the CoAP servers that are members of a specific application group, the CoAP group associated with the application group, and optionally the resources that those servers host for each application group.

  This is achieved by sending the GET request above to the "All CoAP Nodes" IP multicast address (see Section 12.8 of [RFC7252]), with a particular chosen scope (e.g., site-local or realm-local) if IPv6 is used. Also, the request specifies the application group name of interest in the URI query component, as defined in Section 2.2.1.2. For example, the request can use a Group URI with path and query components "/.well-known/core?href=/gp/gp1" to specify the application group with name "gp1".

  Through the corresponding responses, the query result is a list of resources at CoAP servers that are members of the specified application group and for each application group the associated

CoAP group. That is, the client gains knowledge of: i) the set of servers that are members of the specified application group and of the associated CoAP group; ii) for each of those servers, optionally the resources it hosts within the application group.

If the client wishes to discover resources that a particular server hosts within a particular application group, it may use unicast discovery request(s) to this server.

A full example is provided in Appendix C.2.

* A CoAP client can discover the CoAP servers that are members of any application group of a specific type, the CoAP group associated with those application groups, and optionally the resources that those servers host as members of those application groups.

  This is achieved by sending the GET request above to the "All CoAP Nodes" IP multicast address (see Section 12.8 of [RFC7252]), with a particular chosen scope (e.g., site-local or realm-local) if IPv6 is used. Also, the request can specify the application group type of interest in the URI query component as value of a query parameter "rt". For example, the request can use a Group URI with path and query components "/.well-known/core?rt=TypeA" to specify the application group type "TypeA".

  Through the corresponding responses, the query result is a list of resources at CoAP servers that are members of any application group of the specified type and of the CoAP group associated with each of those application groups. That is, the client gains knowledge of: i) the set of servers that are members of the application groups of the specified type and of the associated CoAP group; ii) optionally for each of those servers, the resources it hosts within each of those application groups.

  If the client wishes to discover resources that a particular server hosts within a particular application group, it may use unicast discovery request(s) to this server.

  A full example is provided in Appendix C.3.

* A CoAP client can discover the CoAP servers that are members of any application group configured in the 6LoWPAN wireless mesh network of the client, the CoAP group associated with each application group, and optionally the resources that those servers host as members of the application group.

This is achieved by sending the GET request above with a query specifying a wildcarded group type in the URI query parameter for "rt". For example, the request can use a Group URI with path and query components "/.well-known/core?rt=g.*", so that the query matches any application group type. The request is sent to the "All CoAP Nodes" IP multicast address (see Section 12.8 of [RFC7252]), with a particular chosen scope if IPv6 is used.

Through the corresponding responses, the query result is a list of group resources hosted by any server in the mesh network. Each group resource denotes one application group membership of a server. For each application group, the associated CoAP group is obtained as the URI authority component of the corresponding returned link.

If the client wishes to discover resources that a particular server hosts within a particular application group, it may use unicast discovery request(s) to this server.

Full examples are provided in Appendix C.4.

Note that the specific way of using the above methods, including the ways shown by the examples in Appendix C.4, is application-specific. That is, there is currently no standard way of encoding names of application groups and CoAP groups in group URIs and/or CoRE target attributes used with resource links. In particular, the discovery of application groups and CoAP groups through the RD mentioned in Section 2.2.3.1 is only defined for use with an RD, i.e., not directly with CoAP servers as group members.

2.2.4. Group Maintenance

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

For unsecured group communication (see Section 4), i.e., when the NoSec mode is used, addition/removal of CoAP group members is simply done by configuring these devices to start/stop listening to the group IP multicast address on the group’s UDP port.

For secured group communication (see Section 5), the maintenance operations of the protocol Group OSCORE [I-D.ietf-core-oscore-groupcomm] MUST be implemented as well. When using Group OSCORE, CoAP endpoints participating in group

communication are also members of a corresponding OSCORE security group, and thus share common security material. Additional related maintenance operations are discussed in Section 5.2.

3. CoAP Usage in Group Communication

This section specifies the usage of CoAP in group communication, both unsecured and secured. This includes additional support for protocol extensions, such as Observe (see Section 3.7) and block-wise transfer (see Section 3.8).

How CoAP group messages are carried over various transport layers is the subject of Section 3.9. Finally, Section 3.10 covers the interworking of CoAP group communication with other protocols that may operate in the same network.

3.1. Request/Response Model

3.1.1. General

A CoAP client is an endpoint able to transmit CoAP requests and receive CoAP responses. Since the underlying UDP transport supports multiplexing by means of UDP port number, there can be multiple independent CoAP clients operational on a single host. On each UDP port, an independent CoAP client can be hosted. Each independent CoAP client sends requests that use the associated endpoint's UDP port number as the UDP source port number of the request.

All CoAP requests that are sent via IP multicast MUST be Non-confirmable, see Section 8.1 of [RFC7252]. The Message ID in an IP multicast CoAP message is used for optional message deduplication by both clients and servers, as detailed in Section 4.5 of [RFC7252]. A server sends back a unicast response to a CoAP group request. The unicast responses received by the CoAP client may carry a mixture of success (e.g., 2.05 (Content)) and failure (e.g., 4.04 (Not Found)) response codes, depending on the individual server processing results.

3.1.2. Response Suppression

A server MAY suppress its response for various reasons given in Section 8.2 of [RFC7252]. This document adds the requirement that a server SHOULD suppress the response in case of error or in case there is nothing useful to respond, unless the application related to a particular resource requires such a response to be made for that resource.

The CoAP No-Response Option [RFC7967] can be used by a client to influence the default response suppression on the server side. It is RECOMMENDED that a server supporting this option only takes it into account when processing requests targeting selected resources, as useful in the application context.

Any default response suppression by a server SHOULD be performed consistently, as follows: if a request on a resource produces a particular Response Code and this response is not suppressed, then another request on the same resource that produces a response of the same Response Code class is also not suppressed. For example, if a 4.05 (Method Not Allowed) error response code is suppressed by default on a resource, then a 4.15 Unsupported Content-Format error response code is also suppressed by default for that resource.

3.1.3. Repeating a Request

A CoAP client MAY repeat a group request using the same Token value and same Message ID value, in order to ensure that enough (or all) members of the CoAP group have been reached with the request. This is useful in case a number of members of the CoAP group did not respond to the initial request and the client suspects that the request did not reach these group members. However, in case one or more servers did receive the initial request but the response to that request was lost, this repeat does not help to retrieve the lost response(s) if the server(s) implement the optional Message ID based deduplication (Section 4.5 of [RFC7252]).

A CoAP client MAY repeat a group request using the same Token value and a different Message ID, in which case all servers that received the initial request will again process the repeated request since it appears within a new CoAP message. This is useful in case a client suspects that one or more response(s) to its original request were lost and the client needs to collect more, or even all, responses from members of the CoAP group, even if this comes at the cost of the overhead of certain group members responding twice (once to the original request, and once to the repeated request with different Message ID).

3.1.4. Request/Response Matching and Distinguishing Responses

A CoAP client can distinguish the origin of multiple server responses by the source IP address of the message containing the CoAP response and/or any other available application-specific source identifiers contained in the CoAP response payload or CoAP response options, such as an application-level unique ID associated with the server. If secure communication is provided with Group OSCORE (see Section 5), additional security-related identifiers in the CoAP response enable

the client to retrieve the right security material for decrypting each response and authenticating its source.

While processing a response on the client, the source endpoint of the response is not matched to the destination endpoint of the request, since for a group request these will never match. This is specified in Section 8.2 of [RFC7252], with reference to IP multicast.

Also, when UDP transport is used, this implies that a server MAY respond from a UDP port number that differs from the destination UDP port number of the request, although a CoAP server normally SHOULD respond from the UDP port number that equals the destination port number of the request -- following the convention for UDP-based protocols.

In case a single client has sent multiple group requests and concurrent CoAP transactions are ongoing, the responses received by that client are matched to an active request using only the Token value. Due to UDP level multiplexing, the UDP destination port number of the response MUST match to the client endpoint's UDP port number, i.e., to the UDP source port number of the client's request.

3.1.5. Token Reuse

For CoAP group requests, there are additional constraints on the reuse of Token values at the client, compared to the unicast case defined in [RFC7252] and updated by [RFC9175]. Since for CoAP group requests the number of responses is not bound a priori, the client cannot use the reception of a response as a trigger to "free up" a Token value for reuse.

Reusing a Token value too early could lead to incorrect response/request matching on the client, and would be a protocol error. Therefore, the time between reuse of Token values for different group requests MUST be greater than:

```
MIN_TOKEN_REUSE_TIME = (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 was done in [RFC7390], that is: the expected maximum response delay over all servers that the client can send a CoAP group 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 new group 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 MIN_TOKEN_REUSE_TIME = 500 seconds.

When securing CoAP group communication with Group OSCORE [I-D.ietf-core-oscore-groupcomm], secure binding between requests and responses is ensured (see Section 5). Thus, a client may reuse a Token value after it has been freed up, as discussed above and considering a reuse time greater than MIN_TOKEN_REUSE_TIME. If an alternative security protocol for CoAP group communication is used which does not ensure secure binding between requests and responses, a client MUST follow the Token processing requirements as defined in [RFC9175].

Another method to more easily meet the above constraint is to instantiate multiple CoAP clients at multiple UDP ports on the same host. The Token values only have to be unique within the context of a single CoAP client, so using multiple clients can make it easier to meet the constraint.

3.1.6. Client Handling of Multiple Responses With Same Token

Since a client sending a group request with a Token T will accept multiple responses with the same Token T, it is possible in particular that the same server sends multiple responses with the same Token T back to the client. For example, this server might not implement the optional CoAP message deduplication based on Message ID; or it might be acting out of specification as a malicious, compromised or faulty server.

When this happens, the client normally processes at the CoAP layer each of those responses to the same request coming from the same server. If the processing of a response is successful, the client delivers this response to the application as usual.

Then, the application is in a better position to decide what to do, depending on the available context information. For instance, it might accept and process all the responses from the same server, even if they are not Observe notifications (i.e., they do not include an Observe option). Alternatively, the application might accept and process only one of those responses, such as the most recent one from that server, e.g., when this can trigger a change of state within the application.

As part of a long exchange between the client and any of the servers in the CoAP group, the responses considered above are an example of the more general concept elaborated in Section 2 of [I-D.bormann-core-responses].

3.2. Caching

CoAP endpoints that are members of a CoAP group MAY cache responses to a group request as defined in Section 5.6 of [RFC7252]. In particular, these same rules apply to determine the set of request options used as "Cache-Key".

Furthermore, building on what is defined in Section 8.2.1 of [RFC7252]:

* A client sending a GET or FETCH group request MAY update a cache with the responses from the servers in the CoAP group. Then, the client uses both cached-still-fresh and new responses as the result of the group request.

* A client sending a GET or FETCH group request MAY use a response received from a server, to satisfy a subsequent sent request intended to that server on the related unicast request URI. In particular, the unicast request URI is obtained by replacing the authority component of the request URI with the transport-layer source address of the cached response message.

* A client MAY revalidate a cached response by making a GET or FETCH request on the related unicast request URI.

Note that, in the presence of proxies, doing any of the above (optional) unicast requests requires the client to distinguish the different responses to a group request, as well as to distinguish the different origin servers that responded. This in turn requires additional means to provide the client with information about the origin server of each response, e.g., using the forward-proxying method defines in [I-D.ietf-core-groupcomm-proxy].

The following subsections define the freshness model and validation model to use for cached responses, which update the models defined in Sections 5.6.1 and 5.6.2 of [RFC7252], respectively.

3.2.1. Freshness Model

For caching of group communication responses at client endpoints, the same freshness model relying on the Max-Age Option as defined in Section 5.6.1 of [RFC7252] applies, and the multicast caching rules of Section 8.2.1 of [RFC7252] apply except for the one discussed below.

In Section 8.2.1 of [RFC7252] it is stated that, regardless of the presence of cached responses to the group request, the client endpoint will always send out a new group request onto the network because new members may have joined the CoAP group since the last group request to the same CoAP group or resource. That is, a request is never served from cached responses only. This document updates [RFC7252] by adding the following exception case, where a client endpoint MAY serve a request by using cached responses only, and not send out a new group request onto the network:

* The client knows all current CoAP servers that are members of the CoAP group; and, for each group member, the client's cache currently stores a fresh response.

How the client in the case above determines the CoAP servers that are currently members of the CoAP group is out of scope for this document. It may be, for example, via a group manager server, or by monitoring group joining protocol exchanges.

For caching at proxies, the freshness model defined in [I-D.ietf-core-groupcomm-proxy] can be used.

3.2.2. Validation Model

For validation of cached group communication responses at client endpoints, the multicast validation rules in Section 8.2.1 of [RFC7252] apply, except for the last paragraph which states "A GET request to a multicast group MUST NOT contain an ETag option". This document updates [RFC7252] by allowing a group request to contain ETag Options as specified below.

For validation at proxies, the validation model defined in [I-D.ietf-core-groupcomm-proxy] can be used.

3.2.2.1. ETag Option in a Group Request/Response

A client endpoint MAY include one or more ETag Options in a GET or FETCH group request, to validate one or more stored responses it has cached. In case two or more servers in the CoAP group have responded to a previous request to the same resource with an identical ETag value, it is the responsibility of the client to handle this case. In particular, if the client wishes to validate, using a group request, a response from server 1 with an ETag value N, while it does not wish to validate a response from server 2 with the same ETag value N, there is no way to achieve this. In such cases where an identical ETag value is returned by two or more servers, the client, by default, SHOULD NOT include an ETag Option containing that ETag value in a group request.

A server endpoint MUST process an ETag Option in a GET or FETCH group request in the same way it processes an ETag Option for a unicast request. A server endpoint that includes an ETag Option in a response to a group request SHOULD construct the ETag Option value in such a way that the value will be unique to this particular server with a high probability. This practically prevents a collision of the ETag values from different servers in the same CoAP group and application group, which in turn allows the client to effectively validate a particular response of an origin server. This can be accomplished, for example, by embedding a compact ID of the server within the ETag value, where the ID is unique (or unique with a high probability) in the scope of the group.

Note: a legacy CoAP server might treat an ETag Option in a group request as an unrecognized option per Sections 5.4 and 8.2.1 of [RFC7252], causing it to ignore this (elective) ETag Option regardless of its value, and process the request normally as if that ETag Option was not included.

3.3. URI Path Selection

The URI Path used in a group request is preferably a path that is known to be supported across all members of a CoAP group. However, there are valid use cases where a group request is known to be successful only for a subset of the CoAP group. For instance, the subset may include only members of a specific application group, while the members of the CoAP group for which the request is unsuccessful (for example because they are outside the target application group) either respond with an error status code or ignore the group request (see also Section 3.1.2 on response suppression).

3.4. Port Selection for UDP Transport

A server that is a member of a CoAP group listens for CoAP request messages on the group's IP multicast address, usually on the CoAP default UDP port number 5683, or another non-default UDP port number 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 CoAP group and across all CoAP clients sending group requests to that group.

One way to create multiple CoAP groups is using different UDP ports with the same IP multicast address, in case the devices' network stack only supports a limited number of multicast address subscriptions. 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. Also, a constrained network may be additionally burdened in this case with multicast traffic that is eventually discarded at the UDP layer by most nodes.

The port number 5684 is reserved for DTLS-secured unicast 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 number 5683 MUST be supported (see Section 7.1 of [RFC7252]) for the "All CoAP Nodes" CoAP group as detailed in Section 3.9.

3.5. Proxy Operation

This section defines how proxies operate in a group communication scenario. In particular, Section 3.5.1 defines operations of forward-proxies, while Section 3.5.2 defines operations of reverse-proxies. Furthermore, Section 3.5.3 discusses the case where a client sends a group request to multiple proxies at once. Security operations for a proxy are discussed later in Section 5.3.

3.5.1. Forward-Proxies

CoAP enables a client to request a forward-proxy to process a CoAP request on its behalf, as described in Sections 5.7.2 and 8.2.2 of [RFC7252].

When intending to reach a CoAP group through a proxy, the client sends a unicast CoAP group request to the proxy. The group URI where the request has to be forwarded to is specified in the request,

either as a string in the Proxy-URI Option, or through the Proxy-Scheme Option with the group URI constructed from the usual Uri-* Options. Then, the forward-proxy resolves the group URI to a destination CoAP group, i.e., it sends (e.g., multicasts) the CoAP group request to the group URI, receives the responses and forwards all the individual (unicast) responses back to the client.

However, there are certain issues and limitations with this approach:

* The CoAP client component that has sent the unicast CoAP group request to the proxy may be expecting only one (unicast) response, as usual for a CoAP unicast request. Instead, it receives multiple (unicast) responses, potentially leading to fault conditions in the component or to discarding any received responses following the first one. This issue may occur even if the application calling the CoAP client component is aware that the forward-proxy is going to forward the CoAP group request to the group URI.

* Each individual CoAP response received by the client 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 or inside a CoAP option in the response.

* Unlike a CoAP client, the proxy is likely to lack "application context". In particular, the proxy is not expected to know how many members there are in the CoAP group (not even the order of magnitude), how many members of that group will actually respond, or the minimal amount/percentage of those that will respond.

  Therefore, while still capable to forward the group request to the CoAP group and the corresponding responses to the client, the proxy does not know and cannot reliably determine for how long to collect responses, before it stops forwarding them to the client.

  In principle, a CoAP client that is not using a proxy might face the same problems in collecting responses to a group request. However, unlike a CoAP proxy, the client itself would typically have application-specific rules or knowledge on how to handle this situation. For example, a CoAP client could monitor incoming responses and use this information to decide for how long to continue collecting responses

A forward-proxying method using this approach and addressing the issues raised above is defined in [I-D.ietf-core-groupcomm-proxy].

An alternative solution is for the proxy to collect all the individual (unicast) responses to a CoAP group request and then send back only a single (aggregated) response to the client. However, this solution brings up new issues:

* Like for the approach discussed above, the proxy does not know for how long to collect responses before sending back the aggregated response to the client. Analogous considerations apply to this approach too, both on the client and proxy side.

* There is no default format defined in CoAP for aggregation of multiple responses into a single response. Such a format could be standardized based on, for example, the multipart content-format [RFC8710].

Due to the above issues, it is RECOMMENDED that a CoAP Proxy processes a request to be forwarded to a group URI only if it is explicitly enabled to do so. If such functionality is not explicitly enabled, the default response returned to the client is 5.01 Not Implemented. Furthermore, a proxy SHOULD be explicitly configured (e.g., by allow-listing and/or client authentication) to allow proxied CoAP group requests only from specific client(s).

The operation of HTTP-to-CoAP proxies for multicast CoAP requests is specified in Sections 8.4 and 10.1 of [RFC8075]. In this case, the "application/http" media type is used to let the proxy return multiple CoAP responses -- each translated to a HTTP response -- back to the HTTP client. Of course, in this case the HTTP client sending a group URI to the proxy needs to be aware that it is going to receive this format, and needs to be able to decode it into the responses of multiple CoAP servers. Also, the IP source address of each CoAP response cannot be determined anymore from the "application/http" response. The HTTP client may still be able to identify the CoAP servers by other means such as application-specific information in the response payload.

A forward-proxying method for HTTP-to-CoAP proxies addressing the issues raised above is defined in [I-D.ietf-core-groupcomm-proxy].

3.5.2. Reverse-Proxies

CoAP enables the use of a reverse-proxy, as an endpoint that stands in for one or more other server(s), and satisfies requests on behalf of these, doing any necessary translations (see Section 5.7.3 of [RFC7252]).

In a group communication scenario, a reverse-proxy can rely on its configuration and/or on information in a request from a client, in order to determine that a group request has to be sent to servers in a CoAP group, over a one-to-many transport such as IP/UDP multicast.

For example, specific resources on the reverse-proxy could be allocated, each to a specific application group and/or CoAP group. Or alternatively, the application group and/or CoAP group in question could be encoded as URI path segments. The URI path encodings for a reverse-proxy may also use a URI mapping template as described in Section 5.4 of [RFC8075].

The reverse-proxy practically stands in for a CoAP group, thus preventing the client from reaching the group as a whole with a single group request directly addressed to that group (e.g., via multicast). In addition to that, the reverse-proxy may also stand in for each of the individual servers in the CoAP group (e.g., if acting as firewall), thus also preventing the client from individually reaching any server in the group with a unicast request directly addressed to that server.

For a reverse-proxy that sends a request to servers in a CoAP group, the considerations as defined in Section 5.7.3 of [RFC7252] hold, with the following additions:

* The three issues and limitations defined in Section 3.5.1 for a forward proxy apply to a reverse-proxy as well, and have to be addressed, e.g., using the signaling method defined in [I-D.ietf-core-groupcomm-proxy] or other means.

* A reverse-proxy MAY have preconfigured time duration(s) that are used for collecting server responses and forwarding these back to the client. These duration(s) may be set as global configuration or as resource-specific configurations. If there is such preconfiguration, then an explicit signaling of the time period in the client’s request as defined in [I-D.ietf-core-groupcomm-proxy] is not necessarily needed. Note that a reverse-proxy is in an explicit relation with the origin servers it stands in for. Thus, compared to a forward-proxy, it has a much better basis for determining and configuring such time durations.

* A client that is configured to access a reverse-proxy resource (i.e., one that triggers a CoAP group communication request) SHOULD be configured also to handle potentially multiple responses with the same Token value caused by a single request.

That is, the client needs to preserve the Token value used for the request also after the reception of the first response forwarded back by the proxy (see Section 3.1.6) and keep the request open to potential further responses with this Token. This requirement can be met by a combination of client implementation and proper proxied group communication configuration on the client.

* A client might re-use a Token value in a valid new request to the reverse-proxy, while the reverse-proxy still has an ongoing group communication request for this client with the same Token value (i.e., its time period for response collection has not ended yet).

  If this happens, the reverse-proxy MUST stop the ongoing request and associated response forwarding, it MUST NOT forward the new request to the servers in the CoAP group, and it MUST send a 4.00 (Bad Request) error response to the client. The diagnostic payload of the error response SHOULD indicate to the client that the resource is a reverse-proxy resource, and that for this reason immediate Token re-use is not possible.

  If the reverse-proxy supports the signaling protocol of [I-D.ietf-core-groupcomm-proxy] it can include a Multicast-Signaling Option in the error response to convey the reason for the error in a machine-readable way.

For the operation of HTTP-to-CoAP reverse proxies, see the last two paragraphs of Section 3.5.1 which applies also to the case of reverse-proxies.

3.5.3. Single Group Request to Multiple Proxies

A client might send a group request to multiple proxies at once (e.g., over IP multicast), so that each and every of those proxies forwards it to the servers in the CoAP group. Assuming that no message loss occurs and that N proxies receive and forward the group request, this has the following implications.

* Each server receives N copies of the group request, i.e., one copy from each proxy.

* If the NoSec mode is used (see Section 4), each server treats each received copy of the group request as a different request from a different client. Consistently:

- Each server can reply to each of the N received requests with multiple responses over time (see Section 3.1.6). All the responses to the same received request are sent to the same proxy that has forwarded that request, which in turn relays those responses to the client.

- From each proxy, the client receives all the responses to the group request that each server has sent to that proxy. Even in case the client is able to distinguish the different servers originating the responses (e.g., by using the approach defined in [I-D.ietf-core-groupcomm-proxy]), the client would receive the same response content originated by each server N times, as relayed by the N proxies.

* If secure group communication with Group OSCORE is used (see Section 5), each server is able to determine that each received copy of the group request is in fact originated by the same client. In particular, each server is able to determine that all such received requests are copies of exactly the same group request.

  Consistently, each server S accepts only the first copy of the group request received from one of the proxies, say P, while discarding as replay any later copies received from any other proxy.

  After that, the server S can reply to the accepted request with multiple responses over time (see Section 3.1.6). All those responses are sent to the same proxy P that forwarded the only accepted request, and that in turn relays those responses to the client.

  As a consequence, for each server, the client receives responses originated by that server only from one proxy. That is, the client receives a certain response content only once, like in the case with only one proxy.

3.6. Congestion Control

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

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

* A server may choose not to respond to an IP multicast request if there is nothing useful to respond to, e.g., error or empty response (see Section 8.2 of [RFC7252]).

* A server should limit the support for IP multicast requests to specific resources where multicast operation is required (Section 11.3 of [RFC7252]).

* An IP multicast request MUST be Non-confirmable (Section 8.1 of [RFC7252]).

* A response to an IP multicast request SHOULD be Non-confirmable (Section 5.2.3 of [RFC7252]).

* 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]).

This document also defines these measures to be applicable to alternative transports (other than IP multicast), if not defined otherwise.

Independently of the used transport, additional guidelines to reduce congestion risks defined in this document are as follows:

* A server in a constrained network SHOULD only support group requests for resources that have a small representation (where the representation may be retrieved via a GET, FETCH, or POST method in the request). For example, "small" can be defined as a response payload limited to approximately 5% of the IP Maximum Transmit Unit (MTU) size, so that it fits into a single link-layer frame in case IPv6 over Low-Power Wireless Personal Area Networks (6LoWPAN, see Section 3.9.2) is used on the constrained network.

* A server SHOULD minimize the payload size of a response to a group GET or FETCH request 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].

* A server MAY minimize the payload size of a response to a group GET or FETCH request (e.g., on "/.well-known/core") by 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 a CoAP group request to "/.well-known/core" SHOULD support block-wise transfers. See also Section 3.8.

* A client SHOULD be configured to use CoAP groups 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.

3.7. Observing Resources

The CoAP Observe Option [RFC7641] is a protocol extension of CoAP, which 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) group communication.

This section updates [RFC7641] with the use of the Observe Option in a CoAP GET group request, and defines normative behavior for both client and server. Consistent with Section 2.4 of [RFC8132], the same rules apply when using the Observe Option in a CoAP FETCH group request.

Multicast Observe is a useful way to start observing a particular resource on all members of a CoAP group at the same time. Group members that do not have this particular resource or do not allow the GET or FETCH method on it will either respond with an error status -- 4.04 (Not Found) or 4.05 (Method Not Allowed), respectively -- or will silently suppress the response following the rules of Section 3.1.2, depending on server-specific configuration.

A client that sends a group GET or FETCH request with the Observe Option MAY repeat this request using the same Token value and the same Observe Option value, in order to ensure that enough (or all) members of the CoAP group have been reached with the request. This is useful in case a number of members of the CoAP group did not respond to the initial request. The client MAY additionally use the same Message ID in the repeated request, to avoid that members of the CoAP group that had already received the initial request would respond again. Note that using the same Message ID in a repeated request will not be helpful in case of loss of a response message, since the server that responded already will consider the repeated request as a duplicate message. On the other hand, if the client uses a different, fresh Message ID in the repeated request, then all the members of the CoAP group that receive this new message will typically respond again, which increases the network load.

A client that has sent a group GET or FETCH request with the Observe Option MAY follow up by sending a new unicast CON request with the same Token value and same Observe Option value to a particular server, in order to ensure that the particular server receives the request. This is useful in case a specific member of the CoAP group did not respond to the initial group request, although it was expected to. In this case, the client MUST use a Message ID that differs from that of the initial group request message.

Furthermore, consistent with Section 3.3.1 of [RFC7641] and following its guidelines, a client MAY at any time send a new group/multicast GET or FETCH request with the same Token value and same Observe Option value as the original request. This allows the client to verify that it has an up-to-date representation of an observed resource and/or to re-register its interest to observe a resource.

In the above client behaviors, the Token value is kept identical to the initial request to avoid that a client is included in more than one entry in the list of observers (Section 4.1 of [RFC7641]).

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 give the server time to respond.

A server that receives a GET or FETCH request with the Observe Option, for which request processing is successful, SHOULD respond to this request and not suppress the response. If a server adds a client (as a new entry) to the list of observers for a resource due to an Observe request, the server SHOULD respond to this request and SHOULD NOT suppress the response. An exception to the above is the overriding of response suppression according to a CoAP No-Response Option [RFC7967] specified by the client in the GET or FETCH request (see Section 3.1.2).

A server SHOULD have a mechanism to verify the aliveness of its observing clients and the continued interest of these clients in receiving the observe notifications. This can be implemented by sending notifications occasionally using a Confirmable message (see Section 4.5 of [RFC7641] for details). This requirement overrides the regular behavior of sending Non-confirmable notifications in response to a Non-confirmable request.

A client can use the unicast cancellation methods of Section 3.6 of [RFC7641] and stop the ongoing observation of a particular resource on members of a CoAP group. This can be used to remove specific observed servers, or even all servers in the CoAP group (using serial unicast to each known group member). In addition, a client MAY

explicitly deregister from all those servers at once, by sending a group/multicast GET or FETCH request that includes the Token value of the observation to be canceled and includes an Observe Option with the value set to 1 (deregister). In case not all the servers in the CoAP group received this deregistration request, either the unicast cancellation methods can be used at a later point in time or the group/multicast deregistration request MAY be repeated upon receiving another observe response from a server.

For observing at servers that are members of a CoAP group through a CoAP-to-CoAP proxy, the limitations stated in Section 3.5 apply. The method defined in [I-D.ietf-core-groupcomm-proxy] enables group communication including resource observation through proxies and addresses those limitations.

3.8. 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. Consistent with Section 2.5 of [RFC8132], the same can be done with a multicast FETCH request.

To retrieve any further blocks of the resource from a responding server, the client then has to use unicast requests, separately addressing each different server. Also, a server (member of a targeted CoAP group) that needs to respond to a group 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.

A solution for group/multicast block-wise transfer using the Block1 Option is not specified in [RFC7959] nor in the present document. Such a solution would be useful for group FETCH/PUT/POST/PATCH/iPATCH requests, to efficiently distribute a large request payload as multiple blocks to all members of a CoAP group. Multicast usage of Block1 is non-trivial due to potential message loss (leading to missing blocks or missing confirmations), and potential diverging block size preferences of different members of the CoAP group.

[RFC9177] specifies a specialized alternative method for CoAP block-wise transfer. It specifies that "servers MUST ignore multicast requests that contain the Q-Block2 Option".

3.9. Transport Protocols

In this document UDP, both over IPv4 and IPv6, is considered as the default transport protocol for CoAP group communication.

3.9.1. UDP/IPv6 Multicast Transport

CoAP group communication can use UDP over IPv6 as a transport protocol, provided that IPv6 multicast is enabled. IPv6 multicast MAY be supported in a network only for a limited scope. For example, Section 3.10.2 describes the potential limited support of RPL for multicast, depending on how the protocol is configured.

For a CoAP server node that supports resource discovery as defined in Section 2.4 of [RFC7252], the default port number 5683 MUST be supported as per Sections 7.1 and 12.8 of [RFC7252] for the "All CoAP Nodes" multicast CoAP group. An IPv6 CoAP server SHOULD support the "All CoAP Nodes" multicast CoAP group with at least link-local (2), admin-local (4), and site-local (5) scopes. An IPv6 CoAP server on a 6LoWPAN node (see Section 3.9.2) SHOULD also support the realm-local (3) scope.

Note that a client sending an IPv6 multicast CoAP message to a port number that is not supported by the server will not receive an ICMPv6 Port Unreachable error message from that server, because the server does not send it in this case, per Section 2.4 of [RFC4443].

3.9.2. UDP/IPv6 Multicast Transport over 6LoWPAN

In 6LoWPAN [RFC4944] [RFC6282] networks, an IPv6 packet (up to 1280 bytes) may be fragmented into multiple 6LoWPAN fragments, each fragment small enough to be carried over an IEEE 802.15.4 MAC frame (up to 127 bytes).

These 6LoWPAN fragments are exchanged between 6LoWPAN nodes, potentially involving 6LoWPAN routers operating in a multi-hop network topology. Although 6LoWPAN multicast routing protocols usually define mechanisms to compensate for the loss of transmitted fragments (e.g., using link-layer unicast acknowledgements, or repeated link-layer broadcast transmissions as in MPL -- see Section 3.10.3) a fragment may still be lost in transit. The loss of a single fragment implies the loss of the entire IPv6 packet, because the reassembly back into IPv6 packet will fail in that case. Also, if this fragment loss causes the application-layer retransmission of the entire multi-fragment IPv6 packet, it may happen that much of the same data is transmitted yet again over the constrained network.

For this reason, the performance in terms of packet loss and throughput of using larger, multi-fragment multicast IPv6 packets is on average worse than the performance of smaller, single-fragment IPv6 multicast packets. So it is recommended to design application payloads for group communication sufficiently small: a CoAP request sent over multicast over a 6LoWPAN network interface SHOULD fit in a single IEEE 802.15.4 MAC frame, if possible.

On 6LoWPAN networks, multicast CoAP groups can be defined with realm-local scope [RFC7346]. Such a realm-local CoAP group is restricted to the local 6LoWPAN network/subnet. In other words, a multicast request to that CoAP group does not propagate beyond the 6LoWPAN network segment where the request originated. For example, a multicast discovery request can be sent to the realm-local "All CoAP Nodes" IPv6 multicast CoAP group (see Section 3.9.1) in order to discover only CoAP servers on the local 6LoWPAN network.

3.9.3. UDP/IPv4 Multicast Transport

CoAP group communication can use UDP over IPv4 as a transport protocol, provided that IPv4 multicast is enabled. For a CoAP server node that supports resource discovery as defined in Section 2.4 of [RFC7252], the default port number 5683 MUST be supported as per Sections 7.1 and 12.8 of [RFC7252], for the "All CoAP Nodes" IPv4 multicast CoAP group.

Note that a client sending an IPv4 multicast CoAP message to a port number that is not supported by the server will not receive an ICMP Port Unreachable error message from that server, because the server does not send it in this case, per Section 3.2.2 of [RFC1122].

3.9.4. TCP, TLS, and WebSockets

Because it supports unicast only, [RFC8323] (CoAP over TCP, TLS, and WebSockets) has a restricted scope as a transport for CoAP group communication. This is limited to the use of block-wise transfer discussed in Section 3.8.

That is, after the first group request including the Block2 Option and sent over UDP, the following unicast CoAP requests targeting individual servers to retrieve further blocks may be sent over TCP or WebSockets, possibly protected with TLS.

This requires the individually addressed servers to also support CoAP over TCP/TLS/WebSockests for the targeted resource. A server can indicate its support for multiple alternative transports, and practically enable access to its resources through either of them, by using the method defined in [I-D.ietf-core-transport-indication].

3.9.5. Other Transports

CoAP group communication may be used over transports other than UDP/IP multicast. For example broadcast, non-UDP multicast, geocast, serial unicast, etc. In such cases the particular considerations for UDP/IP multicast in this document may need to be applied to that particular transport.

3.10. Interworking with Other Protocols

3.10.1. MLD/MLDv2/IGMP/IGMPv3

A CoAP node that is an IP host (i.e., not an IP router) may be unaware of the specific IP multicast routing/forwarding protocol being used in its network. When such a node needs to join a specific (CoAP) multicast group, the application process would typically subscribe to the particular IP multicast group via an API method of the IP stack on the node. Then the IP stack would execute a particular (e.g., default) method to communicate its subscription to on-link IP (multicast) routers.

The Multicast Listener Discovery Version 2 (MLDv2) protocol [RFC3810] is the standard IPv6 method to communicate multicast subscriptions, when other methods are not defined. The CoAP server nodes then act in the role of MLD Multicast Address Listener. MLDv2 uses link-local communication between Listeners and IP multicast routers. Constrained IPv6 networks such as ones implementing either RPL (see Section 3.10.2) or MPL (see Section 3.10.3) typically do not support MLDv2 as they have their own mechanisms defined for subscribing to multicast groups.

The Internet Group Management Protocol Version 3 (IGMPv3) protocol [RFC3376] is the standard IPv4 method to signal subscriptions to multicast group. This SHOULD be used by members of a CoAP group to subscribe to its multicast IPv4 address on IPv4 networks unless another method is defined for the network interface/technology used.

The guidelines from [RFC6636] on the tuning of MLD for mobile and wireless networks may be useful when implementing MLD in constrained networks.

3.10.2. RPL

IPv6 Routing Protocol for Low-Power and Lossy Networks (RPL) [RFC6550] is an IPv6 based routing protocol suitable for low-power, lossy networks (LLNs). In such a context, CoAP is often used as an application protocol.

If only RPL is used in a network for routing and its optional multicast support is disabled, there will be no IP multicast routing available. Any IPv6 multicast packets in this case will not propagate beyond a single hop (to direct neighbors in the LLN). This implies that any CoAP group request will be delivered to link-local nodes only, for any scope value >= 2 used in the IPv6 destination address.

RPL supports (see Section 12 of [RFC6550]) advertisement of IP multicast destinations using Destination Advertisement Object (DAO) messages and subsequent routing of multicast IPv6 packets based on this. It requires the RPL mode of operation to be set to a mode that supports multicast, for example 3 (Storing mode with multicast support) or 5 (Non-Storing Mode of Operation with ingress replication multicast support) defined in [I-D.ietf-6lo-multicast-registration].

In mode 3, RPL DAO can be used by an RPL/CoAP node that is either an RPL router or RPL Leaf Node, to advertise its CoAP group membership to parent RPL routers. Then, RPL will route any IP multicast CoAP requests over multiple hops to those CoAP servers that are members of the CoAP group.

The same DAO mechanism can be used by an edge router (e.g., 6LBR) to learn CoAP group membership information of the entire RPL network, in case the edge router is also the root of the RPL Destination-Oriented Directed Acyclic Graph (DODAG). This is useful because the edge router learns which IP multicast traffic it needs to selectively pass through from the backbone network into the LLN subnet. In LLNs, such ingress filtering helps to avoid congestion of the resource-constrained network segment, due to IP multicast traffic from the high-speed backbone IP network.

See [I-D.ietf-6lo-multicast-registration] for more details on RPL Mode 5, and on subscribing to IPv6 multicast groups using 6LoWPAN Neighbor Discovery (ND) and the Extended Address Registration Option (EARO) in RPL networks.

3.10.3. MPL

The Multicast Protocol for Low-Power and Lossy Networks (MPL) [RFC7731] can be used for propagation of IPv6 multicast packets throughout a defined network domain, over multiple hops. MPL is designed to work in LLNs and can operate alone or in combination with RPL. The protocol involves a predefined group of MPL Forwarders to collectively distribute IPv6 multicast packets throughout their MPL Domain. An MPL Forwarder may be associated with multiple MPL Domains at the same time. Non-Forwarders will receive IPv6 multicast packets from one or more of their neighboring Forwarders. Therefore, MPL can

be used to propagate a CoAP multicast group request to all members of the CoAP group.

However, a CoAP multicast request to a CoAP group that originated outside of the MPL Domain will not be propagated by MPL -- unless an MPL Forwarder is explicitly configured as an ingress point that introduces external multicast packets into the MPL Domain. Such an ingress point could be located on an edge router (e.g., 6LBR). Methods to configure which multicast groups are to be propagated into the MPL Domain could be:

* Manual configuration on each ingress MPL Forwarder.

* MLDv2 protocol [RFC3810], which works only in case all CoAP servers joining a CoAP group are in link-local communication range of an ingress MPL Forwarder. This is typically not the case on mesh networks.

* Using 6LoWPAN Neighbor Discovery (ND) and Extended Address Registration Option (EARO) as described in [I-D.ietf-6lo-multicast-registration], in a network that supports 6LoWPAN-ND, RPL, and MPL.

* A new/custom protocol to register multicast groups at an ingress MPL Forwarder. This could be for example a CoAP-based protocol offering multicast group subscription features similar to MLDv2.

For security and performance reasons, also other filtering criteria may be defined at an ingress MPL Forwarder. See Section 6.6 for more details.

4. 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].

The NoSec mode does not require and does not make use of a security group. Indications that endpoints can use the NoSec mode MUST NOT rely on setting up and advertising a pseudo security group with name "NoSec" or any of its lowercase/uppercase combinations.

A CoAP server in NoSec mode MUST NOT be accessible through the public Internet. It is NOT RECOMMENDED to use CoAP group communication in NoSec mode.

The possible, exceptional use of the NoSec mode ought to be limited to specific, well-defined steps that are proven to not require security or to not be able to attain it, e.g., early discovery of devices and resources (see Section 6.1).

Before possibly and exceptionally using the NoSec mode in such circumstances, the security implications in Section 6.1 must be very well considered and understood, especially as to the risk and impact of amplification attacks (see Section 6.3). Consistently with such security implications, the use of the NoSec mode should still be avoided whenever possible.

5. Secured Group Communication using Group OSCORE

This section discusses how CoAP group communication can be secured. In particular, Section 5.1 describes how the Group OSCORE security protocol [I-D.ietf-core-oscore-groupcomm] can be used to protect messages exchanged in a CoAP group, while Section 5.2 provides guidance on required maintenance operations for OSCORE groups used as security groups.

5.1. Group OSCORE

The application-layer protocol Object Security for Constrained RESTful Environments (OSCORE) [RFC8613] 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 [RFC9052][RFC9053] to perform encryption operations and protect a CoAP message carried in a COSE object, by using an Authenticated Encryption with Associated Data (AEAD) algorithm. In particular, OSCORE takes as input an unprotected CoAP message and transforms it into a protected CoAP message transporting the COSE object.

OSCORE makes it possible to selectively protect different parts of a CoAP message in different ways, while still allowing intermediaries (e.g., CoAP proxies) to perform their intended functionalities. That is, some message parts are encrypted and integrity protected; other parts are 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 group requests sent by a CoAP client, e.g., over UDP/IP multicast, as well as multiple corresponding CoAP responses sent as (IP) unicast by different CoAP servers. However, the same security 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 ensures source authentication of all messages exchanged within the OSCORE group. That is, the recipient of a CoAP message protected with Group OSCORE is able to securely verify whether the CoAP endpoint that has generated and sent the message is indeed the alleged one. This is achieved by means of two possible methods.

The first method, called group mode, relies on digital signatures. That is, sender devices sign their outgoing messages using their own private key, and embed the signature in the protected CoAP message.

The second method, called pairwise mode, relies on a symmetric key, which is derived from a pairwise shared secret computed from the asymmetric keys of the message sender and recipient. This method is intended for one-to-one messages sent in the security group, such as all responses individually sent by servers, as well as requests addressed to an individual server.

A Group Manager is responsible for managing one or multiple OSCORE groups. In particular, the Group Manager acts as repository of the security group members' authentication credentials including the corresponding public keys; manages, renews, and provides security material in the security group; and handles the join process of new members in the security group.

As defined in [I-D.ietf-ace-oscore-gm-admin], an administrator entity can interact with the Group Manager to create OSCORE groups and specify their configuration (see Section 2.2.2). During the lifetime of the OSCORE group, the administrator can further interact with the Group Manager, in order to possibly update the configuration of the security group and eventually delete the group.

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 [RFC9200].

A CoAP endpoint can discover OSCORE groups and retrieve information to join them through their respective Group Managers by using the method described in [I-D.tiloca-core-oscore-discovery] and based on the CoRE Resource Directory [RFC9176].

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 and using secure group communication is associated with an OSCORE security group, which includes:

* All members of the CoAP group, i.e., the CoAP endpoints that are configured to receive CoAP group messages sent to the particular CoAP group and -- in case of IP multicast transport -- that are listening to the group's multicast IP address on the group's UDP port.

* All further CoAP endpoints configured only as CoAP clients, that may send CoAP group requests to the CoAP group.

5.2. Secure Group Maintenance

As part of group maintenance operations (see Section 2.2.4), additional key management operations are required for an OSCORE group, also depending on the security requirements of the application (see Section 6.2.1). Specifically:

* 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. When this happens, they are securely provided with the security material to use in that OSCORE group.

  Applications may need backward security. That is, they may require that, after having joined an OSCORE group, a new member of that group cannot read the cleartext of messages exchanged in the group prior to its joining, even if it has recorded them.

  In such a case, new security material to use in the OSCORE group has first to be generated and distributed to the current members of that group, before new endpoints are also provided with that new security material upon their joining.

* 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 security material is generated and securely distributed only to the remaining members of the OSCORE group, together with the list of former members removed from that group.

This ensures forward security in the OSCORE group. That is, it ensures that only the members intended to remain in the OSCORE group are able to continue participating in the secure communications within that group, while the evicted ones are not able to participate after the distribution and installation of the new security material.

Also, this ensures that the members intended to remain in the OSCORE group are able to confidently assert the group membership of other sender nodes, when receiving protected messages in the OSCORE group after the distribution and installation of the new security material (see Section 3.2 of [I-D.ietf-core-oscore-groupcomm]).

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 as defined in [I-D.ietf-ace-key-groupcomm-oscore].

5.3. Proxy Security

Different solutions may be selected for secure group communication via a proxy depending on proxy type, use case and deployment requirements. In this section the options based on Group OSCORE are listed.

For a client performing a group communication request via a forward-proxy, end-to-end security should be implemented. The client then creates a group request protected with Group OSCORE and unicasts this to the proxy. The proxy adapts the request from a forward-proxy request to a regular request and multicasts this adapted request to the indicated CoAP group. During the adaptation, the security provided by Group OSCORE persists, in either case of using the group mode or using the pairwise mode. The first leg of communication from client to proxy can optionally be further protected, e.g., by using (D)TLS and/or OSCORE.

For a client performing a group communication request via a reverse-proxy, either end-to-end-security or hop-by-hop security can be implemented. The case of end-to-end security is the same as for the forward-proxy case.

The case of hop-by-hop security is only possible if the proxy can be completely trusted and it is configured as a member of the OSCORE security group(s) that it needs to access, when sending a group request on behalf of clients. The first leg of communication between client and proxy is then protected with a security method for CoAP unicast, such as (D)TLS, OSCORE, or a combination of such methods.

The second leg between proxy and servers is protected using Group OSCORE. This can be useful in applications where for example the origin client does not implement Group OSCORE, or the group management operations are confined to a particular network domain and the client is outside this domain.

For all the above cases, more details on using Group OSCORE are defined in [I-D.ietf-core-groupcomm-proxy].

6. Security Considerations

This section provides security considerations for CoAP group communication, in general and for the particular transport of IP multicast.

6.1. CoAP NoSec Mode

CoAP group communication, if not protected, is vulnerable to all the attacks mentioned in Section 11 of [RFC7252] for IP multicast. Moreover, as also discussed in [I-D.irtf-t2trg-amplification-attacks], the NoSec mode is susceptible to source IP address spoofing, hence amplification attacks are especially feasible and greatly effective, since a single request can result in multiple responses from multiple servers (see Section 6.3).

Therefore, it is NOT RECOMMENDED to use CoAP group communication in NoSec mode, also in order to prevent proliferation of high-volume amplification attacks as further discussed in Section 6.3. The requirement in Section 4 on publically accessible CoAP servers also aims to prevent amplification attacks.

Exceptionally, and only after the security implications have been very well considered and understood, some applications may rely on a limited use of the NoSec mode, when performing specific, well-defined steps that are proven to not require security or to not be able to attain it.

For example, early link-local discovery of devices and resources as part of an onboarding protocol is a typical use case where the NoSec mode or equivalent unsecured mode is used. In such a discovery step, there may be a querying device that needs to discover nearby devices capable of helping it with the network onboarding process. But there are no mutual security relations configured on the querying device and its neighbor devices at the time it performs the early discovery. These relations are configured later in the process based on secure device identities. Alternatively, a new device to be onboarded may wait for advertisements of nearby devices able to help it do the network onboarding process. Also in this case, these messages cannot

be secured initially because the new device does not yet have any security relation configured with devices that are already a member of the network. See [I-D.ietf-anima-constrained-voucher] for an example of an onboarding protocol that can use CoAP multicast for early link-local discovery.

As a further example, the NoSec mode may be useful in simple applications that neither involve nor may have an impact on sensitive data and personal information. These include, e.g., read-only temperature sensors deployed in an environment where a client reads temperature values but does not use this data to control actuators or to perform other automated actions.

In the exception cases where NoSec mode is used, high-volume and harmful amplifications need to be prevented through appropriate and conservative device configurations: taking the early discovery query as an example, only a few CoAP servers are expected to be configured for responding to multicast group requests that are sent for discovery. And the time window during which such unsecured requests are accepted, can be limited as well. Furthermore the scope is also limited: only link-local requests are accepted.

Except for the class of applications discussed above, and all the more so in applications that obviously have hard security requirements (e.g., health monitoring systems and alarm monitoring systems), CoAP group communication MUST NOT be used in NoSec mode.

6.2. 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 specific dynamics of security group membership (e.g., the problem of distributing security keys across large groups with many members that come and go).

CoAP group communication MUST be protected by using Group OSCORE as specified in [I-D.ietf-core-oscore-groupcomm], with the possible exception of specific, well-defined steps that are proven to not require security or to not be able to attain it (e.g., early discovery).

The same security considerations from Section 13 of [I-D.ietf-core-oscore-groupcomm] hold for this specification.

6.2.1. Group Key Management

A key management scheme for secure revocation and renewal of group security material, namely group rekeying, is required to be adopted in OSCORE groups. The key management scheme has to preserve forward security in the OSCORE group, as well as backward security if this is required by the application (see Section 5.2). In particular, the key management scheme MUST comply with the functional steps defined in Section 3.2 of [I-D.ietf-core-oscore-groupcomm].

Even though security group policies vary depending on the specific applications and their security requirements, they SHOULD also take into account:

* The expected amount of time that the key management scheme requires to rekey the OSCORE group.

* The expected frequency of group membership changes in the OSCORE group (i.e., nodes joining and leaving).

* The identity of the specific CoAP endpoints as they join and leave the OSCORE group.

In particular, it can be desirable to not rekey the OSCORE group upon every single membership change, in case group members frequently join and leave, and at the same time a single group rekeying instance takes a non-negligible time to complete.

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

At the same time, the security implications of delaying the rekeying process have to be carefully considered and understood before employing such security group policies.

In fact, this comes at the cost of not continuously preserving backward and forward security, since group rekeying might not occur upon every single change in the OSCORE group membership. That is, most recently joined nodes would have access to the security material used prior to their joining, and thus be able to access past group communications protected with that security material. Similarly, until the OSCORE group is rekeyed, most recently left nodes would retain access to group communications protected with the existing security material.

6.2.2. Source Authentication

Both the group mode and the pairwise mode of Group OSCORE ensure source authentication of messages exchanged by CoAP endpoints through CoAP group communication.

To this end, outgoing messages are either signed by the message sender endpoint with its own private key (group mode), or protected with a symmetric key, which is in turn derived using the asymmetric keys of the message sender and recipient (pairwise mode).

Thus, both modes allow a recipient CoAP endpoint to verify that a message has actually been originated and sent by a specific and identified CoAP endpoint as a member of the OSCORE group.

Note that Group OSCORE does not protect the addressing information about the CoAP endpoint that has sent the message, e.g., the source IP address and port number used when sending the message. Therefore, Group OSCORE does not provide authentication of such source addressing information.

6.2.3. Countering Attacks

As discussed below, Group OSCORE addresses a number of security attacks mentioned in Section 11 of [RFC7252], with particular reference to their execution over IP multicast.

* Since Group OSCORE provides end-to-end confidentiality and integrity of request/response messages, proxies capable of group communication cannot break message protection, and thus cannot act as man-in-the-middle beyond their legitimate duties (see Section 11.2 of [RFC7252]). In fact, intermediaries such as proxies are not assumed to have access to the OSCORE Security Context used by OSCORE group members. Also, with the notable addition of signatures for the group mode, Group OSCORE protects messages using the same procedure as OSCORE (see Sections 8 and 9 of [I-D.ietf-core-oscore-groupcomm]), and especially processes CoAP options according to the same classification in U/I/E classes.

* Group OSCORE limits the feasibility and impact of amplification attacks (see Section 6.3 of this document and Section 11.3 of [RFC7252]), thanks to the handling of protected group requests on the server side. That is, upon receiving a group request protected with Group OSCORE, a server verifies whether the request is not a replay, and whether it has been originated and sent by the alleged CoAP endpoint in the OSCORE group.

  In order to perform the latter check of source authentication, the server either: i) verifies the signature included in the request by using the public key of the client, when the request is protected using the group mode (see Section 8.2 of [I-D.ietf-core-oscore-groupcomm]); or ii) decrypts and verifies the request by means of an additionally derived pairwise key associated with the client, when the request is protected using the pairwise mode (see Section 9.4 of [I-D.ietf-core-oscore-groupcomm]).

  As also discussed in Section 8 of [I-D.ietf-core-oscore-groupcomm], it is recommended that, when failing to decrypt and verify an incoming group request protected with the group mode, a server does not send back any error message in case any of the following holds: the server determines that the request was indeed sent to the whole CoAP group (e.g., over IP multicast); or the server is not able to determine it altogether.

  Such a message processing on the server limits an adversary to leveraging an intercepted group request protected with Group OSCORE, and then altering the source address to be the one of the intended amplification victim.

  Furthermore, the adversary needs to consider a group request that specifically targets a resource for which the CoAP servers are configured to respond. While this can be often correctly inferable from the application context, it is not explicit from

the group request itself, since Group OSCORE protects the Uri-Path and Uri-Query CoAP Options conveying the respective components of the target URI.

As a further mitigation against amplification attacks, a server can also rely on the Echo Option for CoAP defined in [RFC9175] and include it in a response to a group request. By doing so, the server can assert that the alleged sender of the group request (i.e., the CoAP client associated with a certain authentication credential including the corresponding public key) is indeed reachable at the claimed source address of the group request, especially if such an address differs from the one used in previous group requests from the same (authenticated) device. Although responses including the Echo Option do still result in amplification, this is limited in volume compared to when all servers reply with a full response.

Using the Echo Option does not provide authentication of source addressing information about the sender of a CoAP message. Also, using the Echo Option in itself does not provide source authentication of exchanged messages, which is achieved by means of Group OSCORE (see Section 6.2.2). Using the Echo Option together with Group OSCORE allows a CoAP server in the OSCORE group to assert the freshness of CoAP requests received from other members in the group.

* Group OSCORE limits the impact of attacks based on IP spoofing over IP multicast (see Section 11.4 of [RFC7252]). In fact, requests and corresponding responses sent in the OSCORE group can be correctly generated only by CoAP endpoints that are legitimate members of the group.

  Within an OSCORE group, the shared symmetric-key security material strictly provides only group-level authentication. However, source authentication of messages is also ensured, both in the group mode by means of signatures (see Sections 8.1 and 8.3 of [I-D.ietf-core-oscore-groupcomm]), and in the pairwise mode by using additionally derived pairwise keys (see Sections 9.3 and 9.5 of [I-D.ietf-core-oscore-groupcomm]). Thus, recipient endpoints can verify a message to have been originated and sent by the alleged, identifiable CoAP endpoint in the OSCORE group.

  As noted above, the server may additionally rely on the Echo Option for CoAP defined in [RFC9175], in order to verify the aliveness and reachability of the client sending a request from a particular IP address.

* Group OSCORE does not require members of an OSCORE group to be equipped with a good source of entropy for generating security material (see Section 11.6 of [RFC7252]), and thus does not contribute to create an entropy-related attack vector against such (constrained) CoAP endpoints. In particular, the symmetric keys used for message encryption and decryption are derived through the same HMAC-based HKDF scheme used for OSCORE (see Section 3.2 of [RFC8613]). Besides, the OSCORE Master Secret used in such derivation is securely generated by the Group Manager responsible for the OSCORE group, and securely provided to the CoAP endpoints when they join the OSCORE group.

* Group OSCORE prevents making any single member of an OSCORE group a target for subverting security in the whole group (see Section 11.6 of [RFC7252]), even though all group members share (and can derive) the same symmetric-key security material used in the group. In fact, source authentication is always ensured for exchanged CoAP messages, as verifiable to be originated by the alleged, identifiable sender in the OSCORE group. This relies on including a signature computed with a node's individual private key (in the group mode), or on protecting messages with a pairwise symmetric key, which is in turn derived from the asymmetric keys of the sender and recipient CoAP endpoints (in the pairwise mode).

6.3. Risk of Amplification

Section 11.3 of [RFC7252] highlights that CoAP group requests may be used for accidentally or deliberately performing Denial of Service attacks, especially in the form of a high-volume amplification attack, by using all the servers in the CoAP group as attack vectors.

That is, following a group request sent to a CoAP group, each of the servers in the group may reply with a response which is likely larger in size than the group request. Thus, an attacker sending a single group request may achieve a high amplification factor, i.e., a high ratio between the size of the group request and the total size of the corresponding responses intended to the attack victim.

Thus, consistently with Section 11.3 of [RFC7252], a server in a CoAP group:

* SHOULD limit the support for CoAP group requests only to the group resources of the application group(s) using that CoAP group;

* SHOULD NOT accept group requests that cannot be authenticated in some way, i.e., for which it is not possible to securely assert that they have been originated and sent by the alleged, identifiable CoAP endpoint in the OSCORE group;

* SHOULD NOT provide large amplification factors through its responses to a non-authenticated group request, possibly employing CoAP block-wise transfers [RFC7959] to reduce the amount of amplification.

Amplification attacks using CoAP are further discussed in [I-D.irtf-t2trg-amplification-attacks], which also highlights how the amplification factor would become even higher when CoAP group communication is combined with resource observation [RFC7641]. That is, a single group request may result in multiple notification responses from each of the responding servers, throughout the observation lifetime.

Thus, consistently with Section 7 of [RFC7641], a server in a CoAP group MUST strictly limit the number of notifications it sends between receiving acknowledgments that confirm the actual interest of the client in continuing the observation.

Moreover, it is especially easy to perform an amplification attack when the NoSec mode is used. Therefore, also in order to prevent an easy proliferation of high-volume amplification attacks, it is generally NOT RECOMMENDED to use CoAP group communication in NoSec mode (see Section 6.1).

Besides building on very well understood security implications and being limited to specific, well-defined steps that are proven to not require security or to not be able to attain it, possible exceptions should also be limited to use cases where accesses to a group resource have a specific, narrow, and well understood scope, and where only a few CoAP servers (or, ideally, only one) would possibly respond to a group request.

A relevant exceptional example is a CoAP client performing the discovery of hosts such as a group manager or a Resource Directory [RFC9176], by probing for them through a group request sent to the CoAP group. This early, unprotected step is relevant for a CoAP client that does not know the address of such hosts in advance, and that does not have yet configured a mutual security relation with them. In this kind of deployments, such a discovery procedure does not result in a considerable and harmful amplification, since only the few CoAP servers that are the object of discovery are going to respond to the group request targeting that specific resource. In particular, those hosts can be the only CoAP servers in that specific CoAP group (hence listening for group requests sent to that group), and/or the only CoAP servers explicitly configured to respond to group requests targeting specific group resources.

With the exception of such particular use cases, group communications MUST be secured using Group OSCORE [I-D.ietf-core-oscore-groupcomm], see Section 5. As discussed in Section 6.2.3, this limits the feasibility and impact of amplification attacks.

6.4. Replay of Non-Confirmable Messages

Since all requests sent over IP multicast are Non-confirmable, a client might not be able to know if an adversary has actually captured one of its transmitted requests and later re-injected it in the group as a replay to the server nodes. In fact, even if the servers sent back responses to the replayed request, the client would typically not have a valid matching request active anymore, so this attack would not accomplish anything in the client.

If Group OSCORE is used, such a replay attack on the servers is prevented, since a client protects each different request with a different Sequence Number value, which is in turn included as Partial IV in the protected message and takes part in the construction of the AEAD cipher nonce. Thus, a server would be able to detect the replayed request, by checking the conveyed Partial IV against its own replay window in the OSCORE Recipient Context associated with the client.

This requires a server to have a synchronized, up-to-date view of the sequence number used by the client. If such synchronization is lost, e.g., due to a reboot, or suspected so, the server should use the challenge-response synchronization method based on the Echo Option for CoAP defined in [RFC9175] as described in Section 10 of [I-D.ietf-core-oscore-groupcomm], in order to (re-)synchronize with the client’s sequence number.

6.5. Use of CoAP No-Response Option

When CoAP group communication is used in CoAP NoSec (No Security) mode (see Section 4), the CoAP No-Response Option [RFC7967] could be misused by a malicious client to evoke as many responses from servers to a group request as possible, by using the value ’0’ -- Interested in all responses. This might even override the default behavior of a CoAP server to suppress the response in case there is nothing of interest to respond with. Therefore, this option can be used to perform an amplification attack (see Section 6.3).

A proposed mitigation is to only allow this option to relax the standard suppression rules for a resource in case the option is sent by an authenticated client. If sent by an unauthenticated client, the option can be used to expand the classes of responses suppressed compared to the default rules but not to reduce the classes of responses suppressed.

6.6. 6LoWPAN and MPL

In a 6LoWPAN network, the MPL [RFC7731] protocol may be used to forward multicast packets throughout the network. A 6LoWPAN Router that forwards a large IPv6 packet may have a relatively high impact on the occupation of the wireless channel because sending a large packet consists of the transmission of multiple link-layer IEEE 802.15.4 frames. Also, a constrained 6LoWPAN Router may experience a high memory load due to buffering of the large packet -- MPL requires an MPL Forwarder to store the packet for a longer duration, to allow multiple forwarding transmissions to neighboring Forwarders. This could allow an attacker on the 6LoWPAN network or outside the 6LoWPAN network to execute a Denial of Service (DoS) attack by sending large IPv6 multicast packets. This is also an amplification attack in general, because each of potentially multiple MPL Forwarder(s) repeats the transmission of the IPv6 packet potentially multiple times, hence amplifying the original amount of data sent by the attacker considerably.

The amplification factor may be even further increased by the loss of link-layer frames. If one or more of the fragments are not received correctly by an MPL Forwarder during its packet reassembly time window, the Forwarder discards all received fragments and it will likely at a future point in time trigger a neighboring MPL Forwarder to send the IPv6 packet (fragments) again, because its internal state marks this packet (that it failed to received previously) still as a "new" IPv6 packet. Hence this leads to an MPL Forwarder signaling to neighbors its "old" state, triggering additional transmission(s) of all packet fragments.

For these reasons, a large IPv6 multicast packet is a possible attack vector in a Denial of Service (DoS) amplification attack on a 6LoWPAN network. See Section 6.3 of this document and Section 11.3 of [RFC7252] for more details on amplification. To mitigate the risk, applications sending multicast IPv6 requests to 6LoWPAN hosted CoAP servers SHOULD limit the size of the request to avoid 6LoWPAN fragmentation of the request packet. A 6LoWPAN Router or (MPL) multicast forwarder SHOULD deprioritize forwarding for multi-fragment 6LoWPAN multicast packets. 6LoWPAN Border Routers are typical ingress points where multicast traffic enters into a 6LoWPAN network. Specific MPL Forwarders (whether located on a 6LBR or not) may also

be configured as ingress points. Any such ingress point SHOULD implement multicast packet filtering to prevent unwanted multicast traffic from entering a 6LoWPAN network from the outside. For example, it could filter out all multicast packets for which there is no known multicast listener on the 6LoWPAN network. See Section 3.10 for protocols that allow multicast listeners to signal which groups they would like to listen to. As part of multicast packet filtering, the ingress point SHOULD implement a filtering criterion based on the size of the multicast packet. Ingress multicast packets above a defined size may then be dropped or deprioritized.

6.7. Wi-Fi

In a home automation scenario using Wi-Fi, Wi-Fi security should be enabled to prevent rogue nodes from joining. The Customer Premises Equipment (CPE) that enables access to the Internet should also have its IP multicast filters set so that it enforces multicast scope boundaries to isolate local multicast groups from the rest of the Internet (e.g., as per [RFC6092]). In addition, the scope of IP multicast transmissions and listeners should be site-local (5) or smaller. For site-local scope, the CPE will be an appropriate multicast scope boundary point.

6.8. Monitoring

6.8.1. General Monitoring

CoAP group communication can be used to control a set of related devices: for example, to simultaneously turn on all the lights in a room. This intrinsically exposes the communicating devices to some unique monitoring risks, which they are not as vulnerable to when not using group communication.

For example, an attacker might be able to physically see a set of lights turn on in a room. Then, the attacker can correlate an observed CoAP group communication message to the observed coordinated group action -- even if the CoAP message is (partly) encrypted. This will give the attacker side-channel information to plan further attacks (e.g., by determining the members of the CoAP group, some network topology information may be deduced).

6.8.2. Pervasive Monitoring

CoAP traffic is typically used for the Internet of Things, and CoAP (multicast) group communication may specifically be used for conveniently controlling and monitoring critical infrastructure (e.g., lights, alarms, HVAC, electrical grid, etc.).

However, this may be a prime target of pervasive monitoring attacks [RFC7258], which have to be considered as a key additional threat for group communication. For example, an attacker may attempt to record all the CoAP traffic going over a smart grid (i.e., networked electrical utility) and try to determine critical nodes for further attacks. For instance, the source node (controller) sends out CoAP group messages, which easily identifies it as a controller.

CoAP group communication built on top of IP multicast is inherently more vulnerable compared to communications solely relying on IP unicast, since the same packet may be replicated over many multiple links. In particular, this yields a higher probability of packet capture by a pervasive monitoring system, which in turn results in more information available to analyze within the same time interval. Moreover, a single CoAP group request potentially results in multiple CoAP responses, thus further contributing to the information available to analyze.

This requires CoAP group communication solutions that are built on top of IP multicast to pay particular attention to these dangers.

In order to limit the ease of interception of group communication messages, one mitigation is to restrict the scope of IP multicast to the minimal scope that fulfills the application need. See the congestion control recommendations in the last bullet of Section 3.6 to minimize the scope. Thus, for example, realm-local IP multicast scope is always preferred over site-local scope IP multicast, if it fulfills the application needs.

Even if CoAP group communications are encrypted/protected (see Section 5), an attacker may still attempt to capture this traffic and perform an off-line attack in the future.

7. IANA Considerations

This document has no actions for IANA.

8. References

8.1. Normative References

[I-D.ietf-core-oscore-groupcomm]
Tiloca, M., Selander, G., Palombini, F., Mattsson, J. P., and R. Höglund, "Group Object Security for Constrained RESTful Environments (Group OSCORE)", Work in Progress, Internet-Draft, draft-ietf-core-oscore-groupcomm-21, 4 March 2024, <https://datatracker.ietf.org/doc/html/draft-ietf-core-oscore-groupcomm-21>.

[RFC1122] Braden, R., Ed., "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, DOI 10.17487/RFC1122, October 1989, <https://www.rfc-editor.org/rfc/rfc1122>.

[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/rfc/rfc2119>.

[RFC3376] Cain, B., Deering, S., Kouvelas, I., Fenner, B., and A. Thyagarajan, "Internet Group Management Protocol, Version 3", RFC 3376, DOI 10.17487/RFC3376, October 2002, <https://www.rfc-editor.org/rfc/rfc3376>.

[RFC4443] Conta, A., Deering, S., and M. Gupta, Ed., "Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification", STD 89, RFC 4443, DOI 10.17487/RFC4443, March 2006, <https://www.rfc-editor.org/rfc/rfc4443>.

[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/rfc/rfc4944>.

[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/rfc/rfc6282>.

[RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <https://www.rfc-editor.org/rfc/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/rfc/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/rfc/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/rfc/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/rfc/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/rfc/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/rfc/rfc8174>.

[RFC8613] Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security for Constrained RESTful Environments (OSCORE)", RFC 8613, DOI 10.17487/RFC8613, July 2019, <https://www.rfc-editor.org/rfc/rfc8613>.

[RFC9052] Schaad, J., "CBOR Object Signing and Encryption (COSE): Structures and Process", STD 96, RFC 9052, DOI 10.17487/RFC9052, August 2022, <https://www.rfc-editor.org/rfc/rfc9052>.

[RFC9053] Schaad, J., "CBOR Object Signing and Encryption (COSE): Initial Algorithms", RFC 9053, DOI 10.17487/RFC9053, August 2022, <https://www.rfc-editor.org/rfc/rfc9053>.

[RFC9175] Amsüss, C., Preuß Mattsson, J., and G. Selander, "Constrained Application Protocol (CoAP): Echo, Request-Tag, and Token Processing", RFC 9175, DOI 10.17487/RFC9175, February 2022, <https://www.rfc-editor.org/rfc/rfc9175>.

8.2. Informative References

[Californium] Eclipse Foundation, "Eclipse Californium", June 2022, <https://github.com/eclipse/californium>.

[Go-CoAP] Open Connectivity Foundation (OCF), "Go-CoAP", June 2022, <https://github.com/go-ocf/go-coap>.

[I-D.bormann-core-responses]
Bormann, C. and C. Amsüss, "CoAP: Non-traditional response forms", Work in Progress, Internet-Draft, draft-bormann-core-responses-02, 3 March 2024, <https://datatracker.ietf.org/doc/html/draft-bormann-core-responses-02>.

[I-D.ietf-6lo-multicast-registration]
Thubert, P., "IPv6 Neighbor Discovery Multicast and Anycast Address Listener Subscription", Work in Progress, Internet-Draft, draft-ietf-6lo-multicast-registration-18, 16 April 2024, <https://datatracker.ietf.org/doc/html/draft-ietf-6lo-multicast-registration-18>.

[I-D.ietf-ace-key-groupcomm-oscore]
Tiloca, M., Park, J., and F. Palombini, "Key Management for OSCORE Groups in ACE", Work in Progress, Internet-Draft, draft-ietf-ace-key-groupcomm-oscore-16, 6 March 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-ace-key-groupcomm-oscore-16>.

[I-D.ietf-ace-oscore-gm-admin]
Tiloca, M., Höglund, R., Van der Stok, P., and F. Palombini, "Admin Interface for the OSCORE Group Manager", Work in Progress, Internet-Draft, draft-ietf-ace-oscore-gm-admin-11, 4 March 2024, <https://datatracker.ietf.org/doc/html/draft-ietf-ace-oscore-gm-admin-11>.

[I-D.ietf-anima-constrained-voucher]
Richardson, M., Van der Stok, P., Kampanakis, P., and E. Dijk, "Constrained Bootstrapping Remote Secure Key Infrastructure (cBRSKI)", Work in Progress, Internet-Draft, draft-ietf-anima-constrained-voucher-24, 3 March 2024, <https://datatracker.ietf.org/doc/html/draft-ietf-anima-constrained-voucher-24>.

[I-D.ietf-core-coap-pubsub]
Jimenez, J., Koster, M., and A. Keränen, "A publish-subscribe architecture for the Constrained Application Protocol (CoAP)", Work in Progress, Internet-Draft, draft-ietf-core-coap-pubsub-14, 18 April 2024, <https://datatracker.ietf.org/doc/html/draft-ietf-core-coap-pubsub-14>.

[I-D.ietf-core-groupcomm-proxy]
Tiloca, M. and E. Dijk, "Proxy Operations for CoAP Group Communication", Work in Progress, Internet-Draft, draft-

ietf-core-groupcomm-proxy-01, 4 March 2024, <https://datatracker.ietf.org/doc/html/draft-ietf-core-groupcomm-proxy-01>.

[I-D.ietf-core-transport-indication]
Amsüss, C. and M. S. Lenders, "CoAP Transport Indication", Work in Progress, Internet-Draft, draft-ietf-core-transport-indication-05, 18 March 2024, <https://datatracker.ietf.org/doc/html/draft-ietf-core-transport-indication-05>.

[I-D.irtf-t2trg-amplification-attacks]
Mattsson, J. P., Selander, G., and C. Amsüss, "Amplification Attacks Using the Constrained Application Protocol (CoAP)", Work in Progress, Internet-Draft, draft-irtf-t2trg-amplification-attacks-03, 21 February 2024, <https://datatracker.ietf.org/doc/html/draft-irtf-t2trg-amplification-attacks-03>.

[I-D.tiloca-core-oscore-discovery]
Tiloca, M., Amsüss, C., and P. Van der Stok, "Discovery of OSCORE Groups with the CoRE Resource Directory", Work in Progress, Internet-Draft, draft-tiloca-core-oscore-discovery-15, 4 March 2024, <https://datatracker.ietf.org/doc/html/draft-tiloca-core-oscore-discovery-15>.

[libcoap] Bergmann, O., "libcoap", June 2022, <https://github.com/obgm/libcoap>.

[RFC3810] Vida, R., Ed. and L. Costa, Ed., "Multicast Listener Discovery Version 2 (MLDv2) for IPv6", RFC 3810, DOI 10.17487/RFC3810, June 2004, <https://www.rfc-editor.org/rfc/rfc3810>.

[RFC6092] Woodyatt, J., Ed., "Recommended Simple Security Capabilities in Customer Premises Equipment (CPE) for Providing Residential IPv6 Internet Service", RFC 6092, DOI 10.17487/RFC6092, January 2011, <https://www.rfc-editor.org/rfc/rfc6092>.

[RFC6550] Winter, T., Ed., Thubert, P., Ed., Brandt, A., Hui, J., Kelsey, R., Levis, P., Pister, K., Struik, R., Vasseur, JP., and R. Alexander, "RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks", RFC 6550, DOI 10.17487/RFC6550, March 2012, <https://www.rfc-editor.org/rfc/rfc6550>.

[RFC6636] Asaeda, H., Liu, H., and Q. Wu, "Tuning the Behavior of the Internet Group Management Protocol (IGMP) and Multicast Listener Discovery (MLD) for Routers in Mobile and Wireless Networks", RFC 6636, DOI 10.17487/RFC6636, May 2012, <https://www.rfc-editor.org/rfc/rfc6636>.

[RFC7258] Farrell, S. and H. Tschofenig, "Pervasive Monitoring Is an Attack", BCP 188, RFC 7258, DOI 10.17487/RFC7258, May 2014, <https://www.rfc-editor.org/rfc/rfc7258>.

[RFC7346] Droms, R., "IPv6 Multicast Address Scopes", RFC 7346, DOI 10.17487/RFC7346, August 2014, <https://www.rfc-editor.org/rfc/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/rfc/rfc7390>.

[RFC7731] Hui, J. and R. Kelsey, "Multicast Protocol for Low-Power and Lossy Networks (MPL)", RFC 7731, DOI 10.17487/RFC7731, February 2016, <https://www.rfc-editor.org/rfc/rfc7731>.

[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/rfc/rfc7967>.

[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/rfc/rfc8323>.

[RFC8710] Fossati, T., Hartke, K., and C. Bormann, "Multipart Content-Format for the Constrained Application Protocol (CoAP)", RFC 8710, DOI 10.17487/RFC8710, February 2020, <https://www.rfc-editor.org/rfc/rfc8710>.

[RFC9019] Moran, B., Tschofenig, H., Brown, D., and M. Meriac, "A Firmware Update Architecture for Internet of Things", RFC 9019, DOI 10.17487/RFC9019, April 2021, <https://www.rfc-editor.org/rfc/rfc9019>.

[RFC9124] Moran, B., Tschofenig, H., and H. Birkholz, "A Manifest Information Model for Firmware Updates in Internet of Things (IoT) Devices", RFC 9124, DOI 10.17487/RFC9124, January 2022, <https://www.rfc-editor.org/rfc/rfc9124>.

[RFC9176] Amsüss, C., Ed., Shelby, Z., Koster, M., Bormann, C., and P. van der Stok, "Constrained RESTful Environments (CoRE) Resource Directory", RFC 9176, DOI 10.17487/RFC9176, April 2022, <https://www.rfc-editor.org/rfc/rfc9176>.

[RFC9177] Boucadair, M. and J. Shallow, "Constrained Application Protocol (CoAP) Block-Wise Transfer Options Supporting Robust Transmission", RFC 9177, DOI 10.17487/RFC9177, March 2022, <https://www.rfc-editor.org/rfc/rfc9177>.

[RFC9200] Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "Authentication and Authorization for Constrained Environments Using the OAuth 2.0 Framework (ACE-OAuth)", RFC 9200, DOI 10.17487/RFC9200, August 2022, <https://www.rfc-editor.org/rfc/rfc9200>.

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 [RFC9176]; 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.

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 subcase 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] showed an example of discovering a CoRE Resource Directory using CoAP group communication. As defined in [RFC9176], 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 cohesive collection, 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] showed examples of lighting control of a set 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. A set of devices may be defined as a CoAP group, e.g., as intended to cover "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 set of devices 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 set of devices 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

Group communication can be useful to efficiently distribute new software (firmware, image, application, etc.) to a set of multiple devices, e.g., by relying on the SUIT firmware update architecture [RFC9019] and its manifest information model [RFC9124]. In this case, a CoAP group can be defined in terms of the device type of its members: all devices in the target CoAP 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 CoAP 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 a multicast software update, the CoAP group of recipients can be separately notified that there is new software available and coming, using the above network-wide or group notification.

Appendix B. Examples of Group Naming for Application Groups

This section provides examples for the different methods that can be used to name application groups, as defined in Section 2.2.1.2.

The shown examples consider a CoAP group identified by the group hostname grp.example. Its members are CoAP servers listening to the associated IP multicast address ff35:30:2001:db8:f1::8000:1 and port number 5685.

Note that a group hostname is used here to have better-readable examples. As discussed in Section 2.2.1.2 when considering the authority component and its host subcomponent in the Group URI, in

practice an implementation would likely use an IP address literal as the host component of the Group URI, in order to reduce the size of the CoAP request. In particular, the Uri-Host Option can be fully elided in this case.

Also note that the Uri-Port Option does not appear in the examples, since the port number 5685 is already included in the CoAP request's UDP header (which is not shown in the examples).

B.1. Group Naming using the URI Path Component

Figure 3 provides an example where the URI path component is used for naming application groups.

```
Application group name: gp1

Group URI: coap://grp.example:5685/gp/gp1/light?foo=bar

CoAP group request
   Header: GET (T=NON, Code=0.01, MID=0x7d41)
   Uri-Host: grp.example
   Uri-Path: gp
   Uri-Path: gp1
   Uri-Path: light
   Uri-Query: foo=bar
```

Figure 3: Example of application group name in URI path (1/2)

Figure 4 provides a different example, where an IPv6 literal address and the default CoAP port number 5683 are used in the authority component, which yields a compact CoAP request. Also the resource structure is different in this example.

```
Application group name: gp1

Group URI: coap://[ff35:30:2001:db8:f1::8000:1]/g/gp1/li

CoAP group request
   Header: POST (T=NON, Code=0.02, MID=0x7d41)
   Uri-Path: g
   Uri-Path: gp1
   Uri-Path: li
```

Figure 4: Example of application group name in URI path (2/2)

B.2. Group Naming using the URI Query Component

Figure 5 provides an example where the URI query component is used for naming application groups. In particular, it considers the first alternative discussed in Section 2.2.1.2, where the URI query component consists of only one parameter, which has no value and has the name of the application group as its own identifier.

```
Application group name: gp1

Group URI: coap://grp.example:5685/light?gp1

CoAP group request
   Header: GET (T=NON, Code=0.01, MID=0x7d41)
   Uri-Host: grp.example
   Uri-Path: light
   Uri-Query: gp1
```

Figure 5: Example of application group name in URI query (1/2)

Figure 6 provides another example, which considers the second alternative discussed in Section 2.2.1.2. In particular, the URI query component includes a query parameter "gp" as designated indicator, with value the name of the application group.

```
Application group name: gp1

Group URI: coap://grp.example:5685/light?foo=bar&gp=gp1

CoAP group request
   Header: GET (T=NON, Code=0.01, MID=0x7d41)
   Uri-Host: grp.example
   Uri-Path: light
   Uri-Query: foo=bar
   Uri-Query: gp=gp1
```

Figure 6: Example of application group name in URI query (2/2)

B.3. Group Naming using the URI Authority Component

Figure 7 provides an example where the URI authority component as a whole is used for naming application groups.

```
Application group name: grp.example:5685

Group URI: coap://grp.example:5685/light?foo=bar

CoAP group request
   Header: GET (T=NON, Code=0.01, MID=0x7d41)
   Uri-Host: grp.example
   Uri-Path: light
   Uri-Query: foo=bar
```

Figure 7: Example of application group name in URI authority

B.4. Group Naming using the URI Host Subcomponent

Figure 8 provides an example where the URI host subcomponent of the URI authority component is used for naming application groups.

```
Application group name: grp.example

Group URI: coap://grp.example:5685/light?foo=bar

CoAP group request
   Header: GET (T=NON, Code=0.01, MID=0x7d41)
   Uri-Host: grp.example
   Uri-Path: light
   Uri-Query: foo=bar
```

Figure 8: Example of application group name in URI host

B.5. Group Naming using the URI Port Subcomponent

Figure 9 provides an example where the URI port subcomponent of the URI authority component is used for naming application groups.

```
Application group name: grp1, as inferable from port number 5685

Group URI: coap://grp.example:5685/light?foo=bar

CoAP group request
   Header: GET(T=NON, Code=0.01, MID=0x7d41)
   Uri-Host: grp.example
   Uri-Path: light
   Uri-Query: foo=bar
```

Figure 9: Example of application group name in URI port

B.6. Group Naming using a Custom CoAP Option

Figure 10 provides an example where a new, custom CoAP Option, namely App-Group-Name, is used for naming application groups.

```
Application group name: grp1

Group URI: coap://grp.example:5685/light?foo=bar

CoAP group request
   Header: GET (T=NON, Code=0.01, MID=0x7d41)
   Uri-Host: grp.example
   Uri-Path: light
   Uri-Query: foo=bar
   App-Group-Name: grp1  // new (e.g., custom) CoAP option
```

Figure 10: Example of application group name in a new CoAP Option

Appendix C. Examples of Group Discovery from CoAP Servers

This section provides examples for the different methods that a CoAP client can use to discover application groups and CoAP groups by interacting with CoAP servers, as defined in Section 2.2.3.2.

The examples build on the same assumptions considered in Section 2.2.3.2. In addition, a CoAP group is used and is identified by the URI authority grp.example:5685.

C.1. Application Groups Associated with a CoAP Group

Figure 11 provides an example where a CoAP client discovers all the application groups associated with a specific CoAP group.

As a result, the client gains knowledge of: i) the set of servers that are members of the specified CoAP group and member of any of the associated application groups; ii) for each of those servers, the name of the application groups where the server is a member and that are associated with the CoAP group.

Each of the servers S1 and S2 is identified by the IP source address of the CoAP response. If the client wishes to discover resources that a particular server hosts within a particular application group, it may use unicast discovery request(s) to this server, i.e., to its respective unicast IP address. Alternatively the client may use the discovered group resource type (e.g., rt=g.light) to infer which resources are present below the group resource.

```
// Request to all members of the CoAP group
Req: GET coap://grp.example:5685/.well-known/core?rt=g.*

// Response from server S1, as member of:
//   - The CoAP group "grp.example:5685"
//   - The application group "gp1"
Res: 2.05 (Content)
Content-Format: 40
Payload:
</gp/gp1>;rt=g.light

// Response from server S2, as member of:
// - The CoAP group "grp.example:5685"
// - The application groups "gp1" and "gp2"
Res: 2.05 (Content)
Content-Format: 40
Payload:
</gp/gp1>;rt=g.light,
</gp/gp2>;rt=g.temp
```

Figure 11: Discovery of application groups associated with a CoAP group

C.2. Members of a Given Application Group

Figure 12 provides an example where a CoAP client discovers the CoAP servers that are members of a specific application group and the CoAP group associated with the application group.

Note that, unlike in the example shown in Appendix C.1, now the servers need to respond with an absolute URI and not a relative URI. This is necessary because the responding CoAP endpoint serving the Link Format document (on port 5683) is a different CoAP endpoint from the one hosting the group resource "gp1" (on port 5685). Due to this situation, the responding server includes the full (absolute) URI in the Link Format response from which the client can conveniently gain knowledge of the CoAP group.

Also note that a server could equally well respond with the literal IPv6 multicast address within square brackets instead of the CoAP group name "grp.example". In that case, the client would still gain knowledge of the CoAP group, albeit in a different representation.

```
// Request to realm-local members of the application group "gp1"
Req: GET coap://[ff03::fd]/.well-known/core?href=/gp/gp1

// CoAP response from server S1, as member of:
//   - The CoAP group "grp.example:5685"
//   - The application group "gp1"
Res: 2.05 (Content)
Content-Format: 40
Payload:
<coap://grp.example:5685/gp/gp1>;rt=g.light

// CoAP response from server S2, as member of:
// - The CoAP group "grp.example:5685"
// - The application groups "gp1"
Res: 2.05 (Content)
Content-Format: 40
Payload:
<coap://grp.example:5685/gp/gp1>;rt=g.light
```

Figure 12: Discovery of members of an application group, together with the associated CoAP group

C.3. Members of any Application Group of a Given Type

Figure 13 provides an example where a CoAP client discovers the CoAP servers that are members of any application group of a specific type, and the CoAP group associated with those application groups.

```
// Request to realm-local members of application groups
// with group type "g.temp"
Req: GET coap://[ff03::fd]/.well-known/core?rt=g.temp

// Response from server S1, as member of:
//   - The CoAP group "grp.example:5685"
//   - The application group "gp1" of type "g.temp"
Res: 2.05 (Content)
Content-Format: 40
Payload:
<coap://grp.example:5685/gp/gp1>;rt=g.temp

// Response from server S2, as member of:
//   - The CoAP group "grp.example:5685"
//   - The application groups "gp1" and "gp2" of type "g.temp"
Res: 2.05 (Content)
Content-Format: 40
Payload:
<coap://grp.example:5685/gp/gp1>;rt=g.temp,
<coap://grp.example:5685/gp/gp2>;rt=g.temp
```

Figure 13: Discovery of members of application groups of a specified type, and of the associated CoAP group

C.4. Members of any Application Group in the Network

Figure 14 provides an example where a CoAP client discovers the CoAP servers that are members of any application group configured in the 6LoWPAN wireless mesh network of the client, and the CoAP group associated with each application group. In this example, the scope is realm-local to address all servers in the current 6LoWPAN wireless mesh network of the client.

```
// Request to realm-local members of any application group
Req: GET coap://[ff03::fd]/.well-known/core?rt=g.*

// Response from server S1, as member of:
//   - The CoAP groups "grp.example:5685" and "grp2.example"
//   - The application groups "gp1" and "gp5"
Res: 2.05 (Content)
Content-Format: 40
Payload:
<coap://grp.example:5685/gp/gp1>;rt=g.light,
<coap://grp2.example/gp/gp5>;rt=g.lock

// Response from server S2, as member of:
//   - The CoAP group "grp.example:5685"
//   - The application groups "gp1" and "gp2"
Res: 2.05 (Content)
Content-Format: 40
Payload:
<coap://grp.example:5685/gp/gp1>;rt=g.light,
<coap://grp.example:5685/gp/gp2>;rt=g.light

// Response from server S3, as member of:
//   - The CoAP group "grp2.example"
//   - The application group "gp5"
Res: 2.05 (Content)
Content-Format: 40
Payload:
<coap://grp2.example/gp/gp5>;rt=g.lock
```

Figure 14: Discovery of the resources and members of any application group, and of the associated CoAP group

Alternatively, some applications may use the "rt" attribute on a parent resource to denote support for a particular REST API to access child resources.

For instance, Figure 15 provides a different example where a custom Link Format attribute "gpt" is used to denote the group type within the scope of the application/system. An alternative, shorter encoding (not shown in the figure) is to use only the value "1" for each "gpt" attribute, in order to denote that the resource is of type application group. In that case, information about the semantics/API of the group resource is disclosed only via the "rt" attribute as shown in the figure.

```
// Request to realm-local members of any application group
Req: GET coap://[ff03::fd]/.well-known/core?gpt=*

// Response from server S1, as member of:
//   - The CoAP groups "grp.example:5685" and "grp2.example"
//   - The application groups "gp1" and "gp5"
Res: 2.05 (Content)
Content-Format: 40
Payload:
<coap://grp.example:5685/gp/gp1>;rt=oic.d.light;gpt=light,
<coap://grp2.example/gp/gp5>;rt=oic.d.smartlock;gpt=lock

// Response from server S2, as member of:
//   - The CoAP group "grp.example:5685"
//   - The application groups "gp1" and "gp2"
Res: 2.05 (Content)
Content-Format: 40
Payload:
<coap://grp.example:5685/gp/gp1>;rt=oic.d.light;gpt=light,
<coap://grp.example:5685/gp/gp2>;rt=oic.d.light;gpt=light

// Response from server S3, as member of:
//   - The CoAP group "grp2.example"
//   - The application group "gp5"
Res: 2.05 (Content)
Content-Format: 40
Payload:
<coap://grp2.example/gp/gp5>;rt=oic.d.smartlock;gpt=lock
```

Figure 15: Example of using a custom 'gpt' link attribute to denote group type

Appendix D. Examples of Message Exchanges

This section provides examples of different message exchanges when CoAP is used with group communication. The examples consider:

* A client with address ADDR_CLIENT and port number PORT_CLIENT.

* A CoAP group associated with the IP multicast address ADDR_GRP and port number PORT_GRP.

* An application group "gp1" associated with the CoAP group above.

* Three servers A, B, and C, all of which are members of the CoAP group above and of the application group "gp1". Each server X (with X equal to A, B, or C): listens to its own address ADDR_X and port number PORT_X; and listens to the address ADDR_GRP and port number PORT_GRP. For each server its PORT_X may be different from PORT_GRP or may be equal to it, in general.

In Figure 16, the client sends a Non-confirmable GET request to the CoAP group, targeting the resource "temperature" in the application group "gp1". All servers reply with a 2.05 (Content) response, although the response from server B is lost. As source port number of their response, servers A and B use the destination port number of the request, i.e., PORT_GRP. Instead, server C uses its own port number PORT_C.

```
 Client              A  B  C
    |                |  |  |
    |  GET           |  |  |
    +-------+------->|  |  | Source: ADDR_CLIENT:PORT_CLIENT
    |        \       |  |  | Destination: ADDR_GRP:PORT_GRP
    |         `.------->|  | Header: GET (T=NON, Code=0.01, MID=0x7d41)
    |           `.   |  |  | Token: 0x86
    |             `------->| Uri-Path: "gp"
    |                |  |  | Uri-Path: "gp1"
    |                |  |  | Uri-Path: "temperature"
    |                |  |  |
    |<---------------+  |  | Source: ADDR_A:PORT_GRP
    |      2.05      |  |  | Destination: ADDR_CLIENT:PORT_CLIENT
    |                |  |  | Header: 2.05 (T=NON, Code=2.05, MID=0x60b1)
    |                |  |  | Token: 0x86
    |                |  |  | Payload: "22.3 C"
    |                |  |  |
    |   X------------+  |  | Source: ADDR_B:PORT_GRP
    |      2.05      |  |  | Destination: ADDR_CLIENT:PORT_CLIENT
    |                |  |  | Header: 2.05 (T=NON, Code=2.05, MID=0x01a0)
    |                |  |  | Token: 0x86
    |                |  |  | Payload: "20.9 C"
    |                |  |  |
    |                |  |  |
    |<---------------------+ Source: ADDR_C:PORT_C
    |      2.05      |  |  | Destination: ADDR_CLIENT:PORT_CLIENT
    |                |  |  | Header: 2.05 (T=NON, Code=2.05, MID=0x952a)
    |                |  |  | Token: 0x86
    |                |  |  | Payload: "21.0 C"
    |                |  |  |
```

Figure 16: Example of Non-confirmable group request, followed by Non-confirmable Responses

In Figure 17, the client sends a Non-confirmable GET request to the CoAP group, targeting and requesting to observe the resource "temperature" in the application group "gp1". All servers reply with a 2.05 (Content) notification response. As source port number of their response, servers A and B use the destination port number of the request, i.e., PORT_GRP. Instead, server C uses its own port number PORT_C. Some time later, all servers send a 2.05 (Content) notification response, with the new representation of the "temperature" resource as payload.

```
 Client               A  B  C
    |                 |  |  |
    |  GET            |  |  |
    +-------+-------->|  |  | Source: ADDR_CLIENT:PORT_CLIENT
    |        \        |  |  | Destination: ADDR_GRP:PORT_GRP
    |         `.------->|  | Header: GET (T=NON, Code=0.01, MID=0x7d41)
    |           `.    |  |  | Token: 0x86
    |             `------->| Observe: 0 (register)
    |                 |  |  | Uri-Path: "gp"
    |                 |  |  | Uri-Path: "gp1"
    |                 |  |  | Uri-Path: "temperature"
    |                 |  |  |
    |<---------------+   |  | Source: ADDR_A:PORT_GRP
    |      2.05       |  |  | Destination: ADDR_CLIENT:PORT_CLIENT
    |                 |  |  | Header: 2.05 (T=NON, Code=2.05, MID=0x60b1)
    |                 |  |  | Token: 0x86
    |                 |  |  | Observe: 3
    |                 |  |  | Payload: "22.3 C"
    |                 |  |  |
    |<------------------+    | Source: ADDR_B:PORT_GRP
    |      2.05       |  |  | Destination: ADDR_CLIENT:PORT_CLIENT
    |                 |  |  | Header: 2.05 (T=NON, Code=2.05, MID=0x01a0)
    |                 |  |  | Token: 0x86
    |                 |  |  | Observe: 13
    |                 |  |  | Payload: "20.9 C"
    |                 |  |  |
    |<---------------------+ Source: ADDR_C:PORT_C
    |      2.05       |  |  | Destination: ADDR_CLIENT:PORT_CLIENT
    |                 |  |  | Header: 2.05 (T=NON, Code=2.05, MID=0x952a)
    |                 |  |  | Token: 0x86
    |                 |  |  | Observe: 23
    |                 |  |  | Payload: "21.0 C"
    |                 |  |  |

    // The temperature changes ...

    |                 |  |  |
    |<---------------+   |  | Source: ADDR_A:PORT_GRP
    |      2.05       |  |  | Destination: ADDR_CLIENT:PORT_CLIENT
    |                 |  |  | Header: 2.05 (T=NON, Code=2.05, MID=0x60b2)
    |                 |  |  | Token: 0x86
    |                 |  |  | Observe: 7
    |                 |  |  | Payload: "32.3 C"
    |                 |  |  |
    |<------------------+    | Source: ADDR_B:PORT_GRP
    |      2.05       |  |  | Destination: ADDR_CLIENT:PORT_CLIENT
    |                 |  |  | Header: 2.05 (T=NON, Code=2.05, MID=0x01a1)
    |                 |  |  | Token: 0x86
```

```
    |                 |   |   | Observe: 18
    |                 |   |   | Payload: "30.9 C"
    |                 |   |   |
    |<----------------------+ Source: ADDR_C:PORT_C
    |      2.05       |   |   | Destination: ADDR_CLIENT:PORT_CLIENT
    |                 |   |   | Header: 2.05 (T=NON, Code=2.05, MID=0x952b)
    |                 |   |   | Token: 0x86
    |                 |   |   | Observe: 29
    |                 |   |   | Payload: "31.0 C"
    |                 |   |   |
```

Figure 17: Example of Non-confirmable Observe group request, followed by Non- confirmable Responses as Observe notifications

In Figure 18, the client sends a Non-confirmable GET request to the CoAP group, targeting the resource "log" in the application group "gp1", and requesting a blockwise transfer. All servers reply with a 2.05 (Content) response including the first block. As source port number of its response, each server uses its own port number. After obtaining the first block, the client requests the following blocks separately from each server, by means of unicast exchanges.

```
 Client              A  B  C
    |                |  |  |
    |  GET           |  |  |
    +-------+------->|  |  | Source: ADDR_CLIENT:PORT_CLIENT
    |        \       |  |  | Destination: ADDR_GRP:PORT_GRP
    |         `.------->|  | Header: GET (T=NON, Code=0.01, MID=0x7d41)
    |           `.   |  |  | Token: 0x86
    |             `------->| Uri-Path: "gp"
    |                |  |  | Uri-Path: "gp1"
    |                |  |  | Uri-Path: "log"
    |                |  |  | Block2: 0/0/64
    |                |  |  |
    <---------------+   |  | Source: ADDR_A:PORT_A
    |      2.05      |  |  | Destination: ADDR_CLIENT:PORT_CLIENT
    |                |  |  | Header: 2.05 (T=NON, Code=2.05, MID=0x60b1)
    |                |  |  | Token: 0x86
    |                |  |  | Block2: 0/1/64
    |                |  |  | Payload: 0x0a00 ...
    |                |  |  |
    <------------------+   | Source: ADDR_B:PORT_B
    |      2.05      |  |  | Destination: ADDR_CLIENT:PORT_CLIENT
    |                |  |  | Header: 2.05 (T=NON, Code=2.05, MID=0x01a0)
    |                |  |  | Token: 0x86
    |                |  |  | Block2: 0/1/64
    |                |  |  | Payload: 0x0b00 ...
    |                |  |  |
```

```
    |<---------------------+ Source: ADDR_C:PORT_C
    |      2.05            | | | Destination: ADDR_CLIENT:PORT_CLIENT
    |                      | | | Header: 2.05 (T=NON, Code=4.04, MID=0x952a)
    |                      | | | Token: 0x86
    |                      | | | Block2: 0/1/64
    |                      | | | Payload: 0x0c00 ...
    |                      | | |
    |      GET             | | |
    +---------------->     | | | Source: ADDR_CLIENT:PORT_CLIENT
    |                      | | | Destination: ADDR_A:PORT_A
    |                      | | | Header: GET (T=CON, Code=0.01, MID=0x7d42)
    |                      | | | Token: 0xa6
    |                      | | | Uri-Path: "gp"
    |                      | | | Uri-Path: "gp1"
    |                      | | | Uri-Path: "log"
    |                      | | | Block2: 1/0/64
    |                      | |
    <---------------+      | | | Source: ADDR_A:PORT_A
    |      2.05            | | | Destination: ADDR_CLIENT:PORT_CLIENT
    |                      | | | Header: 2.05 (T=ACK, Code=2.05, MID=0x7d42)
    |                      | | | Token: 0xa6
    |                      | | | Block2: 1/1/64
    |                      | | | Payload: 0x0a01 ...
    |                      | | |
    |      GET             | | |
    +---------------->     | | | Source: ADDR_CLIENT:PORT_CLIENT
    |                      | | | Destination: ADDR_A:PORT_A
    |                      | | | Header: GET (T=CON, Code=0.01, MID=0x7d43)
    |                      | | | Token: 0xa7
    |                      | | | Uri-Path: "gp"
    |                      | | | Uri-Path: "gp1"
    |                      | | | Uri-Path: "log"
    |                      | | | Block2: 2/0/64
    |                      | |
    <---------------+      | | | Source: ADDR_A:PORT_A
    |      2.05            | | | Destination: ADDR_CLIENT:PORT_CLIENT
    |                      | | | Header: 2.05 (T=ACK, Code=2.05, MID=0x7d43)
    |                      | | | Token: 0xa7
    |                      | | | Block2: 2/0/64
    |                      | | | Payload: 0x0a02 ...
    |                      | |
    |      GET             | |
    +------------------>   | | | Source: ADDR_CLIENT:PORT_CLIENT
    |                      | | | Destination: ADDR_B:PORT_B
    |                      | | | Header: GET (T=CON, Code=0.01, MID=0x7d44)
    |                      | | | Token: 0xb6
    |                      | | | Uri-Path: "gp"
    |                      | | | Uri-Path: "gp1"
```

```
    |                 |   |   | Uri-Path: "log"
    |                 |   |   | Block2: 1/0/64
    |                 |   |   |
    |<------------------+     | Source: ADDR_B:PORT_B
    |      2.05       |   |   | Destination: ADDR_CLIENT:PORT_CLIENT
    |                 |   |   | Header: 2.05 (T=ACK, Code=2.05, MID=0x7d44)
    |                 |   |   | Token: 0xb6
    |                 |   |   | Block2: 1/1/64
    |                 |   |   | Payload: 0x0b01 ...
    |                 |   |   |
    |      GET        |   |   |
    +------------------>|     | Source: ADDR_CLIENT:PORT_CLIENT
    |                 |   |   | Destination: ADDR_C:PORT_B
    |                 |   |   | Header: GET (T=CON, Code=0.01, MID=0x7d45)
    |                 |   |   | Token: 0xb7
    |                 |   |   | Uri-Path: "gp"
    |                 |   |   | Uri-Path: "gp1"
    |                 |   |   | Uri-Path: "log"
    |                 |   |   | Block2: 2/0/64
    |                 |   |   |
    |<------------------+     | Source: ADDR_B:PORT_B
    |      2.05       |   |   | Destination: ADDR_CLIENT:PORT_CLIENT
    |                 |   |   | Header: 2.05 (T=ACK, Code=2.05, MID=0x7d45)
    |                 |   |   | Token: 0xb7
    |                 |   |   | Block2: 2/0/64
    |                 |   |   | Payload: 0x0b02 ...
    |                 |   |   |
    |      GET        |   |   |
    +---------------------->| Source: ADDR_CLIENT:PORT_CLIENT
    |                 |   |   | Destination: ADDR_C:PORT_C
    |                 |   |   | Header: GET (T=CON, Code=0.01, MID=0x7d46)
    |                 |   |   | Token: 0xc6
    |                 |   |   | Uri-Path: "gp"
    |                 |   |   | Uri-Path: "gp1"
    |                 |   |   | Uri-Path: "log"
    |                 |   |   | Block2: 1/0/64
    |                 |   |   |
    |<----------------------+ | Source: ADDR_C:PORT_C
    |      2.05       |   |   | Destination: ADDR_CLIENT:PORT_CLIENT
    |                 |   |   | Header: 2.05 (T=ACK, Code=2.05, MID=0x7d46)
    |                 |   |   | Token: 0xc6
    |                 |   |   | Block2: 1/1/64
    |                 |   |   | Payload: 0x0c01 ...
    |                 |   |   |
    |      GET        |   |   |
    +---------------------->| Source: ADDR_CLIENT:PORT_CLIENT
    |                 |   |   | Destination: ADDR_C:PORT_C
    |                 |   |   | Header: GET (T=CON, Code=0.01, MID=0x7d47)
```

```
     |                 |   |   | Token: 0xc7
     |                 |   |   | Uri-Path: "gp"
     |                 |   |   | Uri-Path: "gp1"
     |                 |   |   | Uri-Path: "log"
     |                 |   |   | Block2: 2/0/64
     |                 |   |   |
     |<---------------------+ Source: ADDR_C:PORT_C
     |      2.05       |   |   | Destination: ADDR_CLIENT:PORT_CLIENT
     |                 |   |   | Header: 2.05 (T=ACK, Code=2.05, MID=0x7d47)
     |                 |   |   | Token: 0xc7
     |                 |   |   | Block2: 2/0/64
     |                 |   |   | Payload: 0x0c02 ...
     |                 |   |   |
```

Figure 18: Example of Non-confirmable group request starting a blockwise transfer, followed by Non-confirmable Responses with the first block. The transfer continues over confirmable unicast exchanges

Appendix E. Document Updates

This section is to be removed before publishing as an RFC.

RFC EDITOR: PLEASE REMOVE THIS SECTION.

E.1. Version -10 to -11

* Updated references.

* Editorial improvements.

E.2. Version -09 to -10

* Clarified use of NoSec for 'early discovery' and refer to cBRSKI.

* Clarified mandatory vs optional elements in CoAP group and CoAP group URI.

* Replaced "GROUPNAME" with "APPNAME".

* Explicitly mentioning of the type of group (CoAP/application/security).

* Use of .example for example hostnames.

* The name of a security group is not necessarily a string.

* Changed "has to" to "should" for enforcing access control based on membership to security groups.

* Used normative language for policies about group rekeying.

* Further stressed that group communication ought to be secured.

* Avoid calling applications as "(non-)sensitive" and "(non-)critical".

* Clarification on source authentication, source addresses, and Echo Option.

* Editorial fixes and improvements.

E.3. Version -08 to -09

* Multiple responses in long exchanges are non-traditional responses.

* Updated references.

E.4. Version -07 to -08

* Updated references.

E.5. Version -06 to -07

* Updated list of changes to other documents.

* Added real-life context and clarifications to examples.

* Clarified aliasing of CoAP group names.

* Clarified use of security group names.

* Clarified response suppression.

* Clarified response revalidation.

* Clarified limitations and peculiarities when using proxies.

* Discussed the case of group request sent to multiple proxies at once.

* Discussed limited use of reliable transports with block-wise transfer.

* Revised text on joining CoAP groups and multicast routing.

* Clarified use/avoidance of the CoAP NoSec mode.

* Moved examples of application group naming and group discovery to appendix sections.

* Revised list of references.

* Updated list of implementations supporting group communication.

* Editorial improvements.

E.6. Version -05 to -06

* Harmonized use of "group URI".

* Clarifications about different group types.

* Revised methods to perform group naming.

* Revised methods to discover application groups and CoAP groups.

* Explicit difference between "authentication credential" and "public key".

* Added examples of application group naming.

* Added examples of application/CoAP group discovery.

* Added examples of message exchanges.

* Reference to draft-mattsson-core-coap-attacks replaced with reference to draft-mattsson-t2trg-amplification-attacks.

* Editorial improvements.

E.7. Version -04 to -05

* Clarified changes to other documents.

* Clarified relation between different group types.

* Clarified discovery of application groups.

* Discussed methods to express application group names in requests.

* Revised and extended text on the NoSec mode and amplification attacks.

* Rephrased backward/forward security as properties.

* Removed appendix on Multi-ETag Option for response revalidation.

* Editorial improvements.

E.8. Version -03 to -04

* Multi-ETag Option for response revalidation moved to appendix.

* ETag Option usage added.

* Q-Block Options added in the block-wise transfer section.

* Caching at proxies moved to draft-tiloca-core-groupcomm-proxy.

* Client-Proxy response revalidation with the Group-ETag Option moved to draft-tiloca-core-groupcomm-proxy.

* Security considerations on amplification attacks.

* Generalized transport protocols to include others than UDP/IP multicast; and security protocols other than Group OSCORE.

* Overview of security cases with proxies.

* Editorial improvements.

E.9. Version -02 to -03

* Multiple responses from same server handled at the application.

* Clarifications about issues with forward-proxies.

* Operations for reverse-proxies.

* Caching of responses at proxies.

* Client-Server response revalidation, with Multi-ETag Option.

* Client-Proxy response revalidation, with the Group-ETag Option.

E.10. Version -01 to -02

* Clarified relation between security groups and application groups.
* Considered also FETCH for requests over IP multicast.
* More details on Observe re-registration.
* More details on Proxy intermediaries.
* More details on servers changing port number in the response.
* Usage of the Uri-Host Option to indicate an application group.
* Response suppression based on classes of error codes.

E.11. Version -00 to -01

* Clarifications on group memberships for the different group types.
* Simplified description of Token reusage, compared to the unicast case.
* More details on the rationale for response suppression.
* Clarifications of creation and management of security groups.
* Clients more knowledgeable than proxies about stopping receiving responses.
* Cancellation of group observations.
* Clarification on multicast scope to use.
* Both the group mode and pairwise mode of Group OSCORE are considered.
* Updated security considerations.
* Editorial improvements.

Acknowledgments

The authors sincerely thank Christian Amsüss, Carsten Bormann, Thomas Fossati, Rikard Höglund, Jaime Jiménez, John Preuß Mattsson, Jim Schaad, and Jon Shallow 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 H2020 projects SIFIS-Home (Grant agreement 952652) and ARCADIAN-IoT (Grant agreement 101020259).

Authors' Addresses

Esko Dijk
IoTconsultancy.nl
Utrecht
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
SE-16440 Stockholm Kista
Sweden
Email: marco.tiloca@ri.se

CoRE Working Group
Internet-Draft
Updates: 7595 (if approved)
Intended status: Standards Track
Expires: 23 October 2024

C. Bormann, Ed.
Universität Bremen TZI
H. Birkholz
Fraunhofer SIT

21 April 2024

Constrained Resource Identifiers
draft-ietf-core-href-15

Abstract

The Constrained Resource Identifier (CRI) is a complement to the Uniform Resource Identifier (URI) that represents the URI components in Concise Binary Object Representation (CBOR) instead of in a sequence of characters. This simplifies parsing, comparison, and reference resolution in environments with severe limitations on processing power, code size, and memory size.

// (This "cref" paragraph will be removed by the RFC editor:) The
// present revision 15 of this draft continues -14 by picking up
// more comments, such as moving to a CRI scheme number registration
// system based on unsigned numbers. This revision still contains
// open issues and is intended to serve as a snapshot.

About This Document

This note is to be removed before publishing as an RFC.

Status information for this document may be found at https://datatracker.ietf.org/doc/draft-ietf-core-href/.

Discussion of this document takes place on the Constrained RESTful Environments Working Group mailing list (mailto:core@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/core/. Subscribe at https://www.ietf.org/mailman/listinfo/core/.

Source for this draft and an issue tracker can be found at https://github.com/core-wg/href.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction 3
  1.1. Notational Conventions 4
2. Constraints 5
  2.1. Constraints not expressed by the data model 7
3. Creation and Normalization 8
4. Comparison 9
5. CRI References 9
  5.1. CBOR Representation 10
    5.1.1. scheme-name and scheme-id 12
    5.1.2. The discard Section 13
    5.1.3. Visualization 13
    5.1.4. Examples 14
    5.1.5. Specific Terminology 14
  5.2. Ingesting and encoding a CRI Reference 14
    5.2.1. Error handling and extensibility 15
  5.3. Reference Resolution 16
6. Relationship between CRIs, URIs, and IRIs 17
  6.1. Converting CRIs to URIs 18
7. Extending CRIs 20
  7.1. Extended CRI: Accommodating Percent Encoding (PET) . . . 21

8. CoAP Integration 23
8.1. Converting Between CoAP CRIs and Sets of CoAP Options . . 23
8.1.1. Decomposing a Request CRI into a set of CoAP Options 24
8.1.2. Composing a Request CRI from a Set of CoAP Options . 25
8.2. CoAP Options for Forward-Proxies 25
8.2.1. Proxy-CRI 26
8.2.2. Proxy-Scheme-Number 26
9. Implementation Status 27
10. Security Considerations 27
11. IANA Considerations 27
11.1. CRI Scheme Numbers Registry 28
11.1.1. Instructions for the Designated Expert 28
11.1.2. Structure of Entries 28
11.1.3. Initial Registrations 29
11.2. Update to "Uniform Resource Identifier (URI) Schemes" Registry 29
11.3. CBOR Diagnostic Notation Application-extension Identifiers Registry 29
11.4. CoAP Option Numbers Registry 30
12. References 30
12.1. Normative References 30
12.2. Informative References 32
Appendix A. Mapping Scheme Numbers to Scheme Names 33
Appendix B. The Small Print 50
Appendix C. CBOR Extended Diagnostic Notation (EDN): The "cri" Extension 51
C.1. cri: ABNF Definition of URI Representation of a CRI . . . 52
Appendix D. Change Log 54
Acknowledgements 57
Contributors 57
Authors' Addresses 58

1. Introduction

The Uniform Resource Identifier (URI) [STD66] and its most common usage, the URI reference, are the Internet standard for linking to resources in hypertext formats such as HTML [W3C.REC-html52-20171214] or the HTTP "Link" header field [RFC8288].

A URI reference is a sequence of characters chosen from the repertoire of US-ASCII characters. The individual components of a URI reference are delimited by a number of reserved characters, which necessitates the use of a character escape mechanism called "percent-encoding" when these reserved characters are used in a non-delimiting function. The resolution of URI references involves parsing a character sequence into its components, combining those components with the components of a base URI, merging path components, removing dot-segments, and recomposing the result back into a character sequence.

Overall, the proper handling of URI references is quite intricate. This can be a problem especially in constrained environments [RFC7228], where nodes often have severe code size and memory size limitations. As a result, many implementations in such environments support only an ad-hoc, informally-specified, bug-ridden, non-interoperable subset of half of [STD66].

This document defines the _Constrained Resource Identifier (CRI)_ by constraining URIs to a simplified subset and representing their components in Concise Binary Object Representation (CBOR) [STD94] instead of a sequence of characters. This allows typical operations on URI references such as parsing, comparison, and reference resolution (including all corner cases) to be implemented in a comparatively small amount of code.

As a result of simplification, however, CRIs are not capable of expressing all URIs permitted by the generic syntax of [STD66] (hence the "constrained" in "Constrained Resource Identifier"). The supported subset includes all URIs of the Constrained Application Protocol (CoAP) [RFC7252], most URIs of the Hypertext Transfer Protocol (HTTP) [STD97], Uniform Resource Names (URNs) [RFC8141], and other similar URIs. The exact constraints are defined in Section 2.

1.1. Notational Conventions

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 [BCP14] when, and only when, they appear in all capitals, as shown here.

In this specification, the term "byte" is used in its now customary sense as a synonym for "octet".

Terms defined in this document appear in _cursive_ where they are introduced (in the plaintext form of this document, they are rendered as the new term surrounded by underscores).

The general structure of data items is shown in the Concise Data Definition Language (CDDL) [RFC8610] including its control extensions [RFC9165]. Specific examples are notated in CBOR Extended Diagnostic Notation (EDN), as originally introduced in Section 8 of RFC 8949 [STD94] and extended in Appendix G of [RFC8610]. ([I-D.ietf-cbor-edn-literals] more rigorously defines and further extends EDN.)

2. Constraints

A Constrained Resource Identifier consists of the same five components as a URI: scheme, authority, path, query, and fragment. The components are subject to the following constraints:

C1. The scheme name can be any Unicode string (see Definition D80 in [Unicode]) that matches the syntax of a URI scheme (see Section 3.1 of RFC 3986 [STD66], which constrains scheme names to ASCII) and is lowercase (see Definition D139 in [Unicode]). The scheme is always present.

C2. An authority is always a host identified by an IP address or registered name, along with optional port information, and optionally preceded by user information.

Alternatively, the authority can be absent; the two cases for this defined in Section 3.3 of RFC 3986 [STD66] are modeled by two different values used in place of an absent authority:

* the path can be root-based (zero or more path components that are each started in the URI with "/", as when the authority is present), or

* the path can be rootless, which requires at least one path component.

(Note that, in Figure 1, no-authority is marked as a feature, as not all CRI implementations will support authority-less URIs.)

C3. A userinfo is a text string built out of unreserved characters (Section 2.3 of RFC 3986 [STD66]) or "sub-delims" (Section 2.2 of RFC 3986 [STD66]); any other character needs to be percent-encoded (Section 7.1). Note that this excludes the ":" character, which is commonly deprecated as a way to delimit a cleartext password in a userinfo.

C4. An IP address can be either an IPv4 address or an IPv6 address, optionally with a zone identifier [RFC6874]. Future versions of IP are not supported (it is likely that a binary mapping would be strongly desirable, and that cannot be designed ahead of time, so these versions need to be added as a future extension if needed).

C5. A registered name is a sequence of one or more _labels_, which, when joined with dots (".") in between them, result in a Unicode string that is lowercase and in Unicode Normalization Form C (NFC) (see Definition D120 in [Unicode]). (The syntax may be further restricted by the scheme. As per Section 3.2.2 of RFC 3986 [STD66], a registered name can be empty, for which case a scheme can define a default for the host.)

C6. A port is always an integer in the range from 0 to 65535. Ports outside this range, empty ports (port subcomponents with no digits, see Section 3.2.3 of RFC 3986 [STD66]), or ports with redundant leading zeros, are not supported.

C7. The port is omitted if and only if the port would be the same as the scheme's default port (provided the scheme is defining such a default port) or the scheme is not using ports.

C8. A path consists of zero or more path segments. Note that a path of just a single zero-length path segment is allowed this is considered equivalent to a path of zero path segments by HTTP and CoAP, but this equivalence does not hold for CRIs in general as they only perform normalization on the Syntax-Based Normalization level (Section 6.2.2 of RFC 3986 [STD66]), not on the scheme-specific Scheme-Based Normalization level (Section 6.2.3 of RFC 3986 [STD66]).

(A CRI implementation may want to offer scheme-cognizant interfaces, performing this scheme-specific normalization for schemes it knows. The interface could assert which schemes the implementation knows and provide pre-normalized CRIs. This can also relieve the application from removing a lone zero-length path segment before putting path segments into CoAP Options, i.e., from performing the check and jump in item 8 of Section 6.4 of [RFC7252]. See also SP1 in Appendix B.)

C9. A path segment can be any Unicode string that is in NFC, with the exception of the special "." and ".." complete path segments. Note that this includes the zero-length string.

If no authority is present in a CRI, the leading path segment cannot be empty. (See also SP1 in Appendix B.)

C10. A query always consists of one or more query parameters. A query parameter can be any Unicode string that is in NFC. It is often in the form of a "key=value" pair. When converting a CRI to a URI, query parameters are separated by an ampersand ("&") character. (This matches the structure and encoding of the target URI in CoAP requests.) Queries are optional; there is a difference between an absent query and a single query parameter that is the empty string.

C11. A fragment identifier can be any Unicode string that is in NFC. Fragment identifiers are optional; there is a difference between an absent fragment identifier and a fragment identifier that is the empty string.

C12. The syntax of registered names, path segments, query parameters, and fragment identifiers may be further restricted and sub-structured by the scheme. There is no support, however, for escaping sub-delimiters that are not intended to be used in a delimiting function.

C13. When converting a CRI to a URI, any character that is outside the allowed character range or is a delimiter in the URI syntax is percent-encoded. For CRIs, percent-encoding always uses the UTF-8 encoding form (see Definition D92 in [Unicode]) to convert the character to a sequence of bytes (that is then converted to a sequence of %HH triplets).

Examples for URIs at or beyond the boundaries of these constraints are in SP2 in Appendix B.

2.1. Constraints not expressed by the data model

There are syntactically valid CRIs and CRI references that cannot be converted into a URI or URI reference, respectively.

For CRI references, this is acceptable -- they can be resolved still and result in a valid CRI that can be converted back. Examples of this are:

* [0, ["p"]]: appends a slash and the path segment "p" to its base (and unsets the query and the fragment)

* [0, null, []]: leaves the path alone but unsets the query and the fragment

(Full) CRIs that do not correspond to a valid URI are not valid on their own, and cannot be used. Normatively they are characterized by the Section 6.1 process not producing a valid and syntax-normalized URI. For easier understanding, they are listed here:

* CRIs (and CRI references) containing a path component "." or "..".

  These would be removed by the remove_dot_segments algorithm of [STD66], and thus never produce a normalized URI after resolution.

  (In CRI references, the discard value is used to afford segment removal, and with "." being an unreserved character, expressing them as "%2e" and "%2e%2e" is not even viable, let alone practical).

* CRIs without authority whose path starts with two or more empty segments.

  When converted to URIs, these would violate the requirement that in absence of an authority, a URI's path cannot begin with two slash characters, and they would be indistinguishable from a URI with a shorter path and a present but empty authority component.

* CRIs without authority that are rootless and do not have a path component (e.g., ["a", true]), which would be indistinguishable from its root-based equivalent (["a"]) as both would have the URI a:.

3. Creation and Normalization

In general, resource identifiers are created on the initial creation of a resource with a certain resource identifier, or the initial exposition of a resource under a particular resource identifier.

A Constrained Resource Identifier SHOULD be created by the naming authority that governs the namespace of the resource identifier (see also [BCP190]). For example, for the resources of an HTTP origin server, that server is responsible for creating the CRIs for those resources.

The naming authority MUST ensure that any CRI created satisfies the constraints defined in Section 2. The creation of a CRI fails if the CRI cannot be validated to satisfy all of the constraints.

If a naming authority creates a CRI from user input, it MAY apply the following (and only the following) normalizations to get the CRI more likely to validate:

* map the scheme name to lowercase (C1);

* map the registered name to NFC (C5) and split it on embedded dots;

* elide the port if it is the default port for the scheme (C7);

* map path segments, query parameters and the fragment identifier to NFC form (C9, C10, C11).

Once a CRI has been created, it can be used and transferred without further normalization. All operations that operate on a CRI SHOULD rely on the assumption that the CRI is appropriately pre-normalized. (This does not contradict the requirement that when CRIs are transferred, recipients must operate on as-good-as untrusted input and fail gracefully in the face of malicious inputs.)

4. Comparison

One of the most common operations on CRIs is comparison: determining whether two CRIs are equivalent, without dereferencing the CRIs (i.e., using them to access their respective resource(s)).

Determination of equivalence or difference of CRIs is based on simple component-wise comparison. If two CRIs are identical component-by-component (using code-point-by-code-point comparison for components that are Unicode strings) then it is safe to conclude that they are equivalent.

This comparison mechanism is designed to minimize false negatives while strictly avoiding false positives. The constraints defined in Section 2 imply the most common forms of syntax- and scheme-based normalizations in URIs, but do not comprise protocol-based normalizations that require accessing the resources or detailed knowledge of the scheme's dereference algorithm. False negatives can be caused, for example, by CRIs that are not appropriately pre-normalized and by resource aliases.

When CRIs are compared to select (or avoid) a network action, such as retrieval of a representation, fragment components (if any) should be excluded from the comparison.

5. CRI References

The most common usage of a Constrained Resource Identifier is to embed it in resource representations, e.g., to express a hyperlink between the represented resource and the resource identified by the CRI.

This section defines the representation of CRIs in Concise Binary Object Representation (CBOR) [STD94]. When reduced representation size is desired, CRIs are often not represented directly. Instead, CRIs are indirectly referenced through _CRI references_. These take advantage of hierarchical locality and provide a very compact encoding. The CBOR representation of CRI references is specified in Section 5.1.

The only operation defined on a CRI reference is _reference resolution_: the act of transforming a CRI reference into a CRI. An application MUST implement this operation by applying the algorithm specified in Section 5.3 (or any algorithm that is functionally equivalent to it).

The reverse operation of transforming a CRI into a CRI reference is not specified in detail in this document; implementations are free to use any algorithm as long as reference resolution of the resulting CRI reference yields the original CRI. Notably, a CRI reference is not required to satisfy all of the constraints of a CRI; the only requirement on a CRI reference is that reference resolution MUST yield the original CRI.

When testing for equivalence or difference, applications SHOULD NOT directly compare CRI references; the references should be resolved to their respective CRI before comparison.

5.1. CBOR Representation

// RFC Ed.: throughout this section, please replace RFC-XXXX with the
// RFC number of this specification and remove this note.

A CRI or CRI reference is encoded as a CBOR array (Major type 4 in Section 3.1 of RFC 8949 [STD94]), with the structure described in CDDL as follows:

```
; not expressed in this CDDL spec: trailing nulls to be left off

RFC-XXXX-Definitions = [CRI, CRI-Reference]

CRI = [
  scheme,
  authority / no-authority,
  path / null,
  query / null,
  fragment / null
]


CRI-Reference = [
  ((scheme / null, authority / no-authority)
   // discard),                 ; relative reference
  path / null,
  query / [] / null,            ; [] is explicit unset
  fragment / null
]

scheme      = scheme-name / scheme-id
scheme-name = text .regexp "[a-z][a-z0-9+.-]*"
scheme-id   = nint              ; -1 - scheme-number

no-authority = NOAUTH-ROOTBASED / NOAUTH-ROOTLESS
NOAUTH-ROOTBASED = null .feature "no-authority"
NOAUTH-ROOTLESS = true .feature "no-authority"

authority   = [?userinfo, host, ?port]
userinfo    = (false, text .feature "userinfo")
host        = (host-ip // host-name)
host-name   = (*text) ; lowercase, NFC labels
host-ip     = (bytes .size 4 //
               (bytes .size 16, ?zone-id))
zone-id     = text
port        = 0..65535

discard     = DISCARD-ALL / 0..127
DISCARD-ALL = true
path        = [*text]
query       = [+text]
fragment    = text
```

Figure 1: CDDL for CRI CBOR representation

The rules scheme, authority, path, query, fragment correspond to the (sub-)components of a CRI, as described in Section 2, with the addition of the discard section.

This CDDL specification is simplified for exposition and needs to be augmented by the following rules for interchange of CRIs and CRI references:

* Trailing null values MUST be removed,

* two leading null values (scheme and authority both not given) MUST be represented by using the discard alternative instead, and

* an empty path in a CRI MUST be represented as the empty array [] (note that for CRI-Reference there is a difference between empty and absent paths, represented by [] and null, respectively),

* an entirely empty outer array is not a valid CRI (but a valid CRI reference, as per Section 5.2 equivalent to [0], which essentially copies the base CRI).

For interchange as separate encoded data items, CRIs MUST NOT use indefinite length encoding (see Section 3.2 of RFC 8949 [STD94]); this requirement is relaxed for specifications that embed CRIs into an encompassing CBOR representation that does provide for indefinite length encoding.

5.1.1. scheme-name and scheme-id

In the scheme section, a CRI scheme can be given by its scheme-name (a text string giving the scheme name as in URIs' scheme section, mapped to lower case), or as a negative integer scheme-id derived from the _scheme number_. Scheme numbers are unsigned integers that are mapped to and from URI scheme names by the "CRI Scheme Numbers" registry (Section 11.1). The relationship of a scheme number to its scheme-id is as follows:

```
scheme-id = -1 - scheme-number
scheme-number = -1 - scheme-id
```

For example, the scheme-name coap has the (unsigned integer) scheme-number 0 which is represented in a (negative integer) scheme-id -1.

5.1.2. The discard Section

The discard section can be used in a CRI reference when neither a scheme nor an authority is present. It then expresses the operations performed on a base CRI by CRI references that are equivalent to URI references with relative paths and path prefixes such as "/", "./", "../", "../../", etc.
"." and ".." are not available in CRIs and are therefore expressed using discard after a normalization step, as is the presence or absence of a leading "/".

E.g., a simple URI reference "foo" specifies to remove one leading segment from the base URI's path, which is represented in the equivalent CRI reference discard section as the value 1; similarly "../foo" removes two leading segments, represented as 2; and "/foo" removes all segments, represented in the discard section as the value true. The exact semantics of the section values are defined by Section 5.3.

Most URI references that Section 4.2 of RFC 3986 [STD66] calls "relative references" (i.e., references that need to undergo a resolution process to obtain a URI) correspond to the CRI reference form that starts with discard. The exception are relative references with an authority (called a "network-path reference" in Section 4.2 of RFC 3986 [STD66]), which discard the entire path of the base CRI. These CRI references never carry a discard section: the value of discard defaults to true.

5.1.3. Visualization

The structure of a CRI reference is visualized using the somewhat limited means of a railroad diagram:

```
cri-reference:
     scheme  authority  local-part

         discard

local-part:
        >

                  >

                              >

     path  query  fragment
```

This visualization does not go into the details of the elements.

5.1.4. Examples

```
[-1,             / scheme-id -- equivalent to "coap" /
 [h'C6336401',   / host /
  61616],        / port /
 [".well-known", / path /
  "core"]
]

[true,                  / discard /
 [".well-known",        / path /
  "core"],
 ["rt=temperature-c"]]  / query /

[-6,                / scheme-id -- equivalent to "did" /
 true,              / authority = NOAUTH-ROOTLESS /
 ["web:alice:bob"]  / path /
]
```

5.1.5. Specific Terminology

A CRI reference is considered _well-formed_ if it matches the structure as expressed in Figure 1 in CDDL, with the additional requirement that trailing null values are removed from the array.

A CRI reference is considered _absolute_ if it is well-formed and the sequence of sections starts with a non-null scheme.

A CRI reference is considered _relative_ if it is well-formed and the sequence of sections is empty or starts with a section other than those that would constitute a scheme.

5.2. Ingesting and encoding a CRI Reference

From an abstract point of view, a CRI Reference is a data structure with six sections:

scheme, authority, discard, path, query, fragment

Each of these sections can be unset ("null"), except for discard, which is always an unsigned integer or true. If scheme and/or authority are non-null, discard must be true.

When ingesting a CRI Reference that is in the transfer form, those sections are filled in from the transfer form (unset sections are filled with null), and the following steps are performed:

* If the array is entirely empty, replace it with [0].

* If discard is present in the transfer form (i.e., the outer array starts with true or an unsigned integer), set scheme and authority to null.

* If scheme and/or authority are present in the transfer form (i.e., the outer array starts with null, a text string, or a negative integer), set discard to true.

Upon encoding the abstract form into the transfer form, the inverse processing is performed: If scheme and/or authority are not null, the discard value is not transferred (it must be true in this case). If they are both null, they are both left out and only discard is transferred. Trailing null values are removed from the array. As a special case, an empty array is sent in place for a remaining [0] (URI reference "").

5.2.1. Error handling and extensibility

It is recommended that specifications that describe the use of CRIs in CBOR-based protocols use the error handling mechanisms outlined in this section. Implementations of this document MUST adhere to these rules unless a containing document overrides them.

When encountering a CRI that is well-formed in terms of CBOR, but that

* is not well-formed as a CRI,

* does not meet the other requirements on CRIs that are not covered by the term "well-formed", or

* uses features not supported by the implementation,

the CRI is treated as "unprocessable".

When encountering an unprocessable CRI, the processor skips the entire CRI top-level array, including any CBOR items contained in there, and continues processing the CBOR items surrounding the unprocessable CRI. (Note: this skipping can be implemented in bounded memory for CRIs that do not use indefinite length encoding, as mandated in Section 5.1.)

The unprocessable CRI is treated as an opaque identifier that is distinct from all processable CRIs, and distinct from all unprocessable CRIs with different CBOR representations. It is up to implementation whether unprocessable CRIs with identical representations are treated as identical to each other or not. Unprocessable CRIs cannot be dereferenced, and it is an error to query any of their components.

This mechanism ensures that CRI extensions (using originally defined features or later extensions) can be used without extending the compatibility hazard to the containing document. For example, if a collection of possible interaction targets contains several CRIs, some of which use the "no-authority" feature, an application consuming that collection that does not support that feature can still offer the supported interaction targets.

The duty of checking validity is with the recipients that rely on this validity. An intermediary that does not use the detailed information in a CRI (or merely performs reference resolution) MAY pass on a CRI/CRI reference without having fully checked it, relying on the producer having generated a valid CRI/CRI reference. This is true for both basic CRIs (e.g., checking for valid UTF-8) and for extensions (e.g., checking both for valid UTF-8 and the minimal use of PET elements in extended-cris as per Section 7.1).

5.3. Reference Resolution

The term "relative" implies that a "base CRI" exists against which the relative reference is applied. Aside from fragment-only references, relative references are only usable when a base CRI is known.

The following steps define the process of resolving any well-formed CRI reference against a base CRI so that the result is a CRI in the form of an absolute CRI reference:

1. Establish the base CRI of the CRI reference and express it in the form of an abstract absolute CRI reference.

2. Initialize a buffer with the sections from the base CRI.

3. If the value of discard is true in the CRI reference (which is implicitly the case when scheme and/or authority are present in the reference), replace the path in the buffer with the empty array, unset query and fragment, and set a true authority to null. If the value of discard is an unsigned integer, remove as many elements from the end of the path array; if it is non-zero, unset query and fragment.

Set discard to true in the buffer.

4. If the path section is set in the CRI reference, append all elements from the path array to the array in the path section in the buffer; unset query and fragment.

5. Apart from the path and discard, copy all non-null sections from the CRI reference to the buffer in sequence; unset query in the buffer if query is the empty array [] in the CRI reference; unset fragment in the buffer if query is non-null in the CRI reference.

6. Return the sections in the buffer as the resolved CRI.

6. Relationship between CRIs, URIs, and IRIs

CRIs are meant to replace both Uniform Resource Identifiers (URIs) [STD66] and Internationalized Resource Identifiers (IRIs) [RFC3987] in constrained environments [RFC7228]. Applications in these environments may never need to use URIs and IRIs directly, especially when the resource identifier is used simply for identification purposes or when the CRI can be directly converted into a CoAP request.

However, it may be necessary in other environments to determine the associated URI or IRI of a CRI, and vice versa. Applications can perform these conversions as follows:

CRI to URI
: A CRI is converted to a URI as specified in Section 6.1.

URI to CRI
: The method of converting a URI to a CRI is unspecified; implementations are free to use any algorithm as long as converting the resulting CRI back to a URI yields an equivalent URI.

  Note that CRIs are defined to enable implementing conversions from or to URIs analogously to processing URIs into CoAP Options and back, with the exception that item 8 of Section 6.4 of [RFC7252] and item 7 of Section 6.5 of [RFC7252] do not apply to CRI processing. See SP1 in Appendix B for more details.

CRI to IRI
: A CRI can be converted to an IRI by first converting it to a URI as specified in Section 6.1, and then converting the URI to an IRI as described in Section 3.2 of [RFC3987].

IRI to CRI

An IRI can be converted to a CRI by first converting it to a URI as described in Section 3.1 of [RFC3987], and then converting the URI to a CRI as described above.

Everything in this section also applies to CRI references, URI references, and IRI references.

6.1. Converting CRIs to URIs

Applications MUST convert a CRI reference to a URI reference by determining the components of the URI reference according to the following steps and then recomposing the components to a URI reference string as specified in Section 5.3 of RFC 3986 [STD66].

scheme
: If the CRI reference contains a scheme section, the scheme component of the URI reference consists of the value of that section, if text (scheme-name); or, if a negative integer is given (scheme-id), the lower case scheme name corresponding to the scheme-id as per Section 5.1.1. Otherwise, the scheme component is unset.

authority
: If the CRI reference contains a host-name or host-ip item, the authority component of the URI reference consists of a host subcomponent, optionally followed by a colon (":") character and a port subcomponent, optionally preceded by a userinfo subcomponent. Otherwise, the authority component is unset.

  The host subcomponent consists of the value of the host-name or host-ip item.

  The userinfo subcomponent, if present, is turned into a single string by appending a "@". Otherwise, both the subcomponent and the "@" sign are omitted. Any character in the value of the userinfo elements that is not in the set of unreserved characters (Section 2.3 of RFC 3986 [STD66]) or "sub-delims" (Section 2.2 of RFC 3986 [STD66]) MUST be percent-encoded.

  The host-name is turned into a single string by joining the elements separated by dots (."). Any character in the elements of a host-name item that is not in the set of unreserved characters (Section 2.3 of RFC 3986 [STD66]) or "sub-delims" (Section 2.2 of RFC 3986 [STD66]) MUST be percent-encoded. If there are dots (".") in such elements, the conversion fails (percent-encoding is not able to represent such elements, as normalization would turn the percent-encoding back to the unreserved character that a dot is.)

The value of a host-ip item MUST be represented as a string that matches the "IPv4address" or "IP-literal" rule (Section 3.2.2 of RFC 3986 [STD66]). Any zone-id is appended to the string; the details for how this is done are currently in flux in the URI specification: Section 2 of [RFC6874] uses percent-encoding and a separator of "%25", while proposals for a future superseding zone-id specification document (such as [I-D.carpenter-6man-rfc6874bis]) are being prepared; this also leads to a modified "IP-literal" rule as specified in these documents.

If the CRI reference contains a port item, the port subcomponent consists of the value of that item in decimal notation. Otherwise, the colon (":") character and the port subcomponent are both omitted.

path

If the CRI reference contains a discard item of value true, the path component is considered _rooted_. If it contains a discard item of value 0 and the path item is present, the conversion fails. If it contains a positive discard item, the path component is considered _unrooted_ and prefixed by as many "../" components as the discard value minus one indicates. If the discard value is 1 and the first element of the path contains a :, the path component is prefixed by "./" (this avoids the first element to appear as supplying a URI scheme; compare path-noscheme in Section 4.2 of RFC 3986 [STD66]).

If the discard item is not present and the CRI reference contains an authority that is true, the path component of the URI reference is considered unrooted. Otherwise, the path component is considered rooted.

If the CRI reference contains one or more path items, the path component is constructed by concatenating the sequence of representations of these items. These representations generally contain a leading slash ("/") character and the value of each item, processed as discussed below. The leading slash character is omitted for the first path item only if the path component is considered "unrooted".

Any character in the value of a path item that is not in the set of unreserved characters or "sub-delims" or a colon (":") or commercial at ("@") character MUST be percent-encoded.

If the authority component is present (not null or true) and the path component does not match the "path-abempty" rule (Section 3.3 of RFC 3986 [STD66]), the conversion fails.

If the authority component is not present, but the scheme component is, and the path component does not match the "path-absolute", "path-rootless" (authority == true) or "path-empty" rule (Section 3.3 of RFC 3986 [STD66]), the conversion fails.

If neither the authority component nor the scheme component are present, and the path component does not match the "path-absolute", "path-noscheme" or "path-empty" rule (Section 3.3 of RFC 3986 [STD66]), the conversion fails.

query
: If the CRI reference contains one or more query items, the query component of the URI reference consists of the value of each item, separated by an ampersand ("&") character. Otherwise, the query component is unset.

  Any character in the value of a query item that is not in the set of unreserved characters or "sub-delims" or a colon (":"), commercial at ("@"), slash ("/") or question mark ("?") character MUST be percent-encoded. Additionally, any ampersand character ("&") in the item value MUST be percent-encoded.

fragment
: If the CRI reference contains a fragment item, the fragment component of the URI reference consists of the value of that item. Otherwise, the fragment component is unset.

  Any character in the value of a fragment item that is not in the set of unreserved characters or "sub-delims" or a colon (":"), commercial at ("@"), slash ("/") or question mark ("?") character MUST be percent-encoded.

7. Extending CRIs

CRIs have been designed to relieve implementations operating on CRIs from string scanning, which both helps constrained implementations and implementations that need to achieve high throughput.

The CRI structure described up to this point is termed the _Basic CRI_. It should be sufficient for all applications that use the CoAP protocol, as well as most other protocols employing URIs.

However, Basic CRIs have one limitation: They do not support URI components that _require_ percent-encoding (Section 2.1 of RFC 3986 [STD66]) to represent them in the URI syntax, except where that percent-encoding is used to escape the main delimiter in use.

E.g., the URI

https://alice/3%2f4-inch

is represented by the basic CRI

[-4, ["alice"], ["3/4-inch"]]

However, percent-encoding that is used at the application level is not supported by basic CRIs:

did:web:alice:7%3A1-balun

Extended forms of CRIs may be defined to enable these applications. They will generally extend the potential values of text components of URIs, such as userinfo, hostnames, paths, queries, and fragments.

One such extended form is described in the following Section 7.1. Consumers of CRIs will generally notice when an extended form is in use, by finding structures that do not match the CDDL rules given in Figure 1. Future definitions of extended forms need to strive to be distinguishable in their structures from the extended form presented here as well as other future forms.

Extensions to CRIs MUST NOT allow indefinite length items. This provision ensures that recipients of CRIs can deal with unprocessable CRIs as described in Section 5.2.1.

7.1. Extended CRI: Accommodating Percent Encoding (PET)

This section presents a method to represent percent-encoded segments of userinfo, hostnames, paths, and queries, as well as fragments.

The four CDDL rules

```
userinfo    = (false, text .feature "userinfo")
host-name   = (*text)
path        = [*text]
query       = [+text]
fragment    = text
```

are replaced with

```
userinfo    = (false, text-or-pet .feature "userinfo")
host-name   = (*text-or-pet)
path        = [*text-or-pet]
query       = [+text-or-pet]
fragment    = text-or-pet

text-or-pet = text /
    text-pet-sequence .feature "extended-cri"

; text1 and pet1 alternating, at least one pet1:
text-pet-sequence = [?text1, ((+(pet1, text1), ?pet1) // pet1)]
; pet is percent-encoded bytes
pet1 = bytes .ne ''
text1 = text .ne ""
```

That is, for each of the host-name, path, and query segments, and for the userinfo and fragment components, an alternate representation is provided besides a simple text string: a non-empty array of alternating non-blank text and byte strings, the text strings of which stand for non-percent-encoded text, while the byte strings retain the special semantics of percent-encoded text without actually being percent-encoded.

The above DID URI can now be represented as:

```
[-6, true, [["web:alice:7", ':', "1-balun"]]]
```

(Note that, in CBOR diagnostic notation, single quotes delimit literals for byte strings, double quotes for text strings.)

To yield a valid extended-cri, the use of byte strings MUST be minimal. Both the following examples are therefore not valid:

```
[-6, true, [["web:alice:", '7:', "1-balun"]]]
[-6, true, [["web:alice:7", ':1', "-balun"]]]
```

An algorithm for constructing a valid text-pet-sequence might repeatedly examine the byte sequences in each byte string; if such a sequence stands for an unreserved ASCII character, or constitutes a valid UTF-8 character U+0080, move this character over into a text string by appending it to the end of the preceding text string, prepending it to the start of the following text string, or splitting the byte string and inserting a new text string with this character, all while preserving the order of the bytes. (Note that the properties of UTF-8 make this a simple linear process.)

> Unlike the text elements of a path or a query, which through CoAP's heritage are designed to be processable element by element, a text-pet-sequence does not usually produce a semantically meaningful division into array elements. This consequence of the flexibility in delimiters offered in URIs is demonstrated by this example, which structurally singles out the one ':' that is _not_ a delimiter at the application level. Applications designed for using CRIs will generally avoid using the extended-cri feature. Applications using existing URI structures that require text-pet-sequence elements for their representation typically need to process them byte by byte.

8. CoAP Integration

This section discusses ways in which CRIs can be used in the context of the CoAP protocol [RFC7252].

8.1. Converting Between CoAP CRIs and Sets of CoAP Options

This section provides an analogue to Sections 6.4 and 6.5 of [RFC7252]: Computing a set of CoAP options from a request CRI (Section 8.1.1) and computing a request CRI from a set of COAP options (Section 8.1.2).

This section makes use of the mapping between CRI scheme numbers and URI scheme names shown in Table 1:

CRI scheme number	URI scheme name
0	coap
1	coaps
6	coap+tcp
7	coaps+tcp
8	coap+ws
9	coaps+ws

Table 1: Mapping CRI scheme numbers and URI scheme names

8.1.1. Decomposing a Request CRI into a set of CoAP Options

The steps to parse a request's options from a CRI »cri« are as follows. These steps either result in zero or more of the Uri-Host, Uri-Port, Uri-Path, and Uri-Query Options being included in the request or they fail.

Where the following speaks of deriving a text-string for a CoAP Option value from a data item in the CRI, the presence of any text-pet-sequence subitem (Section 7.1) in this item fails this algorithm.

1. If »cri« is not an absolute CRI reference, then fail this algorithm.

2. Translate the scheme-id into a URI scheme name as per Section 5.1.1 and Table 1; if a scheme-id that corresponds to a scheme number not in this list is being used, or if a scheme-name is being used, fail this algorithm. Remember the specific variant of CoAP to be used based on this URI scheme name.

3. If »cri« has a fragment component, then fail this algorithm.

4. If the host component of »cri« is a host-name, include a Uri-Host Option and let that option's value be the text string value of the host-name.

   If the host component of »cri« is a host-ip, check whether the IP address given represents the request's destination IP address (and, if present, zone-id). Only if it does not, include a Uri-Host Option, and let that option's value be the text value of the URI representation of the IP address, as derived in Section 6.1, Paragraph 2.4.5.

5. If »cri« has a port component, then let »port« be that component's unsigned integer value; otherwise, let »port« be the default port number for the scheme.

6. If »port« does not equal the request's destination UDP port, include a Uri-Port Option and let that option's value be »port«.

7. If the value of the path component of »cri« is empty or consists of a single empty string, then move to the next step.

   Otherwise, for each element in the »path« component, include a Uri-Path Option and let that option's value be the text string value of that element.

8. If »cri« has a query component, then, for each element in the query component, include a Uri-Query Option and let that option's value be the be the text string value of that element.

8.1.2. Composing a Request CRI from a Set of CoAP Options

The steps to construct a CRI from a request's options are as follows. These steps either result in a CRI or they fail.

1. Based on the variant of CoAP used in the request, choose a scheme-id as per Section 5.1.1 and table Table 1. Use that as the first value in the resulting CRI array.

2. If the request includes a Uri-Host Option, insert an authority with its value determined as follows: If the value of the Uri-Host Option is a reg-name, include this as the host-name. If the value is an IP-literal or IPv4address, extract any zone-id, and represent the IP address as a byte string of the correct length in host-ip, followed by any zone-id extracted if present. If the value is none of the three, fail this algorithm.

   If the request does not include a Uri-Host Option, insert an authority with host-ip being the byte string that represents the request's destination IP address and, if one is present in the request's destination, add a zone-id.

3. If the request includes a Uri-Port Option, let »port« be that option's value. Otherwise, let »port« be the request's destination UDP port. If »port« is not the default port for the scheme, then insert the integer value of »port« as the value of port in the authority. Otherwise, elide the port.

4. Insert a path component that contains an array built from the text string values of the Uri-Path Options in the request, or an empty array if no such options are present.

5. Insert a query component that contains an array built from the text string values of the Uri-Query Options in the request, or an empty array if no such options are present.

8.2. CoAP Options for Forward-Proxies

Apart from the above procedures to convert CoAP CRIs to and from sets of CoAP Options, two additional CoAP Options are defined in Section 5.10.2 of [RFC7252] that support requests to forward-proxies:

* Proxy-Uri, and

* its more lightweight variant, Proxy-Scheme

This section defines analogues of these that employ CRIs and the URI Scheme numbering provided by the present specification.

8.2.1. Proxy-CRI

No.	C	U	N	R	Name	Format	Length	Default
TBD235	x	x	-		Proxy-Cri	opaque	1-1023	(none)

Table 2: Proxy-Cri CoAP Option

The Proxy-CRI Option carries an encoded CBOR data item that represents an absolute CRI reference. It is used analogously to Proxy-Uri as defined in Section 5.10.2 of [RFC7252]. The Proxy-Cri Option MUST take precedence over any of the Uri-Host, Uri-Port, Uri-Path or Uri-Query options, as well as over any Proxy-Uri Option (each of which MUST NOT be included in a request containing the Proxy-Cri Option).

8.2.2. Proxy-Scheme-Number

No.	C	U	N	R	Name	Format	Length	Default
TBD239	x	x	-		Proxy-Scheme-Number	uint	0-3	(none)

Table 3: Proxy-Scheme-Number CoAP Option

The Proxy-Scheme-Number Option carries a CRI Scheme Number represented as a CoAP unsigned integer. It is used analogously to Proxy-Scheme as defined in Section 5.10.2 of [RFC7252].

As per Section 3.2 of [RFC7252], CoAP Options are only defined as one of empty, (text) string, opaque (byte string), or uint (unsigned integer). The Option therefore carries an unsigned integer that represents the CRI scheme-number (which relates to a CRI scheme-id as defined in Section 5.1.1). For instance, the scheme name "coap" has the scheme-number 0 and is represented as an unsigned integer by a zero-length CoAP Option value.

// TO DO: Discuss the need for a location-scheme-numeric option?

9. Implementation Status

(Boilerplate as per Section 2.1 of [RFC7942]:)

This section records the status of known implementations of the protocol defined by this specification at the time of posting of this Internet-Draft, and is based on a proposal described in [RFC7942]. The description of implementations in this section is intended to assist the IETF in its decision processes in progressing drafts to RFCs. Please note that the listing of any individual implementation here does not imply endorsement by the IETF. Furthermore, no effort has been spent to verify the information presented here that was supplied by IETF contributors. This is not intended as, and must not be construed to be, a catalog of available implementations or their features. Readers are advised to note that other implementations may exist.

According to [RFC7942], "this will allow reviewers and working groups to assign due consideration to documents that have the benefit of running code, which may serve as evidence of valuable experimentation and feedback that have made the implemented protocols more mature. It is up to the individual working groups to use this information as they see fit".

With the exception of the authority=true fix, host-names split into labels, and Section 7.1, CRIs are implemented in https://gitlab.com/chrysn/micrurus. A golang implementation of version -10 of this document is found at: https://github.com/thomas-fossati/href

10. Security Considerations

Parsers of CRI references must operate on input that is assumed to be untrusted. This means that parsers MUST fail gracefully in the face of malicious inputs. Additionally, parsers MUST be prepared to deal with resource exhaustion (e.g., resulting from the allocation of big data items) or exhaustion of the call stack (stack overflow). See Section 10 of RFC 8949 [STD94] for additional security considerations relating to CBOR.

The security considerations discussed in Section 7 of RFC 3986 [STD66] and Section 8 of [RFC3987] for URIs and IRIs also apply to CRIs.

11. IANA Considerations

11.1. CRI Scheme Numbers Registry

This specification defines a new "CRI Scheme Numbers" sub-registry in the "CoRE Parameters" registry [IANA.core-parameters], with the policy "Expert Review" (Section 4.5 of RFC 8126 [BCP26]). The objective is to have CRI scheme number values registered for all registered URI schemes (Uniform Resource Identifier (URI) Schemes registry), as well as exceptionally for certain text strings that the Designated Expert considers widely used in constrained applications in place of URI scheme names.

11.1.1. Instructions for the Designated Expert

The expert is instructed to be frugal in the allocation of CRI scheme number values whose scheme-id values (Section 5.1.1) have short representations (1+0 and 1+1 encoding), keeping them in reserve for applications that are likely to enjoy wide use and can make good use of their shortness.

When the expert notices that a registration has been made in the Uniform Resource Identifier (URI) Schemes registry (see also Section 11.2), the expert is requested to initiate a parallel registration in the CRI Scheme Numbers registry. CRI scheme number values in the range between 1000 and 20000 (inclusive) should be assigned unless a shorter representation in CRIs appears desirable.

The expert exceptionally also may make such a registration for text strings that have not been registered in the Uniform Resource Identifier (URI) Schemes registry if and only if the expert considers them to be in wide use in place of URI scheme names in constrained applications. (Note that the initial registrations in Table 6 in Appendix A already include such registrations for the text strings "mqtt" and "mqtts".)

A registration in the CRI Scheme Numbers registry does not imply that a URI scheme under this name exists or has been registered in the Uniform Resource Identifier (URI) Schemes registry -- it essentially is only providing an integer identifier for an otherwise uninterpreted text string.

Any questions or issues that might interest a wider audience might be raised by the expert on the core-parameters@ietf.org mailing list for a time-limited discussion.

11.1.2. Structure of Entries

Each entry in the registry must include:

CRI scheme number:
An unsigned integer unique in this registry

URI scheme name:
a text string that would be acceptable for registration as a URI Scheme Name in the Uniform Resource Identifier (URI) Schemes registry

Reference:
a reference to a document, if available, or the registrant

11.1.3. Initial Registrations

The initial registrations for the CRI Scheme Numbers registry are provided in Table 6 in Appendix A.

11.2. Update to "Uniform Resource Identifier (URI) Schemes" Registry

RFC 7595 [BCP35] is updated to add the following note in the "Uniform Resource Identifier (URI) Schemes" Registry [IANA.uri-schemes]:

> The CRI Scheme Numbers Registry registers numeric identifiers for what essentially are URI Scheme names. Registrants for the Uniform Resource Identifier (URI) Schemes Registry are requested to make a parallel registration in the CRI Scheme Numbers registry. The number for this registration will be assigned by the Designated Expert for that registry.

11.3. CBOR Diagnostic Notation Application-extension Identifiers Registry

In the "Application-Extension Identifiers" registry in the "CBOR Diagnostic Notation" registry group [IANA.cbor-diagnostic-notation], IANA is requested to register the application-extension identifier cri as described in Table 4 and defined in Appendix C.

Application-extension Identifier	Description	Change Controller	Reference
cri	Constrained Resource Identifier	IETF	RFC-XXXX, Appendix C

Table 4: CBOR Extended Diagnostic Notation (EDN) Application-extension Identifier for CRI

// RFC Ed.: throughout this section, please replace RFC-XXXX with the
// RFC number of this specification and remove this note.

11.4. CoAP Option Numbers Registry

In the "CoAP Option Numbers" registry in the "CoRE Parameters" registry group [IANA.core-parameters], IANA is requested to register the CoAP Option Numbers as described in Table 5 and defined in Section 8.2.

No.	Name	Reference
TBD235	Proxy-Cri	RFC-XXXX
TBD239	Proxy-Scheme-Number	RFC-XXXX

Table 5: New CoAP Option Numbers

// RFC Ed.: throughout this section, please replace RFC-XXXX with the
// RFC number of this specification and remove this note.

12. References

12.1. Normative References

[BCP14] Best Current Practice 14, <https://www.rfc-editor.org/info/bcp14>. At the time of writing, this BCP comprises the following:

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>.

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>.

[BCP26] Best Current Practice 26, <https://www.rfc-editor.org/info/bcp26>. At the time of writing, this BCP comprises the following:

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>.

[BCP35] Best Current Practice 35, <https://www.rfc-editor.org/info/bcp35>. At the time of writing, this BCP comprises the following:

Thaler, D., Ed., Hansen, T., and T. Hardie, "Guidelines and Registration Procedures for URI Schemes", BCP 35, RFC 7595, DOI 10.17487/RFC7595, June 2015, <https://www.rfc-editor.org/info/rfc7595>.

[IANA.core-parameters] IANA, "Constrained RESTful Environments (CoRE) Parameters", <http://www.iana.org/assignments/core-parameters>.

[IANA.uri-schemes] IANA, "Uniform Resource Identifier (URI) Schemes", <http://www.iana.org/assignments/uri-schemes>.

[RFC3987] Duerst, M. and M. Suignard, "Internationalized Resource Identifiers (IRIs)", RFC 3987, DOI 10.17487/RFC3987, January 2005, <https://www.rfc-editor.org/rfc/rfc3987>.

[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/rfc/rfc6874>.

[RFC8610] Birkholz, H., Vigano, C., and C. Bormann, "Concise Data Definition Language (CDDL): A Notational Convention to Express Concise Binary Object Representation (CBOR) and JSON Data Structures", RFC 8610, DOI 10.17487/RFC8610, June 2019, <https://www.rfc-editor.org/rfc/rfc8610>.

[RFC9165] Bormann, C., "Additional Control Operators for the Concise Data Definition Language (CDDL)", RFC 9165, DOI 10.17487/RFC9165, December 2021, <https://www.rfc-editor.org/rfc/rfc9165>.

[STD66] Internet Standard 66, <https://www.rfc-editor.org/info/std66>. At the time of writing, this STD comprises the following:

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>.

[STD94] Internet Standard 94, <https://www.rfc-editor.org/info/std94>. At the time of writing, this STD comprises the following:

Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", STD 94, RFC 8949, DOI 10.17487/RFC8949, December 2020, <https://www.rfc-editor.org/info/rfc8949>.

[Unicode] The Unicode Consortium, "The Unicode Standard, Version 13.0.0", ISBN 978-1-936213-26-9, March 2020, <https://www.unicode.org/versions/Unicode13.0.0/>.

12.2. Informative References

[BCP190] Best Current Practice 190, <https://www.rfc-editor.org/info/bcp190>. At the time of writing, this BCP comprises the following:

Nottingham, M., "URI Design and Ownership", BCP 190, RFC 8820, DOI 10.17487/RFC8820, June 2020, <https://www.rfc-editor.org/info/rfc8820>.

[I-D.carpenter-6man-rfc6874bis]
Carpenter, B. E., Cheshire, S., and R. M. Hinden, "Representing IPv6 Zone Identifiers in Address Literals and Uniform Resource Identifiers", Work in Progress, Internet-Draft, draft-carpenter-6man-rfc6874bis-03, 8 February 2022, <https://datatracker.ietf.org/doc/html/draft-carpenter-6man-rfc6874bis-03>.

[I-D.ietf-cbor-edn-literals]
Bormann, C., "CBOR Extended Diagnostic Notation (EDN): Application-Oriented Literals, ABNF, and Media Type", Work in Progress, Internet-Draft, draft-ietf-cbor-edn-literals-08, 1 February 2024, <https://datatracker.ietf.org/doc/html/draft-ietf-cbor-edn-literals-08>.

[RFC4180] Shafranovich, Y., "Common Format and MIME Type for Comma-Separated Values (CSV) Files", RFC 4180, DOI 10.17487/RFC4180, October 2005, <https://www.rfc-editor.org/rfc/rfc4180>.

[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/rfc/rfc7228>.

[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/rfc/rfc7252>.

[RFC7942] Sheffer, Y. and A. Farrel, "Improving Awareness of Running Code: The Implementation Status Section", BCP 205, RFC 7942, DOI 10.17487/RFC7942, July 2016, <https://www.rfc-editor.org/rfc/rfc7942>.

[RFC8141] Saint-Andre, P. and J. Klensin, "Uniform Resource Names (URNs)", RFC 8141, DOI 10.17487/RFC8141, April 2017, <https://www.rfc-editor.org/rfc/rfc8141>.

[RFC8288] Nottingham, M., "Web Linking", RFC 8288, DOI 10.17487/RFC8288, October 2017, <https://www.rfc-editor.org/rfc/rfc8288>.

[STD97] Internet Standard 97, <https://www.rfc-editor.org/info/std97>. At the time of writing, this STD comprises the following:

Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP Semantics", STD 97, RFC 9110, DOI 10.17487/RFC9110, June 2022, <https://www.rfc-editor.org/info/rfc9110>.

[W3C.REC-html52-20171214] Danilo, A., Ed., Eicholz, A., Ed., Moon, S., Ed., Faulkner, S., Ed., and T. Leithead, Ed., "HTML 5.2", W3C REC REC-html52-20171214, W3C REC-html52-20171214, 14 December 2017, <https://www.w3.org/TR/2017/REC-html52-20171214/>.

Appendix A. Mapping Scheme Numbers to Scheme Names

// RFC Ed.: throughout this section, please replace RFC-XXXX with the
// RFC number of this specification and remove this note.

Table 6 defines the initial mapping from CRI scheme numbers to URI scheme names.

CRI scheme number	URI scheme name	Reference
0	coap	[RFC-XXXX]
1	coaps	[RFC-XXXX]
2	http	[RFC-XXXX]
3	https	[RFC-XXXX]
4	urn	[RFC-XXXX]
5	did	[RFC-XXXX]
6	coap+tcp	[RFC-XXXX]
7	coaps+tcp	[RFC-XXXX]
8	coap+ws	[RFC-XXXX]
9	coaps+ws	[RFC-XXXX]
1059	ms-gamingoverlay	[RFC-XXXX]
1165	snmp	[RFC-XXXX]
1220	cast	[RFC-XXXX]
1242	openid	[RFC-XXXX]
1319	z39.50	[RFC-XXXX]
1328	dweb	[RFC-XXXX]
1466	psyc	[RFC-XXXX]
1528	ms-people	[RFC-XXXX]
1578	jar	[RFC-XXXX]
1658	wpid	[RFC-XXXX]
1762	payment	[RFC-XXXX]
1895	news	[RFC-XXXX]

1905	irc6	[RFC-XXXX]
1926	turns	[RFC-XXXX]
1946	data	[RFC-XXXX]
1982	ens	[RFC-XXXX]
2154	things	[RFC-XXXX]
2284	resource	[RFC-XXXX]
2326	skype	[RFC-XXXX]
2406	videotex	[RFC-XXXX]
2442	dpp	[RFC-XXXX]
2747	upt	[RFC-XXXX]
2754	platform	[RFC-XXXX]
2790	ed2k	[RFC-XXXX]
2796	taler	[RFC-XXXX]
2806	fm	[RFC-XXXX]
2945	ms-newsandinterests	[RFC-XXXX]
3005	xmlrpc.beep	[RFC-XXXX]
3018	ark	[RFC-XXXX]
3119	wss	[RFC-XXXX]
3143	tel	[RFC-XXXX]
3255	vscode-insiders	[RFC-XXXX]
3342	geo	[RFC-XXXX]
3348	rtmfp	[RFC-XXXX]
3358	mtqp	[RFC-XXXX]
3365	filesystem	[RFC-XXXX]

3503	proxy	[RFC-XXXX]
3524	sms	[RFC-XXXX]
3634	jms	[RFC-XXXX]
3646	mid	[RFC-XXXX]
3690	ms-calculator	[RFC-XXXX]
3775	gitoid	[RFC-XXXX]
3783	calculator	[RFC-XXXX]
3786	about	[RFC-XXXX]
3795	facetime	[RFC-XXXX]
3837	ymsgr	[RFC-XXXX]
3886	dict	[RFC-XXXX]
3906	ldaps	[RFC-XXXX]
3920	rtmp	[RFC-XXXX]
3959	ms-settings-proximity	[RFC-XXXX]
4053	fax	[RFC-XXXX]
4102	ms-drive-to	[RFC-XXXX]
4153	res	[RFC-XXXX]
4183	webcal	[RFC-XXXX]
4193	embedded	[RFC-XXXX]
4315	xftp	[RFC-XXXX]
4327	browserext	[RFC-XXXX]
4355	session	[RFC-XXXX]
4373	dav	[RFC-XXXX]
4419	ipps	[RFC-XXXX]

4515	uuid-in-package	[RFC-XXXX]
4549	dhttp	[RFC-XXXX]
4559	web3	[RFC-XXXX]
4590	iris.lwz	[RFC-XXXX]
4598	diaspora	[RFC-XXXX]
4619	rtsps	[RFC-XXXX]
4674	beshare	[RFC-XXXX]
4709	gtalk	[RFC-XXXX]
4714	hxxps	[RFC-XXXX]
4747	xrcp	[RFC-XXXX]
4882	sgn	[RFC-XXXX]
4929	eid	[RFC-XXXX]
4951	submit	[RFC-XXXX]
5099	ar	[RFC-XXXX]
5109	ms-settings-airplanemode	[RFC-XXXX]
5134	steam	[RFC-XXXX]
5150	adt	[RFC-XXXX]
5152	ms-appinstaller	[RFC-XXXX]
5188	bb	[RFC-XXXX]
5217	udp	[RFC-XXXX]
5296	example	[RFC-XXXX]
5347	ms-remotedesktop	[RFC-XXXX]
5410	ms-sttoverlay	[RFC-XXXX]
5425	irc	[RFC-XXXX]

5472	sieve	[RFC-XXXX]
5477	machineProvisioningProgressReporter	[RFC-XXXX]
5480	lvlt	[RFC-XXXX]
5492	sftp	[RFC-XXXX]
5536	ms-excel	[RFC-XXXX]
5557	dlna-playcontainer	[RFC-XXXX]
5705	go	[RFC-XXXX]
5717	fido	[RFC-XXXX]
5728	chrome	[RFC-XXXX]
5823	shc	[RFC-XXXX]
5825	swidpath	[RFC-XXXX]
5883	microsoft.windows.camera.picker	[RFC-XXXX]
5990	crid	[RFC-XXXX]
6007	at	[RFC-XXXX]
6024	hcp	[RFC-XXXX]
6030	content-type	[RFC-XXXX]
6109	jabber	[RFC-XXXX]
6144	dlna-playsingle	[RFC-XXXX]
6189	ms-spd	[RFC-XXXX]
6341	opaquelocktoken	[RFC-XXXX]
6349	soldat	[RFC-XXXX]
6380	z39.50s	[RFC-XXXX]
6388	ms-media-stream-id	[RFC-XXXX]
6411	ms-mixedrealitycapture	[RFC-XXXX]

6462	quic-transport	[RFC-XXXX]
6503	ham	[RFC-XXXX]
6516	nfs	[RFC-XXXX]
6609	ut2004	[RFC-XXXX]
6632	hydrazone	[RFC-XXXX]
6634	adiumxtra	[RFC-XXXX]
6651	tip	[RFC-XXXX]
6658	lpa	[RFC-XXXX]
6730	cstr	[RFC-XXXX]
6755	ms-settings-screenrotation	[RFC-XXXX]
6774	dab	[RFC-XXXX]
6792	ms-inputapp	[RFC-XXXX]
6808	moz	[RFC-XXXX]
6840	acd	[RFC-XXXX]
6863	ms-access	[RFC-XXXX]
6883	im	[RFC-XXXX]
6903	pttp	[RFC-XXXX]
6924	teamspeak	[RFC-XXXX]
6992	payto	[RFC-XXXX]
7074	secret-token	[RFC-XXXX]
7126	iax	[RFC-XXXX]
7225	isostore	[RFC-XXXX]
7226	bitcoincash	[RFC-XXXX]
7285	smb	[RFC-XXXX]

7364	appdata	[RFC-XXXX]
7456	dtn	[RFC-XXXX]
7520	feed	[RFC-XXXX]
7667	ssh	[RFC-XXXX]
7743	ms-transit-to	[RFC-XXXX]
7809	ms-help	[RFC-XXXX]
7812	vscode	[RFC-XXXX]
7856	apt	[RFC-XXXX]
7868	ms-settings-notifications	[RFC-XXXX]
7874	shttp (OBSOLETE)	[RFC-XXXX]
7913	ethereum	[RFC-XXXX]
7923	tv	[RFC-XXXX]
7942	microsoft.windows.camera.multipicker	[RFC-XXXX]
8041	msnim	[RFC-XXXX]
8085	ms-remotedesktop-launch	[RFC-XXXX]
8093	spiffe	[RFC-XXXX]
8099	redis	[RFC-XXXX]
8159	z39.50r	[RFC-XXXX]
8251	brid	[RFC-XXXX]
8300	tftp	[RFC-XXXX]
8387	content	[RFC-XXXX]
8454	wais	[RFC-XXXX]
8506	view-source	[RFC-XXXX]
8519	soap.beep	[RFC-XXXX]

8577	attachment	[RFC-XXXX]
8601	gopher	[RFC-XXXX]
8687	ircs	[RFC-XXXX]
8713	callto	[RFC-XXXX]
8765	bolo	[RFC-XXXX]
8766	notes	[RFC-XXXX]
8775	ipn	[RFC-XXXX]
8830	ms-infopath	[RFC-XXXX]
9075	ms-settings	[RFC-XXXX]
9136	ms-useractivityset	[RFC-XXXX]
9154	modem	[RFC-XXXX]
9186	bitcoin	[RFC-XXXX]
9198	ms-settings-privacy	[RFC-XXXX]
9204	cap	[RFC-XXXX]
9278	com-eventbrite-attendee	[RFC-XXXX]
9312	pkcs11	[RFC-XXXX]
9318	ipp	[RFC-XXXX]
9338	rediss	[RFC-XXXX]
9444	grd	[RFC-XXXX]
9453	ms-screensketch	[RFC-XXXX]
9487	matrix	[RFC-XXXX]
9520	xcon-userid	[RFC-XXXX]
9535	sips	[RFC-XXXX]
9544	simpleledger	[RFC-XXXX]

9585	mvn	[RFC-XXXX]
9770	keyparc	[RFC-XXXX]
9805	magnet	[RFC-XXXX]
9816	vsls	[RFC-XXXX]
9859	drm	[RFC-XXXX]
9875	hcap	[RFC-XXXX]
9910	wtai	[RFC-XXXX]
9965	num	[RFC-XXXX]
9981	ms-settings-language	[RFC-XXXX]
10119	imap	[RFC-XXXX]
10147	query	[RFC-XXXX]
10176	ves	[RFC-XXXX]
10196	acr	[RFC-XXXX]
10225	barion	[RFC-XXXX]
10229	acct	[RFC-XXXX]
10238	palm	[RFC-XXXX]
10241	ocf	[RFC-XXXX]
10247	lid	[RFC-XXXX]
10317	h323	[RFC-XXXX]
10327	aim	[RFC-XXXX]
10333	turn	[RFC-XXXX]
10361	ms-stickers	[RFC-XXXX]
10373	ms-settings-location	[RFC-XXXX]
10380	dvb	[RFC-XXXX]

10467	xcon	[RFC-XXXX]
10518	ms-screenclip	[RFC-XXXX]
10551	pop	[RFC-XXXX]
10583	dat	[RFC-XXXX]
10591	ms-settings-nfctransactions	[RFC-XXXX]
10640	ms-settings-cloudstorage	[RFC-XXXX]
10687	afs	[RFC-XXXX]
10740	mqtt	[RFC-XXXX]
10744	gizmoproject	[RFC-XXXX]
10831	amss	[RFC-XXXX]
10868	mailserver	[RFC-XXXX]
10926	ni	[RFC-XXXX]
10995	telnet	[RFC-XXXX]
11055	gg	[RFC-XXXX]
11060	blob	[RFC-XXXX]
11072	ms-settings-emailandaccounts	[RFC-XXXX]
11130	ms-project	[RFC-XXXX]
11255	xri	[RFC-XXXX]
11315	msrp	[RFC-XXXX]
11351	ms-settings-connectabledevices	[RFC-XXXX]
11393	cabal	[RFC-XXXX]
11428	nih	[RFC-XXXX]
11467	ms-whiteboard	[RFC-XXXX]
11533	smp	[RFC-XXXX]

11537	vnc	[RFC-XXXX]
11583	graph	[RFC-XXXX]
11645	dvx	[RFC-XXXX]
11718	lorawan	[RFC-XXXX]
11742	lastfm	[RFC-XXXX]
11799	w3	[RFC-XXXX]
11804	mumble	[RFC-XXXX]
11824	feedready	[RFC-XXXX]
11857	microsoft.windows.camera	[RFC-XXXX]
11892	wcr	[RFC-XXXX]
11945	ms-mobileplans	[RFC-XXXX]
11950	ms-settings-lock	[RFC-XXXX]
11962	ws	[RFC-XXXX]
11999	rtspu	[RFC-XXXX]
12029	ms-settings-displays-topology	[RFC-XXXX]
12068	file	[RFC-XXXX]
12102	mailto	[RFC-XXXX]
12174	ms-launchremotedesktop	[RFC-XXXX]
12242	cvs	[RFC-XXXX]
12337	mms	[RFC-XXXX]
12400	ssb	[RFC-XXXX]
12422	iris.xpc	[RFC-XXXX]
12458	starknet	[RFC-XXXX]
12478	qb	[RFC-XXXX]

12493	mss	[RFC-XXXX]
12502	ventrilo	[RFC-XXXX]
12525	ms-lockscreencomponent-config	[RFC-XXXX]
12566	icap	[RFC-XXXX]
12569	mupdate	[RFC-XXXX]
12599	paparazzi	[RFC-XXXX]
12634	fish	[RFC-XXXX]
12644	sip	[RFC-XXXX]
12699	mt	[RFC-XXXX]
12705	acap	[RFC-XXXX]
12718	casts	[RFC-XXXX]
12726	reload	[RFC-XXXX]
12732	spotify	[RFC-XXXX]
12806	fuchsia-pkg	[RFC-XXXX]
12823	ms-gamebarservices	[RFC-XXXX]
12876	hyper	[RFC-XXXX]
12932	dns	[RFC-XXXX]
13014	doi	[RFC-XXXX]
13026	ms-settings-power	[RFC-XXXX]
13068	git	[RFC-XXXX]
13094	openpgp4fpr	[RFC-XXXX]
13098	ms-secondary-screen-controller	[RFC-XXXX]
13228	mvrps	[RFC-XXXX]
13285	snews	[RFC-XXXX]

13340	smtp	[RFC-XXXX]
13348	pack	[RFC-XXXX]
13362	teliaeid	[RFC-XXXX]
13372	mongodb	[RFC-XXXX]
13404	afp	[RFC-XXXX]
13440	msrps	[RFC-XXXX]
13442	ldap	[RFC-XXXX]
13451	mvrp	[RFC-XXXX]
13499	nntp	[RFC-XXXX]
13608	onenote	[RFC-XXXX]
13650	sarif	[RFC-XXXX]
13680	elsi	[RFC-XXXX]
13829	otpauth	[RFC-XXXX]
13846	info	[RFC-XXXX]
13862	aaa	[RFC-XXXX]
13923	svn	[RFC-XXXX]
13986	iris	[RFC-XXXX]
14010	lbry	[RFC-XXXX]
14034	ms-search	[RFC-XXXX]
14090	ms-browser-extension	[RFC-XXXX]
14153	maps	[RFC-XXXX]
14162	swid	[RFC-XXXX]
14168	ms-officeapp	[RFC-XXXX]
14180	ms-settings-bluetooth	[RFC-XXXX]

14310	ms-enrollment	[RFC-XXXX]
14347	dntp	[RFC-XXXX]
14364	ms-walk-to	[RFC-XXXX]
14366	ms-getoffice	[RFC-XXXX]
14367	thismessage	[RFC-XXXX]
14460	message	[RFC-XXXX]
14477	prospero	[RFC-XXXX]
14526	aaas	[RFC-XXXX]
14595	market	[RFC-XXXX]
14627	stun	[RFC-XXXX]
14667	chrome-extension	[RFC-XXXX]
14830	itms	[RFC-XXXX]
14860	ms-whiteboard-cmd	[RFC-XXXX]
14867	wifi	[RFC-XXXX]
14868	icon	[RFC-XXXX]
14878	ftp	[RFC-XXXX]
14901	stuns	[RFC-XXXX]
14906	mqtts	[RFC-XXXX]
14936	ms-settings-workplace	[RFC-XXXX]
14962	tn3270	[RFC-XXXX]
14972	pres	[RFC-XXXX]
14982	p1	[RFC-XXXX]
15061	android	[RFC-XXXX]
15118	simplex	[RFC-XXXX]

15163	ms-visio	[RFC-XXXX]
15202	cid	[RFC-XXXX]
15206	unreal	[RFC-XXXX]
15230	tool	[RFC-XXXX]
15254	ms-secondary-screen-setup	[RFC-XXXX]
15267	rtsp	[RFC-XXXX]
15306	xfire	[RFC-XXXX]
15358	xmpp	[RFC-XXXX]
15361	ms-settings-cellular	[RFC-XXXX]
15579	v-event	[RFC-XXXX]
15639	iris.beep	[RFC-XXXX]
15641	wyciwyg	[RFC-XXXX]
15645	ms-meetnow	[RFC-XXXX]
15679	ms-search-repair	[RFC-XXXX]
15773	ms-settings-camera	[RFC-XXXX]
15776	ms-virtualtouchpad	[RFC-XXXX]
15805	xmlrpc.beeps	[RFC-XXXX]
15972	ipfs	[RFC-XXXX]
15994	ms-settings-wifi	[RFC-XXXX]
16051	aw	[RFC-XXXX]
16069	first-run-pen-experience	[RFC-XXXX]
16079	oid	[RFC-XXXX]
16134	iris.xpcs	[RFC-XXXX]
16138	drop	[RFC-XXXX]

16194	ms-publisher	[RFC-XXXX]
16281	leaptofrogans	[RFC-XXXX]
16292	rmi	[RFC-XXXX]
16300	soap.beeps	[RFC-XXXX]
16377	tag	[RFC-XXXX]
16585	ms-word	[RFC-XXXX]
16632	onenote-cmd	[RFC-XXXX]
16645	ms-powerpoint	[RFC-XXXX]
16728	hxxp	[RFC-XXXX]
16729	secondlife	[RFC-XXXX]
16884	rsync	[RFC-XXXX]
16918	vemmi	[RFC-XXXX]
16933	ipns	[RFC-XXXX]
17039	swh	[RFC-XXXX]
17068	pwid	[RFC-XXXX]
17097	dtmi	[RFC-XXXX]
17134	dis	[RFC-XXXX]
17170	iotdisco	[RFC-XXXX]
17175	ms-restoretabcompanion	[RFC-XXXX]
17264	service	[RFC-XXXX]
17315	finger	[RFC-XXXX]
17361	web+ap	[RFC-XXXX]
17381	ms-eyecontrolspeech	[RFC-XXXX]

Table 6: Mapping Scheme Numbers to Scheme Names

The assignments from this table can be extracted from the XML form of this document (when stored in a file "this.xml") into CSV form [RFC4180] using this short Ruby program:

```
require 'rexml/document'; include REXML
XPath.each(Document.new(File.read("this.xml")),"/rfc/back//tr") {|r|
  puts XPath.each(r,"td").map{|d|d.text()}[0..1].join(",")}
```

Appendix B. The Small Print

This appendix lists a few corner cases of URI semantics that implementers of CRIs need to be aware of, but that are not representative of the normal operation of CRIs.

SP1. Initial (Lone/Leading) Empty Path Segments:

* _Lone empty path segments:_ As per [STD66], s://x is distinct from s://x/ -- i.e., a URI with an empty path ([] in CRI) is different from one with a lone empty path segment ([""]). However, in HTTP and CoAP, they are implicitly aliased (for CoAP, in item 8 of Section 6.4 of [RFC7252]). As per item 7 of Section 6.5 of [RFC7252], recomposition of a URI without Uri-Path Options from the other URI-related CoAP Options produces s://x/, not s://x -- CoAP prefers the lone empty path segment form. Similarly, after discussing HTTP semantics, Section 6.2.3 of RFC 3986 [STD66] states:

  > In general, a URI that uses the generic syntax for authority with an empty path should be normalized to a path of "/".

* _Leading empty path segments without authority_: Somewhat related, note also that URIs and URI references that do not carry an authority cannot represent initial empty path segments (i.e., that are followed by further path segments): s://x//foo works, but in a s://foo URI or an (absolute-path) URI reference of the form //foo the double slash would be mis-parsed as leading in to an authority.

SP2. Constraints (Section 2) of CRIs/basic CRIs

While most URIs in everyday use can be converted to CRIs and back to URIs matching the input after syntax-based normalization of the URI, these URIs illustrate the constraints by example:

* https://host%ffname, https://example.com/x?data=%ff

All URI components must, after percent decoding, be valid UTF-8 encoded text. Bytes that are not valid UTF-8 show up, for example, in BitTorrent web seeds.

* https://example.com/component%3bone;component%3btwo, http://example.com/component%3dequals

  While delimiters can be used in an escaped and unescaped form in URIs with generally distinct meanings, basic CRIs (i.e., without percent-encoded text Section 7.1) only support one escapable delimiter character per component, which is the delimiter by which the component is split up in the CRI.

  Note that the separators . (for authority parts), / (for paths), & (for query parameters) are special in that they are syntactic delimiters of their respective components in CRIs. Thus, the following examples _are_ convertible to basic CRIs:

  https://interior%2edot/

  https://example.com/path%2fcomponent/second-component

  https://example.com/x?ampersand=%26&questionmark=?

* https://alice@example.com/

  The user information can be expressed in CRIs if the "userinfo" feature is present. The URI https://@example.com is represented as [-4, [false, "", "example", "com"]]; the false serves as a marker that the next element is the userinfo.

  The rules do not cater for unencoded ":" in userinfo, which is commonly considered a deprecated inclusion of a literal password.

Appendix C. CBOR Extended Diagnostic Notation (EDN): The "cri" Extension

[I-D.ietf-cbor-edn-literals] more rigorously defines and further extends the CBOR Extended Diagnostic Notation (EDN), as originally introduced in Section 8 of RFC 8949 [STD94] and extended in Appendix G of [RFC8610]. Among others, it provides an extension point for "application-extension identifiers" that can be used to notate CBOR data items in application-specific ways.

The present document defines and registers (Section 11.3) the application-extension identifier "cri", which can be used to notate an EDN literal for a CRI reference as defined in this document.

The text of the literal is a URI Reference as per [STD66] or an IRI Reference as per [RFC3987].

The value of the literal is a CRI reference that can be converted to the text of the literal using the procedure of Section 6.1. Note that there may be more than one CRI reference that can be converted to the URI/IRI reference given; implementations are expected to favor the simplest variant available and make non-surprising choices otherwise.

As an example, the CBOR diagnostic notation

```
cri'https://example.com/bottarga/shaved'
```

is equivalent to

```
[-4, ["example", "com"], ["bottarga", "shaved"]]
```

See Appendix C.1 for an ABNF definition for the content of cri literals.

C.1. cri: ABNF Definition of URI Representation of a CRI

The syntax of the content of cri literals can be described by the ABNF for URI-reference in Section 4.1 of RFC 3986 [STD66] with certain re-arrangements taken from Figure 5 of [I-D.ietf-cbor-edn-literals]; these are reproduced in Figure 2. If the content is not ASCII only (i.e., for IRIs), first apply Section 3.1 of [RFC3987] and apply this grammar to the result.

```
app-string-cri = URI-reference
; ABNF from RFC 3986:

URI           = scheme ":" hier-part [ "?" query ] [ "#" fragment ]

hier-part     = "//" authority path-abempty
                 / path-absolute
                 / path-rootless
                 / path-empty

URI-reference = URI / relative-ref

absolute-URI  = scheme ":" hier-part [ "?" query ]
```

```
relative-ref  = relative-part [ "?" query ] [ "#" fragment ]

relative-part = "//" authority path-abempty
                / path-absolute
                / path-noscheme
                / path-empty

scheme        = ALPHA *( ALPHA / DIGIT / "+" / "-" / "." )

authority     = [ userinfo "@" ] host [ ":" port ]
userinfo      = *( unreserved / pct-encoded / sub-delims / ":" )
host          = IP-literal / IPv4address / reg-name
port          = *DIGIT

IP-literal    = "[" ( IPv6address / IPvFuture  ) "]"

IPvFuture     = "v" 1*HEXDIG "." 1*( unreserved / sub-delims / ":" )

; Use IPv6address, h16, ls32, IPv4adress, dec-octet as re-arranged
; for PEG Compatibility in Figure 5 of [I-D.ietf-cbor-edn-literals]:

IPv6address   =                            6( h16 ":" ) ls32
              /                       "::" 5( h16 ":" ) ls32
              / [ h16               ] "::" 4( h16 ":" ) ls32
              / [ h16 *1( ":" h16 ) ] "::" 3( h16 ":" ) ls32
              / [ h16 *2( ":" h16 ) ] "::" 2( h16 ":" ) ls32
              / [ h16 *3( ":" h16 ) ] "::"    h16 ":"   ls32
              / [ h16 *4( ":" h16 ) ] "::"              ls32
              / [ h16 *5( ":" h16 ) ] "::"              h16
              / [ h16 *6( ":" h16 ) ] "::"

h16           = 1*4HEXDIG
ls32          = ( h16 ":" h16 ) / IPv4address
IPv4address   = dec-octet "." dec-octet "." dec-octet "." dec-octet
dec-octet     = "25" %x30-35         ; 250-255
              / "2" %x30-34 DIGIT    ; 200-249
              / "1" 2DIGIT           ; 100-199
              / %x31-39 DIGIT        ; 10-99
              / DIGIT                ; 0-9
ALPHA         = %x41-5a / %x61-7a
DIGIT         = %x30-39
HEXDIG        = DIGIT / "A" / "B" / "C" / "D" / "E" / "F"
; case insensitive matching, i.e., including lower case

reg-name      = *( unreserved / pct-encoded / sub-delims )

path          = path-abempty    ; begins with "/" or is empty
                / path-absolute   ; begins with "/" but not "//"
```

```
                    / path-noscheme   ; begins with a non-colon segment
                    / path-rootless   ; begins with a segment
                    / path-empty      ; zero characters

   path-abempty  = *( "/" segment )
   path-absolute = "/" [ segment-nz *( "/" segment ) ]
   path-noscheme = segment-nz-nc *( "/" segment )
   path-rootless = segment-nz *( "/" segment )
   path-empty    = 0<pchar>

   segment       = *pchar
   segment-nz    = 1*pchar
   segment-nz-nc = 1*( unreserved / pct-encoded / sub-delims / "@" )
                    ; non-zero-length segment without any colon ":"

   pchar         = unreserved / pct-encoded / sub-delims / ":" / "@"

   query         = *( pchar / "/" / "?" )

   fragment      = *( pchar / "/" / "?" )

   pct-encoded   = "%" HEXDIG HEXDIG

   unreserved    = ALPHA / DIGIT / "-" / "." / "_" / "~"
   reserved      = gen-delims / sub-delims
   gen-delims    = ":" / "/" / "?" / "#" / "[" / "]" / "@"
   sub-delims    = "!" / "$" / "&" / "'" / "(" / ")"
                    / "*" / "+" / "," / ";" / "="
```

Figure 2: ABNF Definition of URI Representation of a CRI

Appendix D. Change Log

This section is to be removed before publishing as an RFC.

Changes from -14 to -15

* Make scheme numbers unsigned and map them to negative numbers used as scheme-id values

Changes from -09 to -14

* Editorial changes; move some examples to Appendix B, break up railroad diagram; mention commonalities with (and tiny difference from) CoAP Options; mention failure of percent-encoding for dots in host-name components

* Explicitly mention invalid case in Section 2.1, Paragraph 5, Item 3 (rootless CRIs without authority that do not have a path component)

* Generalize Section 7, discuss PET (percent-encoded text) extension in more detail

* Add registry of URI scheme numbers (Appendix A, Section 11)

* Add user information to the authority ("userinfo" feature)

* Figure 1: Use separate rule for CRI, allow [] for query in CRI Reference; generalize scheme numbers, add userinfo; add list of additional requirements in prose (Section 5.1, Paragraph 5)

* Discuss Error handling and extensibility (Section 5.2.1)

* Conversion to URI: Handle : in first pathname component of a CRI-Reference (Section 6.1, Paragraph 2.6.1)

* Add Christian Amsüss as contributor

* Add CBOR EDN application-extension "cri" (see Appendix C and Section 11.3).

* Add Section on CoAP integration (and new CoAP Options Proxy-Cri and Proxy-Scheme-Number).

Changes from -08 to -09

* Identify more esoteric features with a CDDL ".feature".

* Clarify that well-formedness requires removing trailing nulls.

* Fragments can contain PET.

* Percent-encoded text in PET is treated as byte strings.

* URIs with an authority but a completely empty path (e.g., http://example.com): CRIs with an authority component no longer always produce at least a slash in the path component.

  For generic schemes, the conversion of scheme://example.com to a CRI is now possible because CRI produces a URI with an authority not followed by a slash following the updated rules of Section 6.1. Schemes like http and coap do not distinguish between the empty path and the path containing a single slash when an authority is set (as recommended in [STD66]). For these

schemes, that equivalence allows implementations to convert the just-a-slash URI to a CRI with a zero length path array (which, however, when converted back, does not produce a slash after the authority).

(Add an appendix "the small print" for more detailed discussion of pesky corner cases like this.)

Changes from -07 to -08

* Fix the encoding of NOAUTH-NOSLASH / NOAUTH-LEADINGSLASH

* Add URN and DID schemes, add example.

* Add PET

* Remove hopeless attempt to encode "remote trailing nulls" rule in CDDL (which is not a transformation language).

Changes from -06 to -07

* More explicitly discuss constraints (Section 2), add examples (Appendix B, Paragraph 6, Item 1).

* Make CDDL more explicit about special simple values.

* Lots of gratuitous changes from XML2RFC redefinition of <tt> semantics.

Changes from -05 to -06

* rework authority:

  - split reg-names at dots;

  - add optional zone identifiers [RFC6874] to IP addresses

Changes from -04 to -05

* Simplify CBOR structure.

* Add implementation status section.

Changes from -03 to -04:

* Minor editorial improvements.

* Renamed path.type/path-type to discard.

* Renamed option to section, substructured into items.

* Simplified the table "resolution-variables".

* Use the CBOR structure inspired by Jim Schaad's proposals.

Changes from -02 to -03:

* Expanded the set of supported schemes (#3).

* Specified creation, normalization and comparison (#9).

* Clarified the default value of the path.type option (#33).

* Removed the append-relation path.type option (#41).

* Renumbered the remaining path.types.

* Renumbered the option numbers.

* Restructured the document.

* Minor editorial improvements.

Changes from -01 to -02:

* Changed the syntax of schemes to exclude upper case characters (#13).

* Minor editorial improvements (#34 #37).

Changes from -00 to -01:

* None.

Acknowledgements

CRIs were developed by Klaus Hartke for use in the Constrained RESTful Application Language (CoRAL). The current author team is completing this work with a view to achieve good integration with the potential use cases, both inside and outside of CoRAL.

Thanks to Christian Amsüss, Thomas Fossati, Ari Keränen, Jim Schaad, Dave Thaler, and Marco Tiloca for helpful comments and discussions that have shaped the document.

Contributors

Klaus Hartke
Ericsson
Torshamnsgatan 23
SE-16483 Stockholm
Sweden
Email: klaus.hartke@ericsson.com

Christian Amsüss
Hollandstr. 12/4
1020 Vienna
Austria
Email: christian@amsuess.com

Authors' Addresses

Carsten Bormann (editor)
Universität Bremen TZI
Postfach 330440
D-28359 Bremen
Germany
Phone: +49-421-218-63921
Email: cabo@tzi.org

Henk Birkholz
Fraunhofer SIT
Rheinstrasse 75
64295 Darmstadt
Germany
Email: henk.birkholz@sit.fraunhofer.de

CoRE Working Group
Internet-Draft
Intended status: Standards Track
Expires: 11 October 2024

F. Palombini
Ericsson
M. Tiloca
R. Höglund
RISE AB
S. Hristozov
Fraunhofer AISEC
G. Selander
Ericsson
9 April 2024

Using Ephemeral Diffie-Hellman Over COSE (EDHOC) with the Constrained Application Protocol (CoAP) and Object Security for Constrained RESTful Environments (OSCORE)

draft-ietf-core-oscore-edhoc-11

Abstract

The lightweight authenticated key exchange protocol Ephemeral Diffie-Hellman Over COSE (EDHOC) can be run over the Constrained Application Protocol (CoAP) and used by two peers to establish a Security Context for the security protocol Object Security for Constrained RESTful Environments (OSCORE). This document details this use of the EDHOC protocol, by specifying a number of additional and optional mechanisms. These especially include an optimization approach for combining the execution of EDHOC with the first OSCORE transaction. This combination reduces the number of round trips required to set up an OSCORE Security Context and to complete an OSCORE transaction using that Security Context.

Discussion Venues

This note is to be removed before publishing as an RFC.

Discussion of this document takes place on the Constrained RESTful Environments Working Group mailing list (core@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/core/.

Source for this draft and an issue tracker can be found at https://github.com/core-wg/oscore-edhoc.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction 3
  1.1. Terminology 4
2. EDHOC Overview 4
3. EDHOC Combined with OSCORE 8
  3.1. EDHOC Option 10
  3.2. Client Processing 11
    3.2.1. Processing of the EDHOC + OSCORE Request 11
    3.2.2. Supporting Block-wise 13
  3.3. Server Processing 13
    3.3.1. Processing of the EDHOC + OSCORE Request 13
    3.3.2. Supporting Block-wise 15
  3.4. Example of EDHOC + OSCORE Request 15
4. Use of EDHOC Connection Identifiers with OSCORE 16
  4.1. Additional Processing of EDHOC Messages 17
    4.1.1. Initiator Processing of Message 1 17
    4.1.2. Responder Processing of Message 2 17
    4.1.3. Initiator Processing of Message 2 18
5. Extension and Consistency of Application Profiles 18
6. Web Linking 18
7. Security Considerations 21

8. IANA Considerations 22
8.1. CoAP Option Numbers Registry 22
8.2. Target Attributes Registry 23
8.3. EDHOC Authentication Credential Types Registry 24
8.4. Expert Review Instructions 25
9. References 26
9.1. Normative References 26
9.2. Informative References 28
Appendix A. Document Updates 28
A.1. Version -10 to -11 28
A.2. Version -09 to -10 29
A.3. Version -08 to -09 29
A.4. Version -07 to -08 29
A.5. Version -06 to -07 29
A.6. Version -05 to -06 30
A.7. Version -04 to -05 30
A.8. Version -03 to -04 31
A.9. Version -02 to -03 31
A.10. Version -01 to -02 32
A.11. Version -00 to -01 32
Acknowledgments 32
Authors' Addresses 32

1. Introduction

Ephemeral Diffie-Hellman Over COSE (EDHOC) [RFC9528] is a lightweight authenticated key exchange protocol, especially intended for use in constrained scenarios. In particular, EDHOC messages can be transported over the Constrained Application Protocol (CoAP) [RFC7252] and used for establishing a Security Context for Object Security for Constrained RESTful Environments (OSCORE) [RFC8613].

This document details the use of the EDHOC protocol with CoAP and OSCORE, and specifies a number of additional and optional mechanisms. These especially include an optimization approach that combines the EDHOC execution with the first OSCORE transaction (see Section 3). This allows for a minimum number of two round trips necessary to setup the OSCORE Security Context and complete an OSCORE transaction, e.g., when an IoT device gets configured in a network for the first time.

This optimization is desirable, since the number of message exchanges can have a substantial impact on the latency of conveying the first OSCORE request, when using certain radio technologies.

Without this optimization, it is not possible to achieve the minimum number of two round trips. This optimization makes it possible, since the message_3 of the EDHOC protocol can be made relatively small (see Section 1.2 of [RFC9528]), thus allowing additional OSCORE-protected CoAP data within target MTU sizes.

The minimum number of two round trips can be achieved only if the default, forward message flow of EDHOC is used, i.e., when a CoAP client acts as EDHOC Initiator and a CoAP server acts as EDHOC Responder. The performance advantage of using this optimization can be lost when used in combination with Block-wise transfers [RFC7959] that rely on specific parameter values and block sizes.

Furthermore, this document defines a number of parameters corresponding to different information elements of an EDHOC application profile (see Section 6). These can be specified as target attributes in the link to an EDHOC resource associated with that application profile, thus enabling an enhanced discovery of such a resource for CoAP 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.

The reader is expected to be familiar with terms and concepts defined in CoAP [RFC7252], CBOR [RFC8949], OSCORE [RFC8613], and EDHOC [RFC9528].

2. EDHOC Overview

This section is not normative and summarizes what is specified in [RFC9528], in particular its Appendix A.2. Thus, it provides a baseline for the enhancements in the subsequent sections.

The EDHOC protocol specified in [RFC9528] allows two peers to agree on a cryptographic secret, in a mutually-authenticated way and by using Diffie-Hellman ephemeral keys to achieve forward secrecy. The two peers are denoted as Initiator and Responder, as the one sending or receiving the initial EDHOC message_1, respectively.

After successful processing of EDHOC message_3, both peers agree on a cryptographic secret that can be used to derive further security material, and especially to establish an OSCORE Security Context [RFC8613]. The Responder can also send an optional EDHOC message_4 to achieve key confirmation, e.g., in deployments where no protected application message is sent from the Responder to the Initiator.

Appendix A.2 of [RFC9528] specifies how to transfer EDHOC over CoAP. That is, the EDHOC data (i.e., the EDHOC message possibly with a prepended connection identifier) are transported in the payload of CoAP requests and responses. The default, forward message flow of EDHOC consists in the CoAP client acting as Initiator and the CoAP server acting as Responder (see Appendix A.2.1 of [RFC9528]). Alternatively, the two roles can be reversed, as per the reverse message flow of EDHOC (see Appendix A.2.2 of [RFC9528]). In the rest of this document, EDHOC messages are considered to be transferred over CoAP.

Figure 1 shows a successful execution of EDHOC, with a CoAP client and a CoAP server running EDHOC as Initiator and Responder, respectively. In particular, it extends Figure 10 from Appendix A.2.1 of [RFC9528], by highlighting when the two peers perform EDHOC verification and establish the OSCORE Security Context, and by adding an exchange of OSCORE-protected CoAP messages after completing the EDHOC execution.

That is, the client sends a POST request to a reserved _EDHOC resource_ at the server, by default at the Uri-Path "/.well-known/edhoc". The request payload consists of the CBOR simple value true (0xf5) concatenated with EDHOC message_1, which also includes the EDHOC connection identifier C_I of the client encoded as per Section 3.3 of [RFC9528]. The request has Content-Format application/cid-edhoc+cbor-seq.

This triggers the EDHOC execution at the server, which replies with a 2.04 (Changed) response. The response payload consists of EDHOC message_2, which also includes the EDHOC connection identifier C_R of the server encoded as per Section 3.3 of [RFC9528]. The response has Content-Format application/edhoc+cbor-seq.

Finally, the client sends a POST request to the same EDHOC resource used earlier when it sent EDHOC message_1. The request payload consists of the EDHOC connection identifier C_R encoded as per Section 3.3 of [RFC9528], concatenated with EDHOC message_3. The request has Content-Format application/cid-edhoc+cbor-seq.

After this exchange takes place, and after successful verifications as specified in the EDHOC protocol, the client and server can derive an OSCORE Security Context, as defined in Appendix A.1 of [RFC9528]. After that, they can use OSCORE to protect their communications as per [RFC8613]. Note that the EDHOC Connection Identifier C_R is used as the OSCORE Sender ID of the client (see Appendix A.1 of [RFC9528]). Therefore, C_R is transported in the 'kid' field of the OSCORE Option of the OSCORE Request (see Section 6.1 of [RFC8613]).

The client and server are required to agree in advance on certain information and parameters describing how they should use EDHOC. These are specified in an application profile associated with the EDHOC resource addressed (see Section 3.9 of [RFC9528].

```
      CoAP client                                          CoAP server
    (EDHOC Initiator)                                   (EDHOC Responder)
            |                                                   |
            |                                                   |
            | ----------------- EDHOC Request -----------------> |
            |   Header: 0.02 (POST)                             |
            |   Uri-Path: "/.well-known/edhoc"                  |
            |   Content-Format: application/cid-edhoc+cbor-seq  |
            |   Payload: true, EDHOC message_1                  |
            |                                                   |
            | <---------------- EDHOC Response------------------ |
            |       Header: 2.04 (Changed)                      |
            |       Content-Format: application/edhoc+cbor-seq  |
            |       Payload: EDHOC message_2                    |
            |                                                   |
    EDHOC verification                                          |
            |                                                   |
            | ----------------- EDHOC Request -----------------> |
            |   Header: 0.02 (POST)                             |
            |   Uri-Path: "/.well-known/edhoc"                  |
            |   Content-Format: application/cid-edhoc+cbor-seq  |
            |   Payload: C_R, EDHOC message_3                   |
            |                                                   |
            |                                          EDHOC verification
            |                                                     +
            |                                             OSCORE Sec Ctx
            |                                              Derivation
            |                                                   |
            | <---------------- EDHOC Response------------------ |
            |       Header: 2.04 (Changed)                      |
            |       Content-Format: application/edhoc+cbor-seq  |
            |       Payload: EDHOC message_4                    |
            |                                                   |
    OSCORE Sec Ctx                                              |
     Derivation                                                 |
            |                                                   |
            | ---------------- OSCORE Request -----------------> |
            |   Header: 0.02 (POST)                             |
            |   OSCORE: { ... ; kid: C_R }                      |
            |   Payload: OSCORE-protected data                  |
            |                                                   |
            | <--------------- OSCORE Response ----------------- |
            |                 Header: 2.04 (Changed)            |
            |                 OSCORE: { ... }                   |
            |                 Payload: OSCORE-protected data    |
            |                                                   |
```

Figure 1: EDHOC and OSCORE run sequentially. The optional message_4 is included in this example.

As shown in Figure 1, this sequential flow where EDHOC is run first and then OSCORE is used takes three round trips to complete.

Section 3 defines an optimization for combining EDHOC with the first OSCORE transaction. This reduces the number of round trips required to set up an OSCORE Security Context and to complete an OSCORE transaction using that Security Context.

3. EDHOC Combined with OSCORE

This section defines an optimization for combining the EDHOC message exchange with the first OSCORE transaction, thus minimizing the number of round trips between the two peers to the absolute possible minimum of two round trips.

To this end, this approach can be used only if the default, forward message flow of EDHOC is used, i.e., when the client acts as Initiator and the server acts as Responder. The same is not possible in the case with reversed roles as per the reverse message flow of EDHOC.

When running the sequential flow of Section 2, the client has all the information to derive the OSCORE Security Context already after receiving EDHOC message_2 and before sending EDHOC message_3.

Hence, the client can potentially send both EDHOC message_3 and the subsequent OSCORE Request at the same time. On a semantic level, this requires sending two REST requests at once, as in Figure 2.

```
     CoAP client                                          CoAP server
   (EDHOC Initiator)                                   (EDHOC Responder)
          |                                                     |
          | ------------------ EDHOC Request -----------------> |
          |   Header: 0.02 (POST)                               |
          |   Uri-Path: "/.well-known/edhoc"                    |
          |   Content-Format: application/cid-edhoc+cbor-seq    |
          |   Payload: true, EDHOC message_1                    |
          |                                                     |
          | <----------------- EDHOC Response------------------ |
          |        Header: Changed (2.04)                       |
          |        Content-Format: application/edhoc+cbor-seq   |
          |        Payload: EDHOC message_2                     |
          |                                                     |
   EDHOC verification                                           |
          +                                                     |
    OSCORE Sec Ctx                                              |
      Derivation                                                |
          |                                                     |
          | -------------- EDHOC + OSCORE Request ------------> |
          |   Header: 0.02 (POST)                               |
          |   OSCORE: { ... ; kid: C_R }                        |
          |   Payload: EDHOC message_3 + OSCORE-protected data  |
          |                                                     |
          |                                            EDHOC verification
          |                                                     +
          |                                              OSCORE Sec Ctx
          |                                                 Derivation
          |                                                     |
          | <--------------- OSCORE Response ------------------ |
          |                    Header: 2.04 (Changed)           |
          |                    OSCORE: { ... }                  |
          |                    Payload: OSCORE-protected data   |
          |                                                     |
```

Figure 2: EDHOC and OSCORE combined.

To this end, the specific approach defined in this section consists of sending a single EDHOC + OSCORE request, which conveys the pair (C_R, EDHOC message_3) within an OSCORE-protected CoAP message.

That is, the EDHOC + OSCORE request is composed of the following two parts combined together in a single CoAP message. The steps for processing the EDHOC + OSCORE request and the two parts combined in there are defined in Section 3.2.1 and Section 3.3.1.

* The OSCORE Request from Figure 1, which is also in this case sent to a protected resource, with the correct CoAP method and options intended for accessing that resource.

* EDHOC data consisting of the pair (C_R, EDHOC message_3) required for completing the EDHOC session, transported as follows:

  - C_R is the OSCORE Sender ID of the client and hence transported in the 'kid' field of the OSCORE Option (see Section 6.1 of [RFC8613]). Unlike in the sequential workflow shown in Figure 1, C_R is thus not transported in the payload of the EDHOC + OSCORE request.

  - EDHOC message_3 is transported in the payload of the EDHOC + OSCORE request, prepended to the payload of the OSCORE Request. This is because EDHOC message_3 may be too large to be included in a CoAP Option, e.g., when conveying a large public key certificate chain as ID_CRED_I (see Section 3.5.3 of [RFC9528]) or when conveying large External Authorization Data as EAD_3 (see Section 3.8 of [RFC9528]).

The rest of this section specifies how to transport the data in the EDHOC + OSCORE request and their processing order. In particular, the use of this approach is explicitly signalled by including an EDHOC Option (see Section 3.1) in the EDHOC + OSCORE request. The processing of the EDHOC + OSCORE request is specified in Section 3.2 for the client side and in Section 3.3 for the server side.

3.1. EDHOC Option

This section defines the EDHOC Option. The option is used in a CoAP request, to signal that the request payload conveys both an EDHOC message_3 and OSCORE-protected data, combined together.

The EDHOC Option has the properties summarized in Table 1, which extends Table 4 of [RFC7252]. The option is Critical, Safe-to-Forward, and part of the Cache-Key. The option MUST occur at most once and MUST be empty. If any value is sent, the recipient MUST ignore it. (Future documents may update the definition of the option, by expanding its semantics and specifying admitted values.) The option is intended only for CoAP requests and is of Class U for OSCORE [RFC8613].

No.	C	U	N	R	Name	Format	Length	Default
21	x				EDHOC	Empty	0	(none)

Table 1: The EDHOC Option. C=Critical, U=Unsafe, N=NoCacheKey, R=Repeatable

The presence of this option means that the message payload also contains EDHOC data, which must be extracted and processed as defined in Section 3.3, before the rest of the message can be processed.

Figure 3 shows an example of a CoAP message transported over UDP and containing both the EDHOC data and the OSCORE ciphertext, using the newly defined EDHOC option for signalling.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Ver| T |  TKL  |      Code     |          Message ID           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Token (if any, TKL bytes) ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Observe Option| OSCORE Option ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| EDHOC Option  | Other Options (if any) ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|1 1 1 1 1 1 1 1| Payload ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 3: Example of CoAP message transported over UDP, combining EDHOC data and OSCORE data as signalled with the EDHOC Option.

3.2. Client Processing

This section describes the processing on the client side.

3.2.1. Processing of the EDHOC + OSCORE Request

The client prepares an EDHOC + OSCORE request as follows.

1. Compose EDHOC message_3 into EDHOC_MSG_3, as per Section 5.4.2 of [RFC9528].

2. Establish the new OSCORE Security Context and use it to encrypt the original CoAP request as per Section 8.1 of [RFC8613].

Note that the OSCORE ciphertext is not computed over EDHOC message_3, which is not protected by OSCORE. That is, the result of this step is the OSCORE Request as in Figure 1.

3. Build COMB_PAYLOAD as the concatenation of EDHOC_MSG_3 and OSCORE_PAYLOAD in this order: COMB_PAYLOAD = EDHOC_MSG_3 | OSCORE_PAYLOAD, where | denotes byte string concatenation and:

   * EDHOC_MSG_3 is the binary encoding of EDHOC message_3 resulting from step 1. As per Section 5.4.1 of [RFC9528], EDHOC message_3 consists of one CBOR data item CIPHERTEXT_3, which is a CBOR byte string. Therefore, EDHOC_MSG_3 is the binary encoding of CIPHERTEXT_3.

   * OSCORE_PAYLOAD is the OSCORE ciphertext of the OSCORE-protected CoAP request resulting from step 2.

4. Compose the EDHOC + OSCORE request, as the OSCORE-protected CoAP request resulting from step 2, where the payload is replaced with COMB_PAYLOAD built at step 3.

   Note that the new payload includes EDHOC message_3, but it does not include the EDHOC connection identifier C_R. As the client is the EDHOC Initiator, C_R is the OSCORE Sender ID of the client, which is already specified as 'kid' in the OSCORE Option of the request from step 2, hence of the EDHOC + OSCORE request.

5. Include the new EDHOC Option defined in Section 3.1 into the EDHOC + OSCORE request.

   The application/cid-edhoc+cbor-seq media type does not apply to this message, whose media type is unnamed.

6. Send the EDHOC + OSCORE request to the server.

With the same server, the client SHOULD NOT have multiple simultaneous outstanding interactions (see Section 4.7 of [RFC7252]) such that: they consist of an EDHOC + OSCORE request; and their EDHOC data pertain to the EDHOC session with the same connection identifier C_R.

(An exception might apply for clients that operate under particular time constraints over particularly unreliable networks, thus raising the chances to promptly complete the EDHOC execution with the server through multiple, simultaneous EDHOC + OSCORE requests. As discussed in Section 7, this does not have any impact in terms of security.)

3.2.2. Supporting Block-wise

If Block-wise [RFC7959] is supported, the client may fragment the first application CoAP request before protecting it as an original message with OSCORE, as defined in Section 4.1.3.4.1 of [RFC8613].

In such a case, the OSCORE processing in step 2 of Section 3.2.1 is performed on each inner block of the first application CoAP request, and the following also applies.

* The client takes the additional following step between steps 2 and 3 of Section 3.2.1.

  A. If the OSCORE-protected request from step 2 conveys a non-first inner block of the first application CoAP request (i.e., the Block1 Option processed at step 2 had NUM different than 0), then the client skips the following steps and sends the OSCORE-protected request to the server. In particular, the client MUST NOT include the EDHOC Option in the OSCORE-protected request.

* The client takes the additional following step between steps 3 and 4 of Section 3.2.1.

  B. If the size of COMB_PAYLOAD exceeds MAX_UNFRAGMENTED_SIZE (see Section 4.1.3.4.2 of [RFC8613]), the client MUST stop processing the request and MUST abandon the Block-wise transfer. Then, the client can continue by switching to the sequential workflow shown in Figure 1. That is, the client first sends EDHOC message_3 prepended by the EDHOC Connection Identifier C_R encoded as per Section 3.3 of [RFC9528], and then sends the OSCORE-protected CoAP request once the EDHOC execution is completed.

The performance advantage of using the EDHOC + OSCORE request can be lost when used in combination with Block-wise transfers that rely on specific parameter values and block sizes. Application policies at the CoAP client can define when and how to detect whether the performance advantage is lost, and, if that is the case, whether to appropriately adjust the parameter values and block sizes, or instead to fall back on the sequential workflow of EDHOC.

3.3. Server Processing

This section describes the processing on the server side.

3.3.1. Processing of the EDHOC + OSCORE Request

In order to process a request containing the EDHOC option, i.e., an EDHOC + OSCORE request, the server MUST perform the following steps.

1. Check that the EDHOC + OSCORE request includes the OSCORE option and that the request payload has the format defined at step 3 of Section 3.2.1 for COMB_PAYLOAD. If this is not the case, the server MUST stop processing the request and MUST reply with a 4.00 (Bad Request) error response.

2. Extract EDHOC message_3 from the payload COMB_PAYLOAD of the EDHOC + OSCORE request, as the first element EDHOC_MSG_3 (see step 3 of Section 3.2.1).

3. Take the value of 'kid' from the OSCORE option of the EDHOC + OSCORE request (i.e., the OSCORE Sender ID of the client), and use it as the EDHOC connection identifier C_R.

4. Retrieve the correct EDHOC session by using the connection identifier C_R from step 3.

   If the application profile used in the EDHOC session specifies that EDHOC message_4 shall be sent, the server MUST stop the EDHOC processing and consider it failed, as due to a client error.

   Otherwise, perform the EDHOC processing on the EDHOC message_3 extracted at step 2 as per Section 5.4.3 of [RFC9528], based on the protocol state of the retrieved EDHOC session.

   The application profile used in the EDHOC session is the same one associated with the EDHOC resource where the server received the request conveying EDHOC message_1 that started the session. This is relevant in case the server provides multiple EDHOC resources, which may generally refer to different application profiles.

5. Establish a new OSCORE Security Context associated with the client as per Appendix A.1 of [RFC9528], using the EDHOC output from step 4.

6. Extract the OSCORE ciphertext from the payload COMB_PAYLOAD of the EDHOC + OSCORE request, as the second element OSCORE_PAYLOAD (see step 3 of Section 3.2.1).

7. Rebuild the OSCORE-protected CoAP request, as the EDHOC + OSCORE request where the payload is replaced with the OSCORE ciphertext extracted at step 6. Then, remove the EDHOC option.

8. Decrypt and verify the OSCORE-protected CoAP request rebuilt at step 7, as per Section 8.2 of [RFC8613], by using the OSCORE Security Context established at step 5.

When the decrypted request is checked for any critical CoAP options (as it is during regular CoAP processing), the presence of an EDHOC option MUST be regarded as an unprocessed critical option, unless it is processed by some further mechanism.

9. Deliver the CoAP request resulting from step 8 to the application.

If steps 4 (EDHOC processing) and 8 (OSCORE processing) are both successfully completed, the server MUST reply with an OSCORE-protected response (see Section 5.4.3 of [RFC9528]). The usage of EDHOC message_4 as defined in Section 5.5 of [RFC9528] is not applicable to the approach defined in this document.

If step 4 (EDHOC processing) fails, the server aborts the session as per Section 5.4.3 of [RFC9528] and responds with an EDHOC error message with error code 1, formatted as defined in Section 6.2 of [RFC9528]. The server MUST NOT establish a new OSCORE Security Context from the present EDHOC session with the client. The CoAP response conveying the EDHOC error message is not protected with OSCORE. As per Section 9.5 of [RFC9528], the server has to make sure that the error message does not reveal sensitive information. The CoAP response conveying the EDHOC error message MUST have Content-Format set to application/edhoc+cbor-seq registered in Section 10.9 of [RFC9528].

If step 4 (EDHOC processing) is successfully completed but step 8 (OSCORE processing) fails, the same OSCORE error handling as defined in Section 8.2 of [RFC8613] applies.

3.3.2. Supporting Block-wise

If Block-wise [RFC7959] is supported, the server takes the additional following step before any other in Section 3.3.1.

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 of [RFC8613]).

3.4. Example of EDHOC + OSCORE Request

Figure 4 shows an example of EDHOC + OSCORE Request transported over UDP. In particular, the example assumes that:

* The OSCORE Partial IV in use is 0, consistently with the first request protected with the new OSCORE Security Context.

* The OSCORE Sender ID of the client is 0x01.

As per Section 3.3.3 of [RFC9528], this straightforwardly corresponds to the EDHOC connection identifier C_R 0x01.

As per Section 3.3.2 of [RFC9528], when using the sequential flow shown in Figure 1, the same C_R with value 0x01 would be encoded on the wire as the CBOR integer 1 (0x01 in CBOR encoding), and prepended to EDHOC message_3 in the payload of the second EDHOC request.

* The EDHOC option is registered with CoAP option number 21.

Note to RFC Editor: Please delete the last bullet point in the previous list, since, at the time of publication, the CoAP option number will be in fact registered.

This results in the following components shown in Figure 4:

* OSCORE option value: 0x090001 (3 bytes)

* EDHOC option value: - (0 bytes)

* EDHOC message_3: 0x52d5535f3147e85f1cfacd9e78abf9e0a81bbf (19 bytes)

* OSCORE ciphertext: 0x612f1092f1776f1c1668b3825e (13 bytes)

```
Protected CoAP request (OSCORE message):

   0x44025d1f               ; CoAP 4-byte header
     00003974               ; Token
     93 090001              ; OSCORE Option
     c0                     ; EDHOC Option
     ff 52d5535f3147e85f1cfacd9e78abf9e0a81bbf
        612f1092f1776f1c1668b3825e
   (46 bytes)
```

Figure 4: Example of CoAP message transported over UDP, combining EDHOC data and OSCORE data as signalled with the EDHOC Option.

4. Use of EDHOC Connection Identifiers with OSCORE

The OSCORE Sender/Recipient IDs are the EDHOC connection identifiers (see Section 3.3.3 of [RFC9528]). This applies also to the optimized workflow defined in Section 3 of this document.

Note that, at step 3 of Section 3.3.1, the value of 'kid' in the OSCORE Option of the EDHOC + OSCORE request is both the server's Recipient ID (i.e., the client's Sender ID) and the EDHOC Connection Identifier C_R of the server.

4.1. Additional Processing of EDHOC Messages

When using EDHOC to establish an OSCORE Security Context, the client and server MUST perform the following additional steps during an EDHOC execution, thus extending Section 5 of [RFC9528].

4.1.1. Initiator Processing of Message 1

The Initiator selects an EDHOC Connection Identifier C_I as follows.

The Initiator MUST choose a C_I that is neither used in any current EDHOC session as this peer's EDHOC Connection Identifier, nor the Recipient ID in a current OSCORE Security Context where the ID Context is not present.

The chosen C_I SHOULD NOT be the Recipient ID of any current OSCORE Security Context. Note that, unless the two peers concurrently use alternative methods to establish OSCORE Security Contexts, this allows the Responder to always omit the 'kid context' in the OSCORE Option of its messages sent to the Initiator, when protecting those with an OSCORE Security Context where C_I is the Responder's OSCORE Sender ID (see Section 6.1 of [RFC8613]).

4.1.2. Responder Processing of Message 2

The Responder selects an EDHOC Connection Identifier C_R as follows.

The Responder MUST choose a C_R that is neither used in any current EDHOC session as this peer's EDHOC Connection Identifier, nor is equal to the EDHOC Connection Identifier C_I specified in the EDHOC message_1 of the present EDHOC session, nor is the Recipient ID in a current OSCORE Security Context where the ID Context is not present.

The chosen C_R SHOULD NOT be the Recipient ID of any current OSCORE Security Context. Note that, for a reason analogous to the one given above with C_I, this allows the Initiator to always omit the 'kid context' in the OSCORE Option of its messages sent to the Responder, when protecting those with an OSCORE Security Context where C_R is the Initiator's OSCORE Sender ID (see Section 6.1 of [RFC8613]).

4.1.3. Initiator Processing of Message 2

If the EDHOC Connection Identifier C_I is equal to the EDHOC Connection Identifier C_R specified in EDHOC message_2, then the Initiator MUST abort the session and reply with an EDHOC error message with error code 1, formatted as defined in Section 6.2 of [RFC9528].

5. Extension and Consistency of Application Profiles

It is possible to include the information below in the application profile referred by the client and server, according to the specified consistency rules.

If the server supports the EDHOC + OSCORE request within an EDHOC execution started at a certain EDHOC resource, then the application profile associated with that resource SHOULD explicitly specify support for the EDHOC + OSCORE request.

In case the application profile indicates that the server supports the optional EDHOC message_4 (see Section 5.5 of [RFC9528]), it is still possible to use the optimized workflow based on the EDHOC + OSCORE request. However, this means the server is not going to send EDHOC message_4, since it is not applicable to the optimized workflow (see Section 3.3.1).

Also, in case the application profile indicates that the server shall send EDHOC message_4, then the application profile MUST NOT specify support for the EDHOC + OSCORE request, and there is no point for the client to use the optimized workflow, which is bound to fail (see Section 3.3.1).

6. Web Linking

Section 10.10 of [RFC9528] registers the resource type "core.edhoc", which can be used as target attribute in a web link [RFC8288] to an EDHOC resource, e.g., using a link-format document [RFC6690]. This enables clients to discover the presence of EDHOC resources at a server, possibly using the resource type as filter criterion.

At the same time, the application profile associated with an EDHOC resource provides information describing how the EDHOC protocol can be used through that resource. A client may become aware of the application profile, e.g., by obtaining its information elements upon discovering the EDHOC resources at the server. This allows the client to discover especially the EDHOC resources whose associated application profile denotes a way of using EDHOC which is most suitable to the client, e.g., with EDHOC cipher suites or authentication methods that the client also supports or prefers.

That is, while discovering an EDHOC resource, a client can contextually obtain relevant pieces of information from the application profile associated with that resource. The resource discovery can occur by means of a direct interaction with the server, or instead by means of the CoRE Resource Directory [RFC9176], where the server may have registered the links to its resources.

In order to enable the above, this section defines a number of parameters, each of which can be optionally specified as a target attribute with the same name in the link to the respective EDHOC resource, or as filter criteria in a discovery request from the client. When specifying these parameters in a link to an EDHOC resource, the target attribute rt="core.edhoc" MUST be included, and the same consistency rules defined in Section 5 for the corresponding information elements of an application profile MUST be followed.

The following parameters are defined.

* 'ed-i', specifying, if present, that the server supports the EDHOC Initiator role, hence the reverse message flow of EDHOC. A value MUST NOT be given to this parameter and any present value MUST be ignored by the recipient.

* 'ed-r', specifying, if present, that the server supports the EDHOC Responder role, hence the forward message flow of EDHOC. A value MUST NOT be given to this parameter and any present value MUST be ignored by the recipient.

* 'ed-method', specifying an authentication method supported by the server. This parameter MUST specify a single value, which is taken from the 'Value' column of the "EDHOC Method Type" registry defined in Section 10.3 of [RFC9528]. This parameter MAY occur multiple times, with each occurrence specifying an authentication method.

* 'ed-csuite', specifying an EDHOC cipher suite supported by the server. This parameter MUST specify a single value, which is taken from the 'Value' column of the "EDHOC Cipher Suites"

registry defined in Section 10.2 of [RFC9528]. This parameter MAY occur multiple times, with each occurrence specifying a cipher suite.

* 'ed-cred-t', specifying a type of authentication credential supported by the server. This parameter MUST specify a single value, which is taken from the 'Value' column of the "EDHOC Authentication Credential Types" Registry defined in Section 8.3 of this document. This parameter MAY occur multiple times, with each occurrence specifying a type of authentication credential.

* 'ed-idcred-t', specifying a type of identifier supported by the server for identifying authentication credentials. This parameter MUST specify a single value, which is taken from the 'Label' column of the "COSE Header Parameters" registry [COSE.Header.Parameters]. This parameter MAY occur multiple times, with each occurrence specifying a type of identifier for authentication credentials.

  Note that the values in the 'Label' column of the "COSE Header Parameters" registry are strongly typed. On the contrary, Link Format is weakly typed and thus does not distinguish between, for instance, the string value "-10" and the integer value -10. Thus, if responses in Link Format are returned, string values which look like an integer are not supported. Therefore, such values MUST NOT be used in the 'ed-idcred-t' parameter.

* 'ed-ead', specifying the support of the server for an External Authorization Data (EAD) item (see Section 3.8 of [RFC9528]). This parameter MUST specify a single value, which is taken from the 'Label' column of the "EDHOC External Authorization Data" registry defined in Section 10.5 of [RFC9528]. This parameter MAY occur multiple times, with each occurrence specifying the ead_label of an EAD item that the server supports.

* 'ed-comb-req', specifying, if present, that the server supports the EDHOC + OSCORE request defined in Section 3. A value MUST NOT be given to this parameter and any present value MUST be ignored by the recipient.

(Future documents may update the definition of the parameters 'ed-i', 'ed-r', and 'ed-comb-req', by expanding their semantics and specifying what they can take as value.)

The example in Figure 5 shows how a client discovers one EDHOC resource at a server, obtaining information elements from the respective application profile. The Link Format notation from Section 5 of [RFC6690] is used.

```
REQ: GET /.well-known/core

RES: 2.05 Content
    </sensors/temp>;osc,
    </sensors/light>;if=sensor,
    </.well-known/edhoc>;rt=core.edhoc;ed-csuite=0;ed-csuite=2;
        ed-method=0;ed-cred-t=1;ed-cred-t=3;ed-idcred-t=4;
        ed-i;ed-r;ed-comb-req
```

Figure 5: The Web Link.

7. Security Considerations

The same security considerations from OSCORE [RFC8613] and EDHOC [RFC9528] hold for this document. In addition, the following considerations also apply.

Section 3.2.1 specifies that a client SHOULD NOT have multiple outstanding EDHOC + OSCORE requests pertaining to the same EDHOC session. Even if a client did not fulfill this requirement, it would not have any impact in terms of security. That is, the server would still not process different instances of the same EDHOC message_3 more than once in the same EDHOC session (see Section 5.1 of [RFC9528]), and would still enforce replay protection of the OSCORE-protected request (see Sections 7.4 and 8.2 of [RFC8613]).

When using the optimized workflow in Figure 2, a minimum of 128-bit security against online brute force attacks is achieved after the client receives and successfully verifies the first OSCORE-protected response (see Sections 9.1 and 9.4 of [RFC9528]). As an example, if EDHOC is used with method 3 (see Section 3.2 of [RFC9528]) and cipher suite 2 (see Section 3.6 of [RFC9528]), then the following holds.

* The Initiator is authenticated with 128-bit security against online attacks. As per Section 9.1 of [RFC9528], this results from the combination of the strength of the 64-bit MAC in EDHOC message_3 and of the 64-bit MAC in the AEAD of the first OSCORE-protected CoAP request, as rebuilt at step 7 of Section 3.3.1.

* The Responder is authenticated with 128-bit security against online attacks. As per Section 9.1 of [RFC9528], this results from the combination of the strength of the 64-bit MAC in EDHOC message_2 and of the 64-bit MAC in the AEAD of the first OSCORE-protected CoAP response.

With reference to the sequential workflow in Figure 1, the OSCORE request might have to undergo access control checks at the server, before being actually executed for accessing the target protected resource. The same MUST hold when the optimized workflow in Figure 2 is used, i.e., when using the EDHOC + OSCORE request.

That is, the rebuilt OSCORE-protected application request from step 7 in Section 3.3.1 MUST undergo the same access control checks that would be performed on a traditional OSCORE-protected application request sent individually as shown in Figure 1.

To this end, validated information to perform access control checks (e.g., an access token issued by a trusted party) has to be available at the server before starting to process the rebuilt OSCORE-protected application request. Such information may have been provided to the server separately before starting the EDHOC execution altogether, or instead as External Authorization Data during the EDHOC execution (see Section 3.8 of [RFC9528]).

Thus, a successful completion of the EDHOC protocol and the following derivation of the OSCORE Security Context at the server do not play a role in determining whether the rebuilt OSCORE-protected request is authorized to access the target protected resource at the server.

8. IANA Considerations

This document has the following actions for IANA.

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

8.1. CoAP Option Numbers Registry

IANA is asked to enter the following option number to the "CoAP Option Numbers" registry within the "CoRE Parameters" registry group.

Number	Name	Reference
21	EDHOC	[RFC-XXXX]

Table 2: Registrations in CoAP Option Numbers Registry

Note to RFC Editor: Please delete this paragraph and all the following text within the present Section 8.1.

[

The CoAP option number 21 is consistent with the properties of the EDHOC Option defined in Section 3.1, and it allows the EDHOC Option to always result in an overall size of 1 byte. This is because:

* The EDHOC option is always empty, i.e., with zero-length value; and

* Since the OSCORE Option with option number 9 is always present in the EDHOC + OSCORE request, the EDHOC Option is encoded with a delta equal to at most 12.

Although the currently unassigned option number 13 would also work well for the same reasons in the use case in question, different use cases or protocols may make a better use of the option number 13. Hence the preference for the option number 21, and why it is _not_ necessary to register additional option numbers than 21.

]

8.2. Target Attributes Registry

IANA is asked to register the following entries in the "Target Attributes" registry [CORE.Target.Attributes] within the "Constrained RESTful Environments (CoRE) Parameters" registry group, as per [I-D.ietf-core-target-attr]. For all entries, the Change Controller is IETF, and the reference is [RFC-XXXX].

Attribute Name:	Brief Description:
ed-i	Hint: support for the EDHOC Initiator role
ed-r	Hint: support for the EDHOC Responder role
ed-method	A supported authentication method for EDHOC
ed-csuite	A supported cipher suite for EDHOC
ed-cred-t	A supported type of authentication credential for EDHOC
ed-idcred-t	A supported type of authentication credential identifier for EDHOC
ed-ead	A supported External Authorization Data (EAD) item for EDHOC
ed-comb-req	Hint: support for the EDHOC+OSCORE request

Table 3: Registrations in Target Attributes Registry

8.3. EDHOC Authentication Credential Types Registry

IANA is requested to create a new "EDHOC Authentication Credential Types" registry within the "Ephemeral Diffie-Hellman Over COSE (EDHOC)" registry group defined in [RFC9528].

As registration policy, the registry uses either "Standards Action with Expert Review", or "Specification Required" per Section 4.6 of [RFC8126]. Expert Review guidelines are provided in Section 8.4.

All assignments according to "Standards Action with Expert Review" are made on a "Standards Action" basis per Section 4.9 of [RFC8126], with Expert Review additionally required per Section 4.5 of [RFC8126]. The procedure for early IANA allocation of Standards Track code points defined in [RFC7120] also applies. When such a procedure is used, review and approval by the designated expert are also required, in order for the WG chairs to determine that the conditions for early allocation are met (see step 2 in Section 3.1 of [RFC7120]).

The columns of this registry are:

* Value: This field contains the value used to identify the type of authentication credential. These values MUST be unique. The value can be an unsigned integer or a negative integer, in the range from -65536 to 65535. Different ranges of values use different registration policies [RFC8126]:

  - Integer values from -24 to 23 are designated as "Standards Action With Expert Review".

  - Integer values from -65536 to -25 and from 24 to 65535 are designated as "Specification Required".

  - Integer values smaller than -65536 and greater than 65535 are marked as "Private Use".

* Description: This field contains a short description of the type of authentication credential.

* Reference: This field contains a pointer to the public specification for the type of authentication credential.

Initial entries in this registry are as listed in Table 4.

Value	Description	Reference
0	CBOR Web Token (CWT) containing a COSE_Key in a 'cnf' claim and possibly other claims. CWT is defined in RFC 8392.	[RFC8392]
1	CWT Claims Set (CCS) containing a COSE_Key in a 'cnf' claim and possibly other claims. CCS is defined in RFC 8392.	[RFC8392]
2	X.509 certificate	[RFC5280]

Table 4: Initial Entries in the "EDHOC Authentication Credential Types" Registry

8.4. Expert Review Instructions

The IANA registry established in this document is defined as "Standards Action with Expert Review" or "Specification Required", depending on the range of values for which an assignment is requested. 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:

* Clarity and correctness of registrations. Experts are expected to check the clarity of purpose and use of the requested entries. Experts need to make sure that registered identifiers indicate a type of authentication credential whose format and encoding is clearly defined in the corresponding specification. Identifiers of types of authentication credentials that do not meet these objective of clarity and completeness must not be registered.

* 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.

* Specifications are required for the "Standards Action With Expert Review" range of point assignment. Specifications should exist for "Specification Required" ranges, but early assignment before a specification is available is considered to be permissible. When specifications are not provided, the description provided needs to have sufficient information to identify what the point is being used for.

* 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.

9. References

9.1. Normative References

[CORE.Target.Attributes]
IANA, "Target Attributes", <https://www.iana.org/assignments/core-parameters/core-parameters.xhtml#target-attributes>.

[COSE.Header.Parameters]
IANA, "COSE Header Parameters", <https://www.iana.org/assignments/cose/cose.xhtml#header-parameters>.

[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/rfc/rfc2119>.

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <https://www.rfc-editor.org/rfc/rfc5280>.

[RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <https://www.rfc-editor.org/rfc/rfc6690>.

[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/rfc/rfc7120>.

[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/rfc/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/rfc/rfc7959>.

[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/rfc/rfc8126>.

[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/rfc/rfc8174>.

[RFC8288] Nottingham, M., "Web Linking", RFC 8288, DOI 10.17487/RFC8288, October 2017, <https://www.rfc-editor.org/rfc/rfc8288>.

[RFC8392] Jones, M., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "CBOR Web Token (CWT)", RFC 8392, DOI 10.17487/RFC8392, May 2018, <https://www.rfc-editor.org/rfc/rfc8392>.

[RFC8613] Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security for Constrained RESTful Environments (OSCORE)", RFC 8613, DOI 10.17487/RFC8613, July 2019, <https://www.rfc-editor.org/rfc/rfc8613>.

[RFC8949] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", STD 94, RFC 8949, DOI 10.17487/RFC8949, December 2020, <https://www.rfc-editor.org/rfc/rfc8949>.

[RFC9176] Amsüss, C., Ed., Shelby, Z., Koster, M., Bormann, C., and P. van der Stok, "Constrained RESTful Environments (CoRE) Resource Directory", RFC 9176, DOI 10.17487/RFC9176, April 2022, <https://www.rfc-editor.org/rfc/rfc9176>.

[RFC9528] Selander, G., Preuß Mattsson, J., and F. Palombini, "Ephemeral Diffie-Hellman Over COSE (EDHOC)", RFC 9528, DOI 10.17487/RFC9528, March 2024, <https://www.rfc-editor.org/rfc/rfc9528>.

9.2. Informative References

[I-D.ietf-core-target-attr] Bormann, C., "CoRE Target Attributes Registry", Work in Progress, Internet-Draft, draft-ietf-core-target-attr-06, 11 October 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-core-target-attr-06>.

Appendix A. Document Updates

This section is to be removed before publishing as an RFC.

A.1. Version -10 to -11

* Avoid using quotation marks for CBOR Simple Values.

* Early mentioning of the optimization properties.

* Less entries of new IANA registry; made their references normative.

* Clarified meaning of "Standards Action with Expert Review" policy.

* Clarifications, simplified phrasing, and editorial improvements.

* Updated references.

A.2. Version -09 to -10

* Expanded acronyms in the document title.

* Clarified transport of EDHOC C_R and EDHOC message_3.

* Simplified text on the use of EDHOC Connection Identifiers as OSCORE Identifiers.

* Added the CoAP OSCORE Option in the figures of the EDHOC message flows.

* Added more pointers to the message processing, now defined in dedicated subsections.

* Detecting and preventing a loss of performance advantage when using Block-wise transfers is for application policies to specifiy.

* Clarified use of EDHOC application profiles.

* Clarified security considerations on the achieved security level.

* Fixes and editorial improvements.

A.3. Version -08 to -09

* Clarified meaning of "EDHOC data".

* Improved description of entries for the new IANA registry.

* Change Controller changed from "IESG" to "IETF".

* Editorial: EDHOC Option number denoted as "21" instead of "TBD21".

* Fixed references to sections of draft-ietf-lake-edhoc

* Clarifications and editorial improvements.

A.4. Version -07 to -08

* Fixes and clarifications from the Shepherd’s review.

A.5. Version -06 to -07

* Changed document title.

* The client creates the OSCORE Security Context after creating EDHOC message_3.

* Revised selection of EDHOC connection identifiers.

* Use of "forward message flow" and "reverse message flow".

* The payload of the combined request is not a CBOR sequence anymore.

* EDHOC error messages from the server are not protected with OSCORE.

* More future-proof error handling on the server side.

* Target attribute names prefixed by "ed-".

* Defined new target attributes "ed-i" and "ed-r".

* Defined single target attribute "ed-ead" signaling supported EAD items.

* Security consideration on the minimally achieved 128-bit security.

* Defined and used the "EDHOC Authentication Credential Types" Registry.

* High-level sentence replacing the appendix on Block-wise performance.

* Revised examples.

* Editorial improvements.

A.6. Version -05 to -06

* Extended figure on EDHOC sequential workflow.

* Revised naming of target attributes.

* Clarified semantics of target attributes 'eadx'.

* Registration of target attributes.

A.7. Version -04 to -05

* Clarifications on Web Linking parameters.

* Added security considerations.

* Revised IANA considerations to focus on the CoAP option number 21.

* Guidelines on using Block-wise moved to an appendix.

* Editorial improvements.

A.8. Version -03 to -04

* Renamed "applicability statement" to "application profile".

* Use the latest Content-Formats.

* Use of SHOULD NOT for multiple simultaneous outstanding interactions.

* No more special conversion from OSCORE ID to EDHOC ID.

* Considerations on using Block-wise.

* Wed Linking signaling of multiple supported EAD labels.

* Added security considerations.

* Editorial improvements.

A.9. Version -02 to -03

* Clarifications on transporting EDHOC message_3 in the CoAP payload.

* At most one simultaneous outstanding interaction as an EDHOC + OSCORE request with the same server for the same session with connection identifier C_R.

* The EDHOC option is removed from the EDHOC + OSCORE request after processing the EDHOC data.

* Added explicit constraints when selecting a Recipient ID as C_X.

* Added processing steps for when Block-wise is used.

* Improved error handling on the server.

* Improved section on Web Linking.

* Updated figures; editorial improvements.

A.10. Version -01 to -02

* New title, abstract and introduction.
* Restructured table of content.
* Alignment with latest format of EDHOC messages.
* Guideline on ID conversions based on application profile.
* Clarifications, extension and consistency on application profile.
* Section on web-linking.
* RFC8126 terminology in IANA considerations.
* Revised Appendix "Checking CBOR Encoding of Numeric Values".

A.11. Version -00 to -01

* Improved background overview of EDHOC.
* Added explicit rules for converting OSCORE Sender/Recipient IDs to EDHOC connection identifiers following the removal of bstr_identifier from EDHOC.
* Revised section organization.
* Recommended number for EDHOC option changed to 21.
* Editorial improvements.

Acknowledgments

The authors sincerely thank Christian Amsüss, Emmanuel Baccelli, Carsten Bormann, Roman Danyliw, Esko Dijk, Joel Halpern, Wes Hardaker, Klaus Hartke, John Preuß Mattsson, David Navarro, Shuping Peng, Jim Schaad, Jürgen Schönwälder, John Scudder, Orie Steele, Gunter Van de Velde, Malia Vuini, and Paul Wouters 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 H2020 project SIFIS-Home (Grant agreement 952652).

Authors' Addresses

Francesca Palombini
Ericsson
Email: francesca.palombini@ericsson.com

Marco Tiloca
RISE AB
Isafjordsgatan 22
SE-16440 Stockholm Kista
Sweden
Email: marco.tiloca@ri.se

Rikard Höglund
RISE AB
Isafjordsgatan 22
SE-16440 Stockholm Kista
Sweden
Email: rikard.hoglund@ri.se

Stefan Hristozov
Fraunhofer AISEC
Email: stefan.hristozov@eriptic.com

Göran Selander
Ericsson
Email: goran.selander@ericsson.com

CoRE Working Group
Internet-Draft
Intended status: Standards Track
Expires: 5 September 2024

M. Tiloca
RISE AB
G. Selander
F. Palombini
J. Mattsson
Ericsson AB
R. Höglund
RISE AB

4 March 2024

Group Object Security for Constrained RESTful Environments (Group OSCORE)
draft-ietf-core-oscore-groupcomm-21

Abstract

This document defines the security protocol Group Object Security for Constrained RESTful Environments (Group OSCORE), providing end-to-end security of CoAP messages exchanged between members of a group, e.g., sent over IP multicast. In particular, the described protocol defines how OSCORE is used in a group communication setting to provide source authentication for CoAP group requests, sent by a client to multiple servers, and for protection of the corresponding CoAP responses. Group OSCORE also defines a pairwise mode where each member of the group can efficiently derive a symmetric pairwise key with any other member of the group for pairwise OSCORE communication. Group OSCORE can be used between endpoints communicating with CoAP or CoAP-mappable HTTP.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction 5
1.1. Terminology 7
2. Security Context 9
2.1. Common Context 11
2.1.1. AEAD Algorithm 11
2.1.2. ID Context 11
2.1.3. Common IV 12
2.1.4. Authentication Credential Format 12
2.1.5. Group Manager Authentication Credential 12
2.1.6. Group Encryption Algorithm 12
2.1.7. Signature Algorithm 12
2.1.8. Signature Encryption Key 13
2.1.9. Pairwise Key Agreement Algorithm 13
2.2. Sender Context and Recipient Context 14
2.3. Establishment of Security Context Parameters 15
2.4. Authentication Credentials 15
2.5. Pairwise Keys 17
2.5.1. Derivation of Pairwise Keys 17
2.5.2. ECDH with Montgomery Coordinates 19
2.5.3. Usage of Sequence Numbers 20
2.5.4. Security Context for Pairwise Mode 21
2.6. Update of Security Context 21
2.6.1. Loss of Mutable Security Context 22
2.6.2. Exhaustion of Sender Sequence Number Space 24
2.6.3. Retrieving New Security Context Parameters 24
3. The Group Manager 27
3.1. Set-up of New Endpoints 27
3.2. Management of Group Keying Material 29
3.2.1. Recycling of Identifiers 32
3.3. Support for Signature Checkers 35
3.4. Responsibilities of the Group Manager 35
4. The COSE Object 37

4.1. Countersignature 37
4.1.1. Clarifications on Using a Countersignature 37
4.2. The 'kid' and 'kid context' parameters 38
4.3. AEAD Nonce 38
4.4. external_aad 38
5. OSCORE Header Compression 41
5.1. Keystream Derivation for Countersignature Encryption . . 41
5.2. Examples of Compressed COSE Objects 43
5.2.1. Examples in Group Mode 43
5.2.2. Examples in Pairwise Mode 44
6. Message Binding, Sequence Numbers, Freshness, and Replay Protection 45
6.1. Supporting Multiple Responses in Long Exchanges 46
6.2. Synchronization and Freshness 46
6.3. Replay Protection 47
6.3.1. Replay Protection of Responses 47
7. Message Reception 49
8. Message Processing in Group Mode 50
8.1. Protecting the Request 51
8.2. Verifying the Request 52
8.3. Protecting the Response 55
8.4. Verifying the Response 57
8.5. External Signature Checkers 60
9. Message Processing in Pairwise Mode 61
9.1. Pre-Conditions 62
9.2. Main Differences from OSCORE 63
9.3. Protecting the Request 63
9.4. Verifying the Request 63
9.5. Protecting the Response 64
9.6. Verifying the Response 65
10. Challenge-Response Based Freshness and Synchronization . . . 66
11. Implementation Compliance 69
12. Web Linking 70
13. Security Considerations 71
13.1. Security of the Group Mode 72
13.2. Security of the Pairwise Mode 74
13.3. Uniqueness of (key, nonce) 75
13.4. Management of Group Keying Material 76
13.5. Update of Security Context and Key Rotation 76
13.5.1. Late Update on the Sender 77
13.5.2. Late Update on the Recipient 77
13.6. Collision of Group Identifiers 78
13.7. Cross-group Message Injection 78
13.7.1. Attack Description 78
13.7.2. Attack Prevention in Group Mode 79
13.8. Prevention of Group Cloning Attack 81
13.9. Group OSCORE for Unicast Requests 81
13.10. End-to-end Protection 82

13.11. Master Secret 83
13.12. Replay Protection 83
13.13. Message Ordering 84
13.14. Message Freshness 84
13.15. Client Aliveness 85
13.16. Cryptographic Considerations 85
13.17. Message Segmentation 87
13.18. Privacy Considerations 87
14. IANA Considerations 88
14.1. OSCORE Flag Bits Registry 88
14.2. Target Attributes Registry 89
15. References 89
15.1. Normative References 89
15.2. Informative References 91
Appendix A. Assumptions and Security Objectives 94
A.1. Assumptions 95
A.2. Security Objectives 96
Appendix B. List of Use Cases 97
Appendix C. Example of Group Identifier Format 99
Appendix D. Document Updates 100
D.1. Version -20 to -21 100
D.2. Version -19 to -20 101
D.3. Version -18 to -19 101
D.4. Version -17 to -18 101
D.5. Version -16 to -17 102
D.6. Version -15 to -16 102
D.7. Version -14 to -15 102
D.8. Version -13 to -14 102
D.9. Version -12 to -13 103
D.10. Version -11 to -12 103
D.11. Version -10 to -11 104
D.12. Version -09 to -10 105
D.13. Version -08 to -09 106
D.14. Version -07 to -08 106
D.15. Version -06 to -07 108
D.16. Version -05 to -06 108
D.17. Version -04 to -05 109
D.18. Version -03 to -04 109
D.19. Version -02 to -03 110
D.20. Version -01 to -02 111
D.21. Version -00 to -01 112
Acknowledgments 113
Authors' Addresses 113

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 [I-D.ietf-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 utilization. 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). Group communication for CoAP [I-D.ietf-core-groupcomm-bis] mainly uses UDP/IP multicast as the underlying data transport.

Object Security for Constrained RESTful Environments (OSCORE) [RFC8613] describes a security protocol based on the exchange of protected CoAP messages. OSCORE builds on CBOR Object Signing and Encryption (COSE) [RFC9052][RFC9053] and provides end-to-end encryption, integrity, replay protection, and binding of response to request between a sender and a recipient, independent of the transport layer 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 into a COSE object, which is conveyed within the CoAP payload and the CoAP OSCORE option of the protected message, thereby replacing those message fields with an authenticated and encrypted object.

This document defines Group OSCORE, a security protocol for group communication with CoAP [I-D.ietf-core-groupcomm-bis] and for CoAP-mappable HTTP requests and responses, providing the same end-to-end security properties as OSCORE also in the case where requests have multiple recipients. In particular, the described protocol defines how OSCORE is used in a group communication setting to provide source authentication for group requests sent by a client to multiple servers, and for protection of the corresponding responses. Group OSCORE also defines a pairwise mode where each member of the group can efficiently derive a symmetric pairwise key with any other member of the group for pairwise-protected OSCORE communication. Just like OSCORE, Group OSCORE is independent of the transport layer and works wherever CoAP does.

As with OSCORE, it is possible to combine Group OSCORE with communication security on other layers. One example is the use of transport layer security, such as DTLS [RFC9147], between one client and one proxy (and vice versa), or between one proxy and one server (and vice versa). This prevents observers from accessing addressing information conveyed in CoAP options that would not be protected by

Group OSCORE, but would be protected by DTLS. These options include Uri-Host, Uri-Port, and Proxy-Uri. Note that DTLS does not define how to secure messages sent over IP multicast. Group OSCORE is also intended to work with OSCORE-capable proxies [I-D.tiloca-core-oscore-capable-proxies] thereby enabling, for example, nested OSCORE operations with OSCORE-protected communication between a CoAP client and a proxy, carrying messages that are additionally protected with Group OSCORE between the CoAP client and the target CoAP servers.

Group OSCORE defines two modes of operation, that can be used independently or together:

* In the group mode, Group OSCORE requests and responses are digitally signed with the private key of the sender and the signature is embedded in the protected CoAP message. The group mode supports all COSE signature algorithms as well as signature verification by intermediaries. This mode is defined in Section 8.

* In the pairwise mode, two group members exchange OSCORE requests and responses (typically) over unicast, and the messages are protected with symmetric keys. These symmetric keys are derived from Diffie-Hellman shared secrets, calculated with the asymmetric keys of the sender and recipient, allowing for shorter integrity tags and therefore lower message overhead. This mode is defined in Section 9.

Both modes provide source authentication of CoAP messages. The application decides what mode to use, potentially on a per-message basis. Such decisions can be based, for instance, on pre-configured policies or dynamic assessing of the target recipient and/or resource, among other things. One important case is when requests are protected in group mode, and responses in pairwise mode. Since such responses convey shorter integrity tags instead of bigger, full-fledged signatures, this significantly reduces the message overhead in case of many responses to one request.

A special deployment of Group OSCORE consists in using the pairwise mode only. For example, consider the case of a constrained-node network [RFC7228] with a large number of CoAP endpoints and the objective to establish secure communication between any pair of endpoints with a small provisioning effort and message overhead. Since the total number of security associations that needs to be established grows with the square of the number of endpoints, it is desirable to restrict the amount of secret keying material provided to each endpoint. Moreover, a key establishment protocol would need to be executed for each security association. One solution to this

issue is to deploy Group OSCORE, with the endpoints being part of a group, and to use the pairwise mode. This solution has the benefit of providing a single shared secret, while distributing only the public keys of group members or a subset of those. After that, a CoAP endpoint can locally derive the OSCORE Security Context for the other endpoint in the group, and protect CoAP communications with very low overhead [I-D.ietf-iotops-security-protocol-comparison].

In some circumstances, Group OSCORE messages may be transported in HTTP, e.g., when they are protected with the pairwise mode and target a single recipient, or when they are protected with the group mode and target multiple CoAP recipients through cross-protocol translators such as HTTP-to-CoAP proxies [RFC8075][I-D.tiloca-core-groupcomm-proxy]. The use of Group OSCORE with HTTP is as defined for OSCORE in Section 11 of [RFC8613].

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 [I-D.ietf-core-groupcomm-bis]; Observe [RFC7641]; CBOR [RFC8949]; COSE [RFC9052][RFC9053] and related countersignatures [RFC9338].

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 [RFC8613].

Terminology for constrained environments, such as "constrained device" and "constrained-node network", is defined in [RFC7228].

This document refers also to the following terminology.

* Keying material: data that is necessary to establish and maintain secure communication among endpoints. This includes, for instance, keys and IVs [RFC4949].

* Authentication credential: information associated with an entity, including that entity's public key and parameters associated with the public key. Examples of authentication credentials are CBOR Web Tokens (CWTs) and CWT Claims Sets [RFC8392], X.509 certificates [RFC5280] and C509 certificates [I-D.ietf-cose-cbor-encoded-cert]. Further details about authentication credentials are provided in Section 2.4.

* Group: a set of endpoints that share group keying material and security parameters (Common Context, see Section 2). That is, unless otherwise specified, the term group used in this document refers to a "security group" (see Section 2.1 of [I-D.ietf-core-groupcomm-bis]), not to be confused with "CoAP group" or "application group".

* Group Manager: an entity responsible for a group, neither required to be an actual group member nor to take part in the group communication. The responsibilities of the Group Manager are listed in Section 3.4.

* Silent server: a member of a group that performs only group mode processing on incoming requests and never sends responses protected with Group OSCORE. For CoAP group communications, requests are normally sent without necessarily expecting a response. A silent server may send unprotected responses, as error responses reporting a Group OSCORE error.

* Group Identifier (Gid): identifier assigned to the group, unique within the set of groups of a given Group Manager.

* Birth Gid: with respect to a group member, the Gid obtained by that group member upon (re-)joining the group.

* Key Generation Number: an integer value identifying the current version of the keying material used in a group.

* 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.

* Group request: a CoAP request message sent by a client in the group to servers in that group.

* Long exchange: an exchange of messages associated with a request that is a group request and/or an Observe request [RFC7641].

In either case, multiple responses can follow from the same server to the request associated with the long exchange. The client terminates a long exchange when freeing up the CoAP Token value used for the associated request, for which no further responses will be accepted afterwards.

2. Security Context

This document refers to a group as a set of endpoints sharing keying material and security parameters for executing the Group OSCORE protocol, see Section 1.1. All members of a group maintain a Security Context as defined in this section.

How the Security Context is established by the group members is out of scope for this document, but if there is more than one Security Context applicable to a message, then the endpoints MUST be able to tell which Security Context was latest established. How to manage information about the group is described in terms of a Group Manager (see Section 3).

An endpoint of the group may use the group mode (see Section 8), the pairwise mode (see Section 9), or both, depending on the modes it supports and on the parameters of the Security Context. The Security Context of Group OSCORE extends the OSCORE Security Context defined in Section 3 of [RFC8613] as follows (see Figure 1).

* One Common Context, shared by all the endpoints in the group and extended as defined below.

  - The new parameter Authentication Credential Format, specifying the format of authentication credentials used in the group (see Section 2.1.4).

  - The new parameter Group Manager Authentication Credential, specifying the authentication credential of the Group Manager responsible for the group (see Section 2.1.5).

  - For the group mode, the Common Context is extended with the following new parameters.

    o Group Encryption Algorithm, used for encrypting and decrypting messages protected in group mode (see Section 2.1.6).

    o Signature Algorithm, used for computing and verifying the countersignature of messages protected in group mode (see Section 2.1.7).

    o Signature Encryption Key, used for encrypting and decrypting the countersignature of messages protected in group mode (see Section 2.1.8).

  - For the pairwise mode, the Common Context is extended with a Pairwise Key Agreement Algorithm (see Section 2.1.9) used for the agreement on a static-static Diffie-Hellman shared secret, from which pairwise keys are derived (see Section 2.5.1).

* One Sender Context, extended with the following new parameters.

  - The endpoint's own private key used to sign messages protected in group mode (see Section 8), or for deriving pairwise keys used with the pairwise mode (see Section 2.5).

  - The endpoint's own authentication credential containing its public key (see Section 2.4).

  - For the pairwise mode, the Sender Context is extended with the Pairwise Sender Keys associated with the other endpoints (see Section 2.5).

  If the endpoint is configured exclusively as silent server (see Section 1.1), then the Sender Context is omitted.

* One Recipient Context for each other endpoint from which messages are received. It is not necessary to maintain Recipient Contexts associated with endpoints from which messages are not (expected to be) received.

  - Each Recipient Context is extended with the authentication credential of the other endpoint, used to verify the signature of messages protected in group mode, or for deriving the pairwise keys used with the pairwise mode (see Section 2.5).

  - For the pairwise mode, each Recipient Context is extended with the Pairwise Recipient Key associated with the other endpoint (see Section 2.5).

Context Component	New Information Elements
Common Context	Authentication Credential Format Group Manager Authentication Credential * Group Encryption Algorithm * Signature Algorithm * Signature Encryption Key ^ Pairwise Key Agreement Algorithm
Sender Context	Endpoint's own private key Endpoint's own authentication credential ^ Pairwise Sender Keys for the other endpoints
Each Recipient Context	Other endpoint's authentication credential ^ Pairwise Recipient Key for the other endpoint

Figure 1: Additions to the OSCORE Security Context. The elements labeled with * and with ^ are relevant only for the group mode and only for the pairwise mode, respectively.

2.1. Common Context

The following sections specify how the Common Context is used and extended compared to [RFC8613]. All algorithms (AEAD, Group Encryption, Signature, Pairwise Key Agreement) are immutable once the Common Context is established. The Common Context may be acquired from the Group Manager (see Section 3).

2.1.1. AEAD Algorithm

The AEAD Algorithm (see Section 3.1 of [RFC8613]) SHALL identify the COSE AEAD algorithm to use for encryption and decryption when messages are protected using the pairwise mode (see Section 9). This algorithm MUST provide integrity protection. If this parameter is not set, the pairwise mode is not used in the group.

2.1.2. ID Context

The ID Context parameter (see Sections 3.1 and 3.3 of [RFC8613]) SHALL contain the Group Identifier (Gid) of the group. The choice of the Gid format is application specific. An example of specific formatting of the Gid is given in Appendix C. The application needs to specify how to handle potential collisions between Gids (see Section 13.6).

2.1.3. Common IV

The Common IV parameter (see Section 3.1 of [RFC8613]) SHALL identify the Common IV used in the group. Differently from OSCORE, the length of the Common IV is determined as follows.

* If only one among the AEAD Algorithm and the Group Encryption Algorithm is set (see Section 2.1.1 and Section 2.1.6), the length of the Common IV is the AEAD nonce length for the set algorithm.

* If both the AEAD Algorithm and the Group Encryption Algorithm are set, the length of the Common IV is the greatest AEAD nonce length among those of the two algorithms.

2.1.4. Authentication Credential Format

The new parameter Authentication Credential Format specifies the format of authentication credentials used in the group. Further details on authentication credentials are compiled in Section 2.4.

2.1.5. Group Manager Authentication Credential

The new parameter Group Manager Authentication Credential specifies the authentication credential of the Group Manager, including the Group Manager's public key. The endpoint MUST achieve proof-of-possession of the corresponding private key. Further details on the provisioning of the Group Manager's authentication credential to the group members are out of the scope of this document.

2.1.6. Group Encryption Algorithm

The new parameter Group Encryption Algorithm identifies the algorithm to use for encryption and decryption, when messages are protected in group mode (see Section 8). This algorithm MAY provide integrity protection. If this parameter is not set, the group mode is not used in the group.

2.1.7. Signature Algorithm

The new parameter Signature Algorithm identifies the digital signature algorithm used for computing and verifying the countersignature on the COSE object (see Sections 3.2 and 3.3 of [RFC9338]), when messages are protected in group mode (see Section 8). If this parameter is not set, the group mode is not used in the group.

2.1.8. Signature Encryption Key

The new parameter Signature Encryption Key specifies the encryption key for deriving a keystream to encrypt/decrypt a countersignature, when a message is protected in group mode (see Section 8).

The Signature Encryption Key is derived as defined for Sender/Recipient Keys in Section 3.2.1 of [RFC8613], with the following differences.

* The 'id' element of the 'info' array is the empty byte string.

* The 'alg_aead' element of the 'info' array specifies the Group Encryption Algorithm from the Common Context (see Section 2.1.6), encoded as a CBOR integer or text string, consistently with the "Value" field in the "COSE Algorithms" Registry for this algorithm.

* The 'type' element of the 'info' array is "SEKey". The label is an ASCII string and does not include a trailing NUL byte.

* L and the 'L' element of the 'info' array are the size of the key for the Group Encryption Algorithm specified in the Common Context (see Section 2.1.6), in bytes. While the obtained Signature Encryption Key is never used with the Group Encryption Algorithm, its length was chosen to obtain a matching level of security.

2.1.9. Pairwise Key Agreement Algorithm

The new parameter Pairwise Key Agreement Algorithm identifies the elliptic curve Diffie-Hellman algorithm used to derive a static-static Diffie-Hellman shared secret, from which pairwise keys are derived (see Section 2.5.1) to protect messages with the pairwise mode (see Section 9). If this parameter is not set, the pairwise mode is not used in the group.

If the HKDF Algorithm specified in the Common Context is "HKDF SHA-256" (identified as "HMAC 256/256", COSE algorithm encoding: 5), then the Pairwise Key Agreement Algorithm is "ECDH-SS + HKDF-256" (COSE algorithm encoding: -27).

If the HKDF Algorithm specified in the Common Context is "HKDF SHA-512" (identified as "HMAC 512/512", COSE algorithm encoding: 7), then the Pairwise Key Agreement Algorithm is "ECDH-SS + HKDF-512" (COSE algorithm encoding: -28).

Note that the HKDF Algorithm in the Common Context is denoted by the corresponding COSE HMAC Algorithm. For example, the HKDF Algorithm "HKDF SHA-256" is specified as the HMAC Algorithm "HMAC 256/256" (COSE algorithm encoding: 5).

More generally, if Pairwise Key Agreement Algorithm is set, it MUST identify a COSE algorithm such that: i) it performs a direct ECDH Static-Static key agreement; and ii) it indicates the use of the same HKDF Algorithm used in the group as specified in the Common Context.

2.2. Sender Context and Recipient Context

The Sender ID SHALL be unique for each endpoint in a group with a certain triplet (Master Secret, Master Salt, Group Identifier), see Section 3.3 of [RFC8613].

The maximum length of a Sender ID in bytes equals L minus 6, where L is determined as follows.

* If only one among the AEAD Algorithm and the Group Encryption Algorithm is set (see Section 2.1.1 and Section 2.1.6), then L is the AEAD nonce length for the set algorithm.

* If both the AEAD Algorithm and the Group Encryption Algorithm are set, then L is the smallest AEAD nonce length among those of the two algorithms.

With the exception of the authentication credential of the sender endpoint, a receiver endpoint can derive a complete Security Context from a received Group OSCORE message and the Common Context (see Section 2.3).

The authentication credentials in the Recipient Contexts can be retrieved from the Group Manager (see Section 3) upon joining the group. An authentication credential can alternatively be acquired from the Group Manager at a later time, for example the first time a message is received from a particular endpoint in the group (see Section 8.2 and Section 8.4).

For severely constrained devices, it may be infeasible to simultaneously handle the ongoing processing of a recently received message in parallel with the retrieval of the sender endpoint’s authentication credential. Such devices can be configured to drop a received message for which there is no (complete) Recipient Context, and retrieve the sender endpoint’s authentication credential in order to have it available to verify subsequent messages from that endpoint.

An endpoint may admit a maximum number of Recipient Contexts for a same Security Context, e.g., due to memory limitations. After reaching that limit, the endpoint has to delete a current Recipient Context to install a new one (see Section 2.6.1.2). It is up to the application to define policies for Recipient Contexts to delete.

2.3. Establishment of Security Context Parameters

OSCORE defines the derivation of Sender Context and Recipient Context (specifically, of Sender/Recipient Keys) and of the Common IV, from a set of input parameters (see Section 3.2 of [RFC8613]).

The derivation of Sender/Recipient Keys and of the Common IV defined in OSCORE applies also to Group OSCORE, with the following modifications compared to Section 3.2.1 of [RFC8613].

* If Group Encryption Algorithm in the Common Context is set (see Section 2.1.6), then the 'alg_aead' element of the 'info' array MUST specify Group Encryption Algorithm from the Common Context as a CBOR integer or text string, consistently with the "Value" field in the "COSE Algorithms" Registry for this algorithm.

* If Group Encryption Algorithm in the Common Context is not set, then the 'alg_aead' element of the 'info' array MUST specify AEAD Algorithm from the Common Context (see Section 2.1.1), as per Section 5.4 of [RFC8613].

* When deriving the Common IV, the 'L' element of the 'info' array MUST specify the length of the Common IV in bytes, which is determined as defined in Section 2.1.3.

2.4. Authentication Credentials

The authentication credentials of the endpoints in a group MUST be encoded according to the format used in the group, as indicated by the Authentication Credential Format parameter in the Common Context (see Section 2.1.4). The authentication credential of the Group Manager SHOULD be encoded according to that same format. The format of authentication credentials MUST provide the public key and a comprehensive set of information related to the public key algorithm, including, e.g., the used elliptic curve (when applicable).

If Group Encryption Algorithm in the Common Context is not set (see Section 2.1.6), then the public key algorithm is the Pairwise Key Agreement Algorithm used in the group (see Section 2.1.9), else the Signature Algorithm used in the group (see Section 2.1.7).

If the authentication credentials are X.509 certificates [RFC5280] or C509 certificates [I-D.ietf-cose-cbor-encoded-cert], the public key algorithm is fully described by the "algorithm" field of the "SubjectPublicKeyInfo" structure, and by the "subjectPublicKeyAlgorithm" element, respectively.

If authentication credentials are CBOR Web Tokens (CWTs) or CWT Claims Sets [RFC8392], the public key algorithm is fully described by a COSE key type and its "kty" and "crv" parameters.

Authentication credentials are used to derive pairwise keys (see Section 2.5.1) and are included in the external additional authenticated data when processing messages (see Section 4.4). In both these cases, an endpoint in a group MUST treat authentication credentials as opaque data, i.e., by considering the same binary representation made available to other endpoints in the group, possibly through a designated trusted source (e.g., the Group Manager).

For example, an X.509 certificate is provided as its direct binary serialization. If C509 certificates or CWTs are used as authentication credentials, each is provided as the binary serialization of a (possibly tagged) CBOR array. If CWT Claims Sets are used as authentication credentials, each is provided as the binary serialization of a CBOR map.

If authentication credentials are CWTs, then the untagged CWT associated with an entity is stored in the Security Context and used as authentication credential for that entity.

If authentication credentials are X.509 / C509 certificates or CWTs, and the authentication credential associated with an entity is provided within a chain or a bag, then only the end-entity certificate or end-entity untagged CWT is stored in the Security Context and used as authentication credential for that entity.

Storing whole authentication credentials rather than only a subset of those may result in a non-negligible storage overhead. On the other hand, it also ensures that authentication credentials are correctly used in a simple, flexible and non-error-prone way, also taking into account future credential formats as entirely new or extending existing ones. In particular, it is ensured that:

* When used to derive pairwise keys and when included in the external additional authenticated data, authentication credentials can also specify possible metadata and parameters related to the included public key. Besides the public key algorithm, these comprise other relevant pieces of information such as key usage, expiration time, issuer, and subject.

* All endpoints using another endpoint's authentication credential use exactly the same binary serialization, as obtained and distributed by the credential provider (e.g., the Group Manager), and as originally crafted by the credential issuer. In turn, this does not require to define and maintain canonical subsets of authentication credentials and their corresponding encoding, and spares endpoints from error-prone re-encoding operations.

Depending on the particular deployment and the intended group size, limiting the storage overhead of endpoints in a group can be an incentive for system/network administrators to prefer using a compact format of authentication credentials in the first place.

2.5. Pairwise Keys

Certain signature schemes, such as EdDSA and ECDSA, support a secure combined signature and encryption scheme. This section specifies the derivation of "pairwise keys", for use in the pairwise mode defined in Section 9.

Group OSCORE keys used for both signature and encryption MUST be used only for purposes related to Group OSCORE. These include the processing of messages with Group OSCORE, as well as performing proof-of-possession of private keys, e.g., upon joining a group through the Group Manager (see Section 3).

2.5.1. Derivation of Pairwise Keys

Using the Group OSCORE Security Context (see Section 2), a group member can derive AEAD keys, to protect point-to-point communication between itself and any other endpoint X in the group by means of the AEAD Algorithm from the Common Context (see Section 2.1.1). The key derivation of these so-called pairwise keys follows the same construction as in Section 3.2.1 of [RFC8613]:

```
Pairwise Sender Key    = HKDF(Sender Key, IKM-Sender, info, L)
Pairwise Recipient Key = HKDF(Recipient Key, IKM-Recipient, info, L)
```

with

```
IKM-Sender    = Sender Auth Cred | Recipient Auth Cred | Shared Secret
IKM-Recipient = Recipient Auth Cred | Sender Auth Cred | Shared Secret
```

where:

* The Pairwise Sender Key is the AEAD key for processing outgoing messages addressed to endpoint X.

* The Pairwise Recipient Key is the AEAD key for processing incoming messages from endpoint X.

* HKDF is the OSCORE HKDF algorithm [RFC8613] from the Common Context.

* The Sender Key from the Sender Context is used as salt in the HKDF, when deriving the Pairwise Sender Key.

* The Recipient Key from the Recipient Context associated with endpoint X is used as salt in the HKDF, when deriving the Pairwise Recipient Key.

* Sender Auth Cred is the endpoint's own authentication credential from the Sender Context.

* Recipient Auth Cred is the endpoint X's authentication credential from the Recipient Context associated with the endpoint X.

* The Shared Secret is computed as a cofactor Diffie-Hellman shared secret, see Section 5.7.1.2 of [NIST-800-56A], using the Pairwise Key Agreement Algorithm. The endpoint uses its private key from the Sender Context and the other endpoint's public key included in Recipient Auth Cred. Note the requirement of validation of public keys in Section 13.16.

  In case the other endpoint's public key has COSE Key Type "EC2" (e.g., for the curves P-256, P-384, and P-512), then the public key is used as is. In case the other endpoint's public key has COSE Key Type "OKP", the procedure is described in Section 5 of [RFC7748]. In particular, if the public key is for X25519 or X448, it is used as is. Otherwise, if the public key is for the curve Ed25519 or Ed448, it is first mapped to Montgomery coordinates (see Section 2.5.2).

* IKM-Sender is the Input Keying Material (IKM) used in the HKDF for the derivation of the Pairwise Sender Key. IKM-Sender is the byte string concatenation of Sender Auth Cred, Recipient Auth Cred, and the Shared Secret. The authentication credentials Sender Auth Cred and Recipient Auth Cred are binary encoded as defined in Section 2.4.

* IKM-Recipient is the Input Keying Material (IKM) used in the HKDF for the derivation of the Pairwise Recipient Key. IKM-Recipient is the byte string concatenation of Recipient Auth Cred, Sender Auth Cred, and the Shared Secret. The authentication credentials Recipient Auth Cred and Sender Auth Cred are binary encoded as defined in Section 2.4.

* info and L are as defined in Section 3.2.1 of [RFC8613]. That is:

  - The 'alg_aead' element of the 'info' array takes the value of AEAD Algorithm from the Common Context (see Section 2.1.1).

  - L and the 'L' element of the 'info' array are the size of the key for the AEAD Algorithm from the Common Context (see Section 2.1.1), in bytes.

If EdDSA asymmetric keys are used, the Edward coordinates are mapped to Montgomery coordinates using the maps defined in Sections 4.1 and 4.2 of [RFC7748], before using the X25519 or X448 function defined in Section 5 of [RFC7748]. For further details, see Section 2.5.2. ECC asymmetric keys in Montgomery or Weierstrass form are used directly in the key agreement algorithm, without coordinate mapping.

After establishing a partially or completely new Security Context (see Section 2.6 and Section 3.2), the old pairwise keys MUST be deleted. Since new Sender/Recipient Keys are derived from the new group keying material (see Section 2.2), every group member MUST use the new Sender/Recipient Keys when deriving new pairwise keys.

As long as any two group members preserve the same asymmetric keys, their Diffie-Hellman shared secret does not change across updates of the group keying material.

2.5.2. ECDH with Montgomery Coordinates

2.5.2.1. Curve25519

The y-coordinate of the other endpoint's Ed25519 public key is decoded as specified in Section 5.1.3 of [RFC8032]. The Curve25519 u-coordinate is recovered as u = (1 + y) / (1 - y) (mod p) following the map in Section 4.1 of [RFC7748]. Note that the mapping is not defined for y = 1, and that y = -1 maps to u = 0 which corresponds to the neutral group element and thus will result in a degenerate shared secret. Therefore, implementations MUST abort if the y-coordinate of the other endpoint's Ed25519 public key is 1 or -1 (mod p).

The private signing key byte strings (i.e., the lower 32 bytes used for generating the public key, see step 1 of Section 5.1.5 of [RFC8032]) are decoded the same way for signing in Ed25519 and scalar multiplication in X25519. Hence, in order to compute the shared secret, the endpoint applies the X25519 function to the Ed25519 private signing key byte string and the encoded u-coordinate byte string as specified in Section 5 of [RFC7748].

2.5.2.2. Curve448

The y-coordinate of the other endpoint's Ed448 public key is decoded as specified in Section 5.2.3. of [RFC8032]. The Curve448 u-coordinate is recovered as u = y^2 * (d * y^2 - 1) / (y^2 - 1) (mod p) following the map from "edwards448" in Section 4.2 of [RFC7748], and also using the relation x^2 = (y^2 - 1)/(d * y^2 - 1) from the curve equation. Note that the mapping is not defined for y = 1 or -1. Therefore, implementations MUST abort if the y-coordinate of the peer endpoint's Ed448 public key is 1 or -1 (mod p).

The private signing key byte strings (i.e., the lower 57 bytes used for generating the public key, see step 1 of Section 5.2.5 of [RFC8032]) are decoded the same way for signing in Ed448 and scalar multiplication in X448. Hence, in order to compute the shared secret, the endpoint applies the X448 function to the Ed448 private signing key byte string and the encoded u-coordinate byte string as specified in Section 5 of [RFC7748].

2.5.3. Usage of Sequence Numbers

When using any of its Pairwise Sender Keys, a sender endpoint including the 'Partial IV' parameter in the protected message MUST use the current fresh value of the Sender Sequence Number from its Sender Context (see Section 2.2). That is, the same Sender Sequence Number space is used for all outgoing messages protected with Group OSCORE, thus limiting both storage and complexity.

On the other hand, when combining communications with the group mode and the pairwise mode, this may result in the Partial IV values moving forward more often. This can happen when a client engages in frequent or long sequences of one-to-one exchanges with servers in the group, by sending requests over unicast. In turn, this contributes to a sooner exhaustion of the Sender Sequence Number space of the client, which would then require to take actions for deriving a new Sender Context before resuming communications in the group (see Section 2.6.2).

2.5.4. Security Context for Pairwise Mode

If the pairwise mode is supported, the Security Context additionally includes the Pairwise Key Agreement Algorithm and the pairwise keys, as described at the beginning of Section 2.

The pairwise keys as well as the shared secrets used in their derivation (see Section 2.5.1) may be stored in memory or recomputed every time they are needed. The shared secret changes only when a public/private key pair used for its derivation changes, which results in the pairwise keys also changing. Additionally, the pairwise keys change if the Sender ID changes or if a new Security Context is established for the group (see Section 2.6.3). In order to optimize protocol performance, an endpoint may store the derived pairwise keys for easy retrieval.

In the pairwise mode, the Sender Context includes the Pairwise Sender Keys to use with the other endpoints (see Figure 1). In order to identify the right key to use, the Pairwise Sender Key for endpoint X may be associated with the Recipient ID of endpoint X, as defined in the Recipient Context (i.e., the Sender ID from the point of view of endpoint X). In this way, the Recipient ID can be used to lookup for the right Pairwise Sender Key. This association may be implemented in different ways, e.g., by storing the pair (Recipient ID, Pairwise Sender Key) or linking a Pairwise Sender Key to a Recipient Context.

2.6. Update of Security Context

It is RECOMMENDED that the immutable part of the Security Context is stored in non-volatile memory, or that it can otherwise be reliably accessed throughout the operation of the group, e.g., after device reboots. However, also immutable parts of the Security Context may need to be updated, for example due to scheduled key renewal, new or re-joining members in the group, or the fact that the endpoint changes Sender ID (see Section 2.6.3).

The mutable parts of the Security Context are updated by the endpoint when executing the security protocol, but may be lost (see Section 2.6.1) or become outdated by exhaustion of Sender Sequence Numbers (see Section 2.6.2).

2.6.1. Loss of Mutable Security Context

An endpoint may lose its mutable Security Context, e.g., due to a reboot occurred in an unprepared way (see Section 2.6.1.1) or due to a deleted Recipient Context (see Section 2.6.1.2).

If it is not feasible or practically possible to store and maintain up-to-date the mutable part in non-volatile memory (e.g., due to limited number of write operations), the endpoint MUST be able to detect a loss of the mutable Security Context, to prevent the re-use of a nonce with the same AEAD key, and to handle incoming replayed messages.

2.6.1.1. Total Loss

In case a loss of the Sender Context and/or of the Recipient Contexts is detected (e.g., following a reboot occurred in an unprepared way), the endpoint MUST NOT protect further messages using this Security Context, to avoid reusing an AEAD nonce with the same AEAD key.

Before resuming its operations in the group, the endpoint MUST retrieve new Security Context parameters from the Group Manager (see Section 2.6.3) and use them to derive a new Sender Context and Recipient Contexts (see Section 2.2). Since the new Sender Context includes newly derived encryption keys, an endpoint will not reuse the same pair (key, nonce), even when it is a server using the Partial IV of (old re-injected) requests to build the AEAD nonce for protecting the responses.

From then on, the endpoint MUST use the latest installed Sender Context to protect outgoing messages. Newly derived Recipient Contexts will have a Replay Window which is initialized as valid.

If an endpoint is not configured as silent server and is not able to establish an updated Sender Context, e.g., because of lack of connectivity with the Group Manager, then the endpoint MUST NOT protect further messages using the current Security Context and MUST NOT accept incoming messages from other group members, as currently unable to detect possible replays.

If an endpoint is configured as silent server and is not able to establish an updated Security Context, e.g., because of lack of connectivity with the Group Manager, then the endpoint MUST NOT accept incoming messages from other group members, as currently unable to detect possible replays.

An adversary may leverage the above to perform a Denial of Service attack and prevent some group members from communicating altogether. That is, the adversary can first block the communication path between the Group Manager and some individual group members. This can be achieved, for instance, by injecting fake responses to DNS queries for the Group Manager hostname, or by removing a network link used for routing traffic towards the Group Manager. Then, the adversary can induce an unprepared reboot for some endpoints in the group, e.g., by triggering a short power outage. After that, such endpoints that have lost their Sender Context and/or Recipient Contexts following the reboot would not be able to obtain new Security Context parameters from the Group Manager, as specified above. Thus, they would not be able to further communicate in the group until connectivity with the Group Manager is restored.

2.6.1.2. Deleted Recipient Contexts

The Security Context may contain a large and variable number of Recipient Contexts. A Recipient Context may need to be deleted, because the maximum number of Recipient Contexts has been reached (see Section 2.2), or due to some other reason.

When a Recipient Context is deleted, information about previously received messages from the corresponding other endpoint is lost. Therefore, if the Recipient Context is derived again from the same Security Context, there is a risk that a replayed message is not detected. If one Recipient Context has been deleted from the current Security Context, then the Replay Window of any new Recipient Context in this Security Context MUST be initialized as invalid. Messages associated with a Recipient Context that has an invalid Replay Window MUST NOT be delivered to the application.

If the endpoint receives a request to process with the new Recipient Context and the endpoint supports the CoAP Echo Option [RFC9175], then it is RECOMMENDED to follow the procedure specified in Section 10 which establishes a valid Replay Window. In particular, the endpoint MUST use its Partial IV when generating the AEAD nonce and MUST include the Partial IV in the response message conveying the Echo Option.

Alternatively, the endpoint MAY retrieve or wait for new Security Context parameters from the Group Manager and derive new Sender and Recipient Contexts, as defined in Section 2.6.1.1. In this case the Replay Windows of all Recipient Contexts become valid if they are not already.

2.6.2. Exhaustion of Sender Sequence Number Space

Since an endpoint increments its Sender Sequence Number for each new outgoing message including a Partial IV, an endpoint can eventually exhaust the Sender Sequence Number space.

Implementations MUST be able to detect an exhaustion of Sender Sequence Number space, after the endpoint has consumed the largest usable value. This may be influenced by additional limitations besides the mere 40-bit size limit of the Partial IV. If an implementation's integers support wrapping addition, the implementation MUST treat the Sender Sequence Number as exhausted when a wrap-around is detected.

Upon exhausting the Sender Sequence Number space, the endpoint MUST NOT use this Security Context to protect further messages including a Partial IV.

The endpoint SHOULD inform the Group Manager, retrieve new Security Context parameters from the Group Manager (see Section 2.6.3), and use them to derive a new Sender Context (see Section 2.2).

From then on, the endpoint MUST use its latest installed Sender Context to protect outgoing messages.

2.6.3. Retrieving New Security Context Parameters

The Group Manager can assist an endpoint with an incomplete Sender Context to retrieve missing data of the Security Context and thereby become fully operational in the group again. The two main options for the Group Manager are described in this section: i) assignment of a new Sender ID to the endpoint (see Section 2.6.3.1); and ii) establishment of a new Security Context for the group (see Section 2.6.3.2). The update of the Replay Window in each of the Recipient Contexts is discussed in Section 2.6.1.

As group membership changes, or as group members get new Sender IDs (see Section 2.6.3.1), so do the relevant Recipient IDs that the other endpoints need to keep track of. As a consequence, group members may end up retaining stale Recipient Contexts, that are no longer useful to verify incoming secure messages.

The Recipient ID ('kid') SHOULD NOT be considered as a persistent and reliable identifier of a group member. Such an indication can be achieved only by using that member's public key, when verifying countersignatures of received messages (in group mode), or when verifying messages integrity-protected with pairwise keying material derived from authentication credentials and associated asymmetric keys (in pairwise mode).

Furthermore, applications MAY define policies to: i) delete (long-)unused Recipient Contexts and reduce the impact on storage space; as well as ii) check with the Group Manager that an authentication credential with the public key included therein is currently the one associated with a 'kid' value, after a number of consecutive failed verifications.

2.6.3.1. New Sender ID for the Endpoint

The Group Manager may assign a new Sender ID to an endpoint, while leaving the Gid, Master Secret and Master Salt unchanged in the group. In this case, the Group Manager MUST assign a Sender ID that has not been used in the group since the latest time when the current Gid value was assigned to the group (see Section 3.2).

Having retrieved the new Sender ID, and potentially other missing data of the immutable Security Context, the endpoint can derive a new Sender Context (see Section 2.2). When doing so, the endpoint resets the Sender Sequence Number in its Sender Context to 0, and derives a new Sender Key. This is in turn used to possibly derive new Pairwise Sender Keys.

From then on, the endpoint MUST use its latest installed Sender Context to protect outgoing messages.

The assignment of a new Sender ID may be the result of different processes. The endpoint may request a new Sender ID, e.g., because of the exhaustion of the Sender Sequence Number space (see Section 2.6.2). An endpoint may request to re-join the group, e.g., because of losing its mutable Security Context (see Section 2.6.1), and is provided with a new Sender ID together with the latest immutable Security Context.

For the other group members, the Recipient Context corresponding to the old Sender ID becomes stale (see Section 3.2).

2.6.3.2. New Security Context for the Group

The Group Manager may establish a new Security Context for the group (see Section 3.2). The Group Manager does not necessarily establish a new Security Context for the group if one member has an outdated Security Context (see Section 2.6.3.1), unless that was already planned or required for other reasons.

All the group members need to acquire new Security Context parameters from the Group Manager. Once having acquired new Security Context parameters, each group member performs the following actions.

* From then on, it MUST NOT use the current Security Context to start processing new messages for the considered group.

* It completes any ongoing message processing for the considered group.

* It derives and installs a new Security Context. In particular:

  - It re-derives the keying material stored in its Sender Context and Recipient Contexts (see Section 2.2). The Master Salt used for the re-derivations is the updated Master Salt parameter if provided by the Group Manager, or the empty byte string otherwise.

  - It resets its Sender Sequence Number in its Sender Context to 0.

  - It re-initializes the Replay Window of each Recipient Context as valid and with 0 as its current lower limit.

  - For each long exchange where it is a client and that it wants to keep active, it sets the Response Number of each associated server as not initialized (see Section 6.1).

From then on, it can resume processing new messages for the considered group. In particular:

* It MUST use its latest installed Sender Context to protect outgoing messages.

* It SHOULD use only its latest installed Recipient Contexts to process incoming messages, unless application policies admit to temporarily retain and use the old, recent, Security Context (see Section 13.5.1).

The distribution of a new Gid and Master Secret may result in temporarily misaligned Security Contexts among group members. In particular, this may result in a group member not being able to process messages received right after a new Gid and Master Secret have been distributed. A discussion on practical consequences and possible ways to address them, as well as on how to handle the old Security Context, is provided in Section 13.5.

3. The Group Manager

As with OSCORE, endpoints communicating with Group OSCORE need to establish the relevant Security Context. Group OSCORE endpoints need to acquire OSCORE input parameters, information about the group(s) and about other endpoints in the group(s). This document is based on the existence of an entity called Group Manager that is responsible for the group, but it does not mandate how the Group Manager interacts with the group members. The list of responsibilities of the Group Manager is compiled in Section 3.4.

The Group Manager assigns unique Group Identifiers (Gids) to the groups under its control. Within each of such groups, the Group Manager assigns unique Sender IDs (and thus Recipient IDs) to the respective group members. The maximum length of Sender IDs depends on the length of the AEAD nonce for the algorithms used in the group (see Section 2.2).

According to a hierarchical approach, the Gid value assigned to a group is associated with a dedicated space for the values of Sender ID and Recipient ID of the members of that group. When an endpoint (re-)joins a group, it is provided with the current Gid to use in the group. The Group Manager also assigns an integer Key Generation Number counter to each of its groups, identifying the current version of the keying material used in that group. Further details about identifiers and keys are provided in Section 3.2.

The Group Manager maintains records of the authentication credentials of endpoints in a group, and provides information about the group and its members to other group members (see Section 3.1), and to external entities with a specific role (see Section 3.3).

3.1. Set-up of New Endpoints

From the Group Manager, an endpoint acquires group data such as the Gid and OSCORE input parameters including its own Sender ID, with which it can derive the Sender Context.

When joining the group or later on as a group member, an endpoint can also retrieve from the Group Manager the authentication credential of the Group Manager as well as the authentication credential and other information associated with other members of the group, with which it can derive the corresponding Recipient Context. An application can configure a group member to asynchronously retrieve information about Recipient Contexts, e.g., by Observing [RFC7641] a resource at the Group Manager to get updates on the group membership.

Upon endpoints' joining, the Group Manager collects their authentication credentials and MUST verify proof-of-possession of the respective private key. Together with the requested authentication credentials of other group members, the Group Manager MUST provide the joining endpoints with the Sender ID of the associated group members and the current Key Generation Number in the group (see Section 3.2).

An endpoint may join a group, for example, by explicitly interacting with the responsible Group Manager, or by being configured with some tool performing the tasks of the 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 that the Group Manager has with joining endpoints and group members MUST be secured. Specific details on how to secure such communications 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 the scope of this document.

In case of successful authorization check, the Group Manager provides the joining endpoint with the keying material to initialize the Security Context. The actual provisioning of keying material and parameters to the joining endpoint is out of the scope of this document.

One realization of a Group Manager is specified 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 [RFC9200].

3.2. Management of Group Keying Material

In order to establish a new Security Context for a group, the Group Manager MUST generate and assign to the group a new Group Identifier (Gid) and a new value for the Master Secret parameter. When doing so, a new value for the Master Salt parameter MAY also be generated and assigned to the group. When establishing the new Security Context, the Group Manager should preserve the current value of the Sender ID of each group member.

The specific group key management scheme used to distribute new keying material is out of the scope of this document. A simple group key management scheme is defined in [I-D.ietf-ace-key-groupcomm-oscore]. When possible, the delivery of rekeying messages should use a reliable transport, in order to reduce chances of group members missing a rekeying instance.

The set of group members should not be assumed as fixed, i.e., the group membership is subject to changes, possibly on a frequent basis.

The Group Manager MUST rekey the group without undue delay in case one or more endpoints leave the group. An endpoint may leave the group at own initiative, or may be evicted from the group by the Group Manager, e.g., in case an endpoint is compromised, or is suspected to be compromised. In either case, rekeying the group excludes such endpoints from future communications in the group, and thus preserves forward security. If a network node is compromised or suspected to be compromised, the Group Manager MUST evict from the group all the endpoints hosted by that node that are member of the group and rekey the group accordingly.

If required by the application, the Group Manager MUST rekey the group also before one or more new joining endpoints are added to the group, thus preserving backward security.

Separately for each group, the value of the Key Generation Number increases by one each time the Group Manager distributes new keying material to that group (see Section 3.2). The first Key Generation Number assigned to every group MUST be 0.

The establishment of the new Security Context for the group takes the following steps.

1. The Group Manager MUST increment the Key Generation Number for the group by 1.

2. The Group Manager MUST build a set of stale Sender IDs including:

* The Sender IDs that, during the current Gid, were both assigned to an endpoint and subsequently relinquished (see Section 2.6.3.1).

* The current Sender IDs of the group members that the upcoming group rekeying aims to exclude from future group communications, if any.

3. The Group Manager rekeys the group, by distributing:

   * The new keying material, i.e., the new Master Secret, the new Gid and (optionally) the new Master Salt.

   * The new Key Generation Number from step 1.

   * The set of stale Sender IDs from step 2.

   Further information may be distributed, depending on the specific group key management scheme used in the group.

When receiving the new group keying material, a group member considers the received stale Sender IDs and performs the following actions.

* The group member MUST remove every authentication credential associated with a stale Sender ID from its list of group members' authentication credentials used in the group.

* The group member MUST delete each of its Recipient Contexts used in the group whose corresponding Recipient ID is a stale Sender ID.

After that, the group member installs the new keying material and derives the corresponding new Security Context.

A group member might miss one or more consecutive instances of group rekeying. As a result, the group member will retain old group keying material with Key Generation Number GEN_OLD. Eventually, the group member can notice the discrepancy, e.g., by repeatedly failing to verify incoming messages, or by explicitly querying the Group Manager for the current Key Generation Number. Once the group member gains knowledge of having missed a group rekeying, it MUST delete the old keying material it stores.

Then, the group member proceeds according to the following steps.

1. The group member retrieves from the Group Manager the current group keying material, together with the current Key Generation Number GEN_NEW. The group member MUST NOT install the obtained group keying material yet.

2. The group member asks the Group Manager for the set of stale Sender IDs.

3. If no exact indication can be obtained from the Group Manager, the group member MUST remove all the authentication credentials from its list of group members' authentication credentials used in the group and MUST delete all its Recipient Contexts used in the group.

   Otherwise, the group member MUST remove every authentication credential associated with a stale Sender ID from its list of group members' authentication credentials used in the group, and MUST delete each of its Recipient Contexts used in the group whose corresponding Recipient ID is a stale Sender ID.

4. The group member installs the current group keying material, and derives the corresponding new Security Context.

Alternatively, the group member can re-join the group. In such a case, the group member MUST take one of the following two actions.

* First, the group member performs steps 2 and 3 above. Then, the group member re-joins the group.

* The group member re-joins the group with the same roles it currently has in the group, and, during the re-joining process, it asks the Group Manager for the authentication credentials of all the current group members.

  Then, given Z, the set of authentication credentials received from the Group Manager, the group member removes every authentication credential which is not in Z from its list of group members' authentication credentials used in the group, and deletes each of its Recipient Contexts used in the group that does not include any of the authentication credentials in Z.

By removing authentication credentials and deleting Recipient Contexts associated with stale Sender IDs, it is ensured that a recipient endpoint storing the latest group keying material does not store the authentication credentials of sender endpoints that are not current group members. This in turn allows group members to rely on stored authentication credentials to confidently assert the group membership of sender endpoints, when receiving incoming messages protected in group mode (see Section 8).

Such a strictness in managing the authentication credentials and Recipient Contexts associated with other group members is required for two reasons. First, as further discussed in Section 13.1, it ensures that the group mode can be used securely, even in a group where the Group Encryption Algorithm does not provide integrity protection (see Section 2.1.6) and external signature checkers are used (see Section 8.5). Second, it ensures that the wrong (old) authentication credential associated with a group member A is never used with a sender ID that used to be associated with A and has been later issued to a different group member B (see Section 3.2.1.2), thus preventing the need to recover from an identity mix-up.

3.2.1. Recycling of Identifiers

This section specifies how the Group Manager handles and possibly reassigns Gid values and Sender ID values in a group.

3.2.1.1. Recycling of Group Identifiers

Since the Gid value changes every time a group is rekeyed, it can happen that, after several rekeying instances, the whole space of Gid values has been used for the group in question. When this happens, the Group Manager has no available Gid values to use that have never been assigned to the group during the group’s lifetime.

The occurrence of such an event and how long it would take to occur depend on the format and encoding of Gid values used in the group (see, e.g., Appendix C), as well as on the frequency of rekeying instances yielding a change of Gid value. Independently for each group under its control, the Group Manager can take one of the two following approaches.

* The Group Manager does not reassign Gid values. That is, once the whole space of Gid values has been used for a group, the Group Manager terminates the group and may re-establish a new group.

* While the Gid value changes every time a group is rekeyed, the Group Manager can reassign Gid values previously used during a group's lifetime. By doing so, the group can continue to exist even once the whole space of Gid values has been used.

  The Group Manager MAY support and use this approach. In such a case, the Group Manager MUST take additional actions when handling Gid values and rekeying the group, as specified below.

  When a node (re-)joins the group and it is provided with the current Gid to use in the group, the Group Manager considers such a Gid as the Birth Gid of that endpoint for that group. For each group member, the Group Manager MUST store the latest corresponding Birth Gid until that member leaves the group. In case the endpoint has in fact re-joined the group, the newly determined Birth Gid overwrites the one currently stored.

  When establishing a new Security Context for the group, the Group Manager takes the additional following step between steps 1 and 2 of Section 3.2.

  A. The Group Manager MUST check if the new Gid to be distributed is equal to the Birth Gid of any of the current group members. If any of such "elder members" is found in the group, then:

  - The Group Manager MUST evict the elder members from the group. That is, the Group Manager MUST terminate their membership and, in the following steps, it MUST rekey the group in such a way that the new keying material is not provided to those evicted elder members.

    This ensures that any response from the same server to the request of a long exchange can never successfully match against the request of two different long exchanges.

    In fact, the excluded elder members could eventually re-join the group, thus terminating any of their ongoing long exchanges (see Section 6.1).

    Therefore, it is ensured by construction that no client can have with the same server two ongoing long exchanges, such that the two respective requests were protected using the same Partial IV, Gid, and Sender ID.

3.2.1.2. Recycling of Sender IDs

From the moment when a Gid is assigned to a group until the moment a new Gid is assigned to that same group, the Group Manager MUST NOT reassign a Sender ID within the group. This prevents to reuse a Sender ID ('kid') with the same triple (Gid, Master Secret, Master Salt). Within this restriction, the Group Manager can assign a Sender ID used under an old Gid value (including under a same, recycled Gid value), thus avoiding Sender ID values to irrecoverably grow in size.

Even when an endpoint joining a group is recognized as a current member of that group, e.g., through the ongoing secure communication association, the Group Manager MUST assign a new Sender ID different than the one currently used by the endpoint in the group, unless the group is rekeyed first and a new Gid value is established.

3.2.1.3. Relation between Identifiers and Keying Material

Figure 2 overviews the different identifiers and keying material components, considering their relation and possible reuse across group rekeying.

```
 Components changed in lockstep
     upon a group rekeying
 +----------------------------+            * Changing a kid does not
 |                            |              need changing the Group ID
 | Master               Group |<--> kid1
 | Secret <---> o <--->  ID   |            * A kid is not reassigned
 |              ^             |<--> kid2     under the ongoing usage of
 |              |             |              the current Group ID
 |              |             |<--> kid3
 |              v             |            * Upon changing the Group ID,
 |         Master Salt        | ... ...      every current kid should
 |         (optional)         |              be preserved for efficient
 |                            |              key rollover
 | The Key Generation Number  |
 | is incremented by 1        |            * After changing Group ID, an
 |                            |              unused kid can be assigned,
 +----------------------------+              even if it was used before
                                             the Group ID change
```

Figure 2: Relations among keying material components.

3.3. Support for Signature Checkers

The Group Manager may serve signature checkers, e.g., intermediary gateways, which verify countersignatures of messages protected in group mode (see Section 8.5). These entities do not join a group as members, but can retrieve authentication credentials of group members and other selected group data from the Group Manager.

In order to verify countersignatures of messages in a group, a signature checker needs to retrieve the following information about the group:

* The current ID Context (Gid) used in the group.

* The authentication credentials of the group members and of the Group Manager.

  If the signature checker is provided with a CWT for a given entity, then the authentication credential associated with that entity is the untagged CWT.

  If the signature checker is provided with a chain or a bag of X.509 / C509 certificates or of CWTs for a given entity, then the authentication credential associated with that entity is the end-entity certificate or end-entity untagged CWT.

* The current Signature Encryption Key (see Section 2.1.8).

* The identifiers of the algorithms used in the group (see Section 2), i.e.: i) Group Encryption Algorithm and Signature Algorithm; and ii) AEAD Algorithm and Pairwise Key Agreement Algorithm, if such parameters are set in the Common Context (see Section 2.1.1 and Section 2.1.9).

A signature checker MUST be authorized before it can retrieve such information, for example with the use of [I-D.ietf-ace-key-groupcomm-oscore].

3.4. 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, ensuring uniqueness of Gids within the set of its OSCORE groups and, optionally, the secure recycling of Gids.

2. Defining policies for authorizing the joining of its OSCORE groups.

3. Handling the join process to add new endpoints as group members.

4. Establishing the Common Context part of the Security Context, and providing it to authorized group members during the join process, together with the corresponding Sender Context.

5. Updating the Key Generation Number and the Gid of its OSCORE groups, upon renewing the respective Security Context.

6. Generating and managing Sender IDs within its OSCORE groups, as well as assigning and providing them to new endpoints during the join process, or to current group members upon request of renewal or re-joining. This includes ensuring that:

   * Each Sender ID is unique within each of the OSCORE groups;

   * Each Sender ID is not reassigned within the same group since the latest time when the current Gid value was assigned to the group. That is, the Sender ID is not reassigned even to a current group member re-joining the same group, without a rekeying happening first.

7. Defining communication policies for each of its OSCORE groups, and signaling them to new endpoints during the join process.

8. Renewing the Security Context of an OSCORE group upon membership change, by revoking and renewing common security parameters and keying material (rekeying).

9. Providing the management keying material that a new endpoint requires to participate in the rekeying process, consistently with the key management scheme used in the group joined by the new endpoint.

10. Assisting a group member that has missed a group rekeying instance to understand which authentication credentials and Recipient Contexts to delete, as associated with former group members.

11. Acting as key repository, in order to handle the authentication credentials of the members of its OSCORE groups, and providing such authentication credentials to other members of the same group upon request. The actual storage of authentication credentials may be entrusted to a separate secure storage device or service.

12. Validating that the format and parameters of authentication credentials of group members are consistent with the public key algorithm and related parameters used in the respective OSCORE group.

The Group Manager specified in [I-D.ietf-ace-key-groupcomm-oscore] provides this functionality.

4. The COSE Object

Building on Section 5 of [RFC8613], this section defines how to use COSE [RFC9052] to wrap and protect data in the original message. Like OSCORE, Group OSCORE uses the untagged COSE_Encrypt0 structure with an Authenticated Encryption with Associated Data (AEAD) algorithm. Unless otherwise specified, the following modifications to what is defined for OSCORE apply, for both the group mode and the pairwise mode of Group OSCORE.

4.1. Countersignature

When protecting a message in group mode, the 'unprotected' field MUST additionally include the following parameter:

* Countersignature0 version 2: its value is set to the countersignature of the COSE object.

  The countersignature is computed by the sender as described in Sections 3.2 and 3.3 of [RFC9338], by using its private key and according to the Signature Algorithm in the Security Context.

  In particular, the Countersign_structure contains the context text string "CounterSignature0", the external_aad as defined in Section 4.4 of this document, and the ciphertext of the COSE object as payload.

4.1.1. Clarifications on Using a Countersignature

The literature commonly refers to a countersignature as a signature computed by an entity A over a document already protected by a different entity B.

However, the COSE_Countersignature0 structure belongs to the set of abbreviated countersignatures defined in Sections 3.2 and 3.3 of [RFC9338], which were designed primarily to deal with the problem of encrypted group messaging, but where it is required to know who originated the message.

Since the parameters for computing or verifying the abbreviated countersignature generated by A are provided by the same context used to describe the security processing performed by B and to be countersigned, these structures are applicable also when the two entities A and B are actually the same one, like the sender of a Group OSCORE message protected in group mode.

4.2. The 'kid' and 'kid context' parameters

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, if the request was protected in group mode. That is, unlike in [RFC8613], the 'kid' parameter is always present in responses to a request that was protected in group mode.

The value of the 'kid context' parameter in the 'unprotected' field of request messages MUST be set to the ID Context, i.e., the Group Identifier value (Gid) of the group. That is, unlike in [RFC8613], the 'kid context' parameter is always present in requests.

4.3. AEAD Nonce

The AEAD nonce is constructed like in OSCORE, with the difference that step 4 in Section 5.2 of [RFC8613] is replaced with:

A. and then XOR with the X least significant bits of the Common IV, where X is the length in bits of the AEAD nonce.

4.4. external_aad

The external_aad of the Additional Authenticated Data (AAD) is different compared to OSCORE [RFC8613], and is defined in this section.

The same external_aad structure is used in group mode and pairwise mode for encryption/decryption (see Section 5.3 of [RFC9052]), as well as in group mode for computing and verifying the countersignature (see Sections 3.2 and 3.3 of [RFC9338]).

In particular, the external_aad includes also the Signature Algorithm, the Group Encryption Algorithm, the Pairwise Key Agreement Algorithm, the value of the 'kid context' in the COSE object of the request, the OSCORE option of the protected message, the sender's authentication credential, and the Group Manager's authentication credential.

The external_aad SHALL be a CBOR array wrapped in a bstr object as defined below, following the notation of [RFC8610]:

```
external_aad = bstr .cbor aad_array

aad_array = [
   oscore_version : uint,
   algorithms : [alg_aead : int / tstr / null,
                 alg_group_enc : int / tstr / null,
                 alg_signature : int / tstr / null,
                 alg_pairwise_key_agreement : int / tstr / null],
   request_kid : bstr,
   request_piv : bstr,
   options : bstr,
   request_kid_context : bstr,
   OSCORE_option : bstr,
   sender_cred : bstr,
   gm_cred : bstr
]
```

Figure 3: external_aad

Compared with Section 5.4 of [RFC8613], the aad_array has the following differences.

* The 'algorithms' array is extended as follows.

  The parameter 'alg_aead' MUST be set to the CBOR simple value null (0xf6) if the parameter AEAD Algorithm in the Security Context is not set (see Section 2). Otherwise, regardless of whether the endpoint supports the pairwise mode or not, this parameter MUST specify AEAD Algorithm from the Common Context (see Section 2.1.1) as per Section 5.4 of [RFC8613].

  Furthermore, the 'algorithms' array additionally includes:

  - 'alg_group_enc', which specifies Group Encryption Algorithm from the Common Context (see Section 2.1.6). This parameter MUST be set to the CBOR simple value null (0xf6) if the parameter Group Encryption Algorithm in the Common Context is not set. Otherwise, regardless of whether the endpoint supports the group mode or not, this parameter MUST specify Group Encryption Algorithm as a CBOR integer or text string, consistently with the "Value" field in the "COSE Algorithms" Registry for this algorithm.

  - 'alg_signature', which specifies Signature Algorithm from the Common Context (see Section 2.1.7). This parameter MUST be set to the CBOR simple value null (0xf6) if the parameter Signature Algorithm in the Common Context is not set. Otherwise, regardless of whether the endpoint supports the group mode or

not, this parameter MUST specify Signature Algorithm as a CBOR integer or text string, consistently with the "Value" field in the "COSE Algorithms" Registry for this algorithm.

- 'alg_pairwise_key_agreement', which specifies Pairwise Key Agreement Algorithm from the Common Context (see Section 2.1.9). This parameter MUST be set to the CBOR simple value null (0xf6) if Pairwise Key Agreement Algorithm in the Common Context is not set. Otherwise, regardless of whether the endpoint supports the pairwise mode or not, this parameter MUST specify Pairwise Key Agreement Algorithm as a CBOR integer or text string, consistently with the "Value" field in the "COSE Algorithms" Registry for this algorithm.

* The new element 'request_kid_context' contains the value of the 'kid context' in the COSE object of the request (see Section 4.2).

  This enables endpoints to safely keep a long exchange active beyond a possible change of Gid (i.e., of ID Context), following a group rekeying (see Section 3.2). In fact, it ensures that every response within a long exchange cryptographically matches with only one request (i.e., the request associated with that long exchange), rather than with multiple requests that were protected with different keying material but share the same 'request_kid' and 'request_piv' values.

* The new element 'OSCORE_option', containing the value of the OSCORE Option present in the protected message, encoded as a binary string. This prevents the attack described in Section 13.7 when using the group mode, as further explained in Section 13.7.2.

  Note for implementation: this construction requires the OSCORE option of the message to be generated and finalized before computing the ciphertext of the COSE_Encrypt0 object (when using the group mode or the pairwise mode) and before calculating the countersignature (when using the group mode). Also, the aad_array needs to be large enough to contain the largest possible OSCORE option.

* The new element 'sender_cred', containing the sender's authentication credential. This parameter MUST be set to a CBOR byte string, which encodes the sender's authentication credential in its original binary representation made available to other endpoints in the group (see Section 2.4).

* The new element 'gm_cred', containing the Group Manager's authentication credential. This parameter MUST be set to a CBOR byte string, which encodes the Group Manager's authentication

credential in its original binary representation made available to other endpoints in the group (see Section 2.4). This prevents the attack described in Section 13.8.

5. OSCORE Header Compression

The OSCORE header compression defined in Section 6 of [RFC8613] is used for compactly encoding the COSE_Encrypt0 object specified in Section 4 of this document, with the following differences.

* When the Group OSCORE message is protected in group mode, the message payload SHALL encode the ciphertext of the COSE object, concatenated with the encrypted countersignature of the COSE object. That is:

  - The plain, original countersignature of the COSE object, namely SIGNATURE, is specified in the "Countersignature0 version 2" parameter within the 'unprotected' field of the COSE object (see Section 4.1).

  - The encrypted countersignature, namely ENC_SIGNATURE, is computed as

    ENC_SIGNATURE = SIGNATURE XOR KEYSTREAM

    where KEYSTREAM is derived as per Section 5.1.

* When the Group OSCORE message is protected in pairwise mode, the message payload SHALL encode the ciphertext of the COSE object.

* This document defines the usage of the sixth least significant bit, called "Group Flag", in the first byte of the OSCORE option containing the OSCORE flag bits. This flag bit is specified in Section 14.1.

* The Group Flag MUST be set to 1 if the Group OSCORE message is protected using the group mode (see Section 8).

* The Group Flag MUST be set to 0 if the Group OSCORE message is protected using the pairwise mode (see Section 9). The Group Flag MUST also be set to 0 for ordinary OSCORE messages processed according to [RFC8613].

5.1. Keystream Derivation for Countersignature Encryption

The following defines how an endpoint derives the keystream KEYSTREAM, used to encrypt/decrypt the countersignature of an outgoing/incoming message M protected in group mode.

The keystream SHALL be derived as follows, by using the HKDF Algorithm from the Common Context (see Section 3.2 of [RFC8613]), which consists of composing the HKDF-Extract and HKDF-Expand steps [RFC5869].

KEYSTREAM = HKDF(salt, IKM, info, L)

The input parameters of HKDF are as follows.

* salt takes as value the Partial IV (PIV) used to protect M. Note that, if M is a response, salt takes as value either: i) the fresh Partial IV generated by the server and included in the response; or ii) the same Partial IV of the request generated by the client and not included in the response.

* IKM is the Signature Encryption Key from the Common Context (see Section 2.1.8).

* info is the serialization of a CBOR array consisting of (the notation follows [RFC8610]):

```
info = [
  id : bstr,
  id_context : bstr,
  type : bool,
  L : uint
]
```

where:

* id is the Sender ID of the endpoint that generated PIV.

* id_context is the ID Context (Gid) used when protecting M.

  Note that, in case of group rekeying, a server might use a different Gid when protecting a response, compared to the Gid that it used to verify (that the client used to protect) the request, see Section 8.3.

* type is the CBOR simple value true (0xf5) if M is a request, or the CBOR simple value false (0xf4) otherwise.

* L is the size of the countersignature, as per Signature Algorithm from the Common Context (see Section 2.1.7), in bytes.

5.2. Examples of Compressed COSE Objects

This section covers a list of OSCORE Header Compression examples of Group OSCORE used in group mode (see Section 5.2.1) or in pairwise mode (see Section 5.2.2).

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 5 and divided into two parts, since the object is transported in two CoAP fields: OSCORE option and payload.

The examples assume that the plaintext (see Section 5.3 of [RFC8613]) is 6 bytes long, and that the AEAD tag is 8 bytes long, hence resulting in a ciphertext which is 14 bytes long. When using the group mode, the COSE_Countersignature0 byte string as described in Section 4 is assumed to be 64 bytes long.

5.2.1. Examples in Group Mode

Request with ciphertext = 0xaea0155667924dff8a24e4cb35b9, kid = 0x25, Partial IV = 5 and kid context = 0x44616c.

* Before compression (96 bytes):

```
[
  / protected / h'',
  / unprotected / {
    / kid /                          4 : h'25',
    / Partial IV /                   6 : h'05',
    / kid context /                 10 : h'44616c',
    / Countersignature0 version 2 /  12 : h'66e6d9b0
    db009f3e105a673f8855611726caed57f530f8cae9d0b168
    513ab949fedc3e80a96ebe94ba08d3f8d3bf83487458e2ab
    4c2f936ff78b50e33c885e35'
  },
  / ciphertext / h'aea0155667924dff8a24e4cb35b9'
]
```

* After compression (85 bytes):

```
Flag byte: 0b00111001 = 0x39 (1 byte)

Option Value: 0x39 05 03 44 61 6c 25 (7 bytes)

Payload: 0xaea0155667924dff8a24e4cb35b9 de9e ... f1
(14 bytes + size of the encrypted countersignature)
```

Response with ciphertext = 0x60b035059d9ef5667c5a0710823b, kid = 0x52 and no Partial IV.

* Before compression (88 bytes):

```
[
  / protected / h'',
  / unprotected / {
    / kid /                          4 : h'52',
    / Countersignature0 version 2 /  12 : h'f5b659b8
    24487eb349c5c5c8a3fe401784cade2892725438e8be0fab
    daa2867ee6d29f68edb0818e50ebf98c28b923d0205f5162
    e73662e27c1a3ec562a49b80'
  },
  / ciphertext / h'60b035059d9ef5667c5a0710823b'
]
```

* After compression (80 bytes):

```
Flag byte: 0b00101000 = 0x28 (1 byte)

Option Value: 0x28 52 (2 bytes)

Payload: 0x60b035059d9ef5667c5a0710823b ca1e ... b3
(14 bytes + size of the encrypted countersignature)
```

5.2.2. Examples in Pairwise Mode

Request with ciphertext = 0xaea0155667924dff8a24e4cb35b9, kid = 0x25, Partial IV = 5 and kid context = 0x44616c.

* Before compression (29 bytes):

```
[
  / protected / h'',
  / unprotected / {
    / kid /           4 : h'25',
    / Partial IV /    6 : h'05',
    / kid context /  10 : h'44616c'
  },
  / ciphertext / h'aea0155667924dff8a24e4cb35b9'
]
```

* After compression (21 bytes):

  Flag byte: 0b00011001 = 0x19 (1 byte)

  Option Value: 0x19 05 03 44 61 6c 25 (7 bytes)

  Payload: 0xaea0155667924dff8a24e4cb35b9 (14 bytes)

Response with ciphertext = 0x60b035059d9ef5667c5a0710823b and no Partial IV.

* Before compression (18 bytes):

```
[
  / protected / h'',
  / unprotected / {},
  / ciphertext / h'60b035059d9ef5667c5a0710823b'
]
```

* After compression (14 bytes):

  Flag byte: 0b00000000 = 0x00 (1 byte)

  Option Value: 0x (0 bytes)

  Payload: 0x60b035059d9ef5667c5a0710823b (14 bytes)

6. Message Binding, Sequence Numbers, Freshness, and Replay Protection

Like OSCORE, Group OSCORE provides message binding of responses to requests, as well as uniqueness of AEAD (key, nonce) pair (see Sections 7.1 and 7.2 of [RFC8613], respectively).

6.1. Supporting Multiple Responses in Long Exchanges

For each of its ongoing long exchange, a client maintains one Response Number for each different server. Then, separately for each server, the client uses the associated Response Number to perform ordering and replay protection of responses received from that server within that long exchange (see Section 6.3.1).

That is, the Response Number has the same purpose that the Notification Number has in OSCORE (see Section 4.1.3.5.2 of [RFC8613]), but a client uses it for handling any response from the associated server within a long exchange.

Group OSCORE allows to preserve a long exchange active indefinitely, even in case the group is rekeyed, with consequent change of ID Context, or in case the client obtains a new Sender ID.

As defined in Section 8, this is achieved by the client and server(s) storing the 'kid' and 'kid context' used in the original request, throughout the whole duration of the long exchange.

Upon leaving the group or before re-joining the group, a group member MUST terminate all the ongoing long exchanges that it has started in the group as a client, and hence frees up the CoAP Token associated with the corresponding request.

6.2. Synchronization and Freshness

If the application requires freshness, e.g., according to time- or event-based policies (see Section 2.5.1 of [RFC9175]), a server can use the approach in Section 10 as a variant of the procedure in Appendix B.1.2 of [RFC8613], before delivering request messages from a client to the application. This also makes the server (re-)synchronize with the client's Sender Sequence Number.

Like in OSCORE [RFC8613], assuming an honest server, the message binding guarantees that a response is not older than the request it replies to. Therefore, building on Section 7.3 of [RFC8613], the following properties hold for Group OSCORE.

* The freshness of a response can be assessed if it is received soon after the request.

  For responses within a long exchange, this assessment gets weaker with time. If such responses are Observe notifications [RFC7641], it is RECOMMENDED that the client regularly re-register the observation.

If the request was neither a group request nor an Observe request, there is at most a single valid response and only from one, individually targeted server in the group. Thus, freshness can be assessed depending on when the request was sent.

* It is not guaranteed that a misbehaving server did not create the response before receiving the request, i.e., Group OSCORE does not verify server aliveness.

* For requests and responses, the received Partial IV allows a recipient to determine the relative order of requests or responses.

6.3. Replay Protection

Like in OSCORE [RFC8613], the replay protection relies on the Partial IV of incoming messages. A server updates the Replay Window of its Recipient Contexts based on the Partial IV values in received request messages, which correspond to the Sender Sequence Numbers of the clients. Note that there can be large jumps in these Sender Sequence Number values, for example when a client exchanges unicast messages with other servers. The operation of validating the Partial IV and performing replay protection MUST be atomic. The update of Replay Windows is described in Section 2.6.1.

The protection from replay of requests is performed as per Section 7.4 of [RFC8613], separately for each client and by leveraging the Replay Window in the corresponding Recipient Context. The protection from replay of responses in a long exchange is performed as defined in Section 6.3.1.

6.3.1. Replay Protection of Responses

A client uses the method defined in this section in order to check whether a received response is a replay.

This especially applies to responses received within a long exchange, during which multiple such responses can be received from the same server to the corresponding request. These include Observe notifications [RFC7641]; and non-notification responses as a reply to a group request, which the client can receive until the CoAP Token value associated with the group request is freed up (see Section 3.1.6 of [I-D.ietf-core-groupcomm-bis]).

When sending a response (both successful and error), a server MUST include its Sender Sequence Number as Partial IV in the response, except when sending the first response to the corresponding request, in which case the Partial IV in the response MAY be omitted.

In order to protect against replay, the client SHALL maintain for each ongoing long exchange one Response Number for each different server. The Response Number is a non-negative integer containing the largest Partial IV of the responses received from that server during the long exchange, while using the same Security Context.

Then, separately for each server, the client uses the associated Response Number to perform ordering and replay protection of the responses from that server during the long exchange, by comparing their Partial IVs with one another and against the Response Number.

For every long exchange, the Response Number associated with a server is initialized to the Partial IV of the response from that server such that, within the long exchange, it is the first response from that server to include a Partial IV and to be successfully verified with the used Security Context. Note that, when a new Security Context is established in the group, the client sets the Response Number of each associated server as not initialized (see Section 2.6.3.2), hence later responses within the same long exchange and protected with the new Security Context will result in a new initialization of Response Numbers. Furthermore, for every long exchange, a client MUST only accept at most one response without Partial IV from each server, and treat it as the oldest response from that server within that long exchange.

During a long exchange, a client receiving a response containing a Partial IV SHALL compare the Partial IV with the Response Number associated with the replying server within that long exchange. The client MUST stop processing a response from a server, if that response has a Partial IV that has been previously received from that server during that long exchange, while using the same Security Context. Applications MAY decide that a client only processes responses within a long exchange if those have a greater Partial IV than the Response Number associated with the replying server within that long exchange.

If the verification of the response succeeds, and the received Partial IV (when included) was greater than the Response Number associated with the replying server, then the client SHALL overwrite that Response Number with the received Partial IV.

As long as a server uses the same Security Context to protect its responses to the same request, the client MUST consider the response with the highest Partial IV as the freshest response from that server among those protected with that Security Context, regardless of the order of arrival. Within a long exchange, implementations need to make sure that a response without Partial IV is considered the oldest response from the replying server within that long exchange.

The method defined in this section is not relevant for responses to requests that are neither group requests nor Observe requests. In fact, for each of such requests, there is at most one response and only from one, individually targeted server in the group.

7. Message Reception

Upon receiving a protected message, a recipient endpoint retrieves a Security Context as in [RFC8613]. An endpoint MUST be able to distinguish between a Security Context to process OSCORE messages as in [RFC8613] and a Group OSCORE Security Context to process Group OSCORE messages as defined in this document.

To this end, an endpoint can take into account the different structure of the Security Context defined in Section 2, for example based on the presence of Signature Algorithm and Pairwise Key Agreement Algorithm in the Common Context. Alternatively, implementations can use an additional parameter in the Security Context, to explicitly mark that it is intended for processing Group OSCORE messages.

If either of the following conditions holds, a recipient endpoint MUST discard the incoming protected message:

* The Group Flag is set to 0 and the retrieved Security Context is associated with an OSCORE group, but the endpoint does not support the pairwise mode or any of the following algorithms is not set in the Security Context: the AEAD Algorithm and the Pairwise Key Agreement Algorithm.

* The Group Flag is set to 1 and the retrieved Security Context is associated with an OSCORE group, but the endpoint does not support the group mode or any of the following algorithms is not set in the Security Context: the Group Encryption Algorithm and the Signature Algorithm.

* The Group Flag is set to 1 but there is no Security Context associated with an OSCORE group.

  Future specifications may define how to process incoming messages protected with Security Contexts as in [RFC8613], when the Group Flag bit is set to 1.

Otherwise, if a Security Context associated with an OSCORE group is retrieved, the recipient endpoint processes the message with Group OSCORE, using the group mode (see Section 8) if the Group Flag is set to 1, or the pairwise mode (see Section 9) if the Group Flag is set to 0.

Note that, if the Group Flag is set to 0, and the recipient endpoint retrieves a Security Context which is valid to process the message but is not associated with an OSCORE group, then the message is processed according to [RFC8613].

8. Message Processing in Group Mode

When using the group mode, messages are protected and processed as specified in [RFC8613], with the modifications described in this section. The security objectives of the group mode are discussed in Appendix A.2.

The possible use of the group mode is indicated by the Group Manager as part of the group data provided to candidate group members when joining the group, according to which the parameters Group Encryption Algorithm and Signature Algorithm in the Security Context are set (see Section 2).

During all the steps of the message processing, an endpoint MUST use the same Security Context for the considered group. That is, an endpoint MUST NOT install a new Security Context for that group (see Section 2.6.3.2) until the message processing is completed.

The group mode SHOULD be used to protect group requests intended for multiple recipients or for the whole group. For an example where this is not fulfilled, see [I-D.amsuess-core-cachable-oscore]. This applies to both requests directly addressed to multiple recipients, e.g., sent by the client over multicast, as well as requests sent by the client over unicast to a proxy, that forwards them to the intended recipients over multicast [I-D.ietf-core-groupcomm-bis].

As per [RFC7252][I-D.ietf-core-groupcomm-bis], group requests sent over multicast MUST be Non-confirmable, and thus are not retransmitted by the CoAP messaging layer. Instead, applications should store such outgoing messages for a predefined, sufficient amount of time, in order to correctly perform potential retransmissions at the application layer. If performed, these retransmissions are repetitions of previous protected messages, which the sender endpoint does not protect again with Group OSCORE.

According to Section 5.2.3 of [RFC7252], responses to Non-confirmable group requests SHOULD also be Non-confirmable, but endpoints MUST be prepared to receive Confirmable responses in reply to a Non-confirmable group request. Confirmable group requests are acknowledged when sent over non-multicast transports, as specified in [RFC7252].

Furthermore, endpoints in the group locally perform error handling and processing of invalid messages according to the same principles adopted in [RFC8613]. In addition, a recipient MUST stop processing and reject any message that is malformed and that does not follow the format specified in Section 4 of this document, or that is not cryptographically validated in a successful way as per the processing defined in Section 8.2 and Section 8.4 of this document.

In either case, it is RECOMMENDED that a server does not send back any error message in reply to a received request, if any of the two following conditions holds:

* The server is not able to identify the received request as a group request, i.e., as sent to all servers in the group.

* The server identifies the received request as a group request.

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 network.

8.1. Protecting the Request

When using the group mode to protect a request, a client SHALL proceed as described in Section 8.1 of [RFC8613], with the following modifications.

* In step 2, the Additional Authenticated Data is modified as described in Section 4 of this document.

* In step 4, the encryption of the COSE object is modified as described in Section 4 of this document. The encoding of the compressed COSE object is modified as described in Section 5 of this document. In particular, the Group Flag MUST be set to 1. The Group Encryption Algorithm from the Common Context MUST be used.

* In step 5, the countersignature is computed and the format of the OSCORE message is modified as described in Section 4 and Section 5 of this document. In particular, the payload of the Group OSCORE message includes also the encrypted countersignature.

In addition, the following applies when sending a request that establishes a long exchange.

* If the client intends to keep the long exchange active beyond a possible change of Sender ID, the client MUST store the value of the 'kid' parameter from the request, and retain it until the long

exchange is terminated. Even in case the client is individually rekeyed and receives a new Sender ID from the Group Manager (see Section 2.6.3.1), the client MUST NOT update the stored 'kid' parameter value associated with the long exchange and the corresponding request.

* If the client intends to keep the long exchange active beyond a possible change of ID Context following a group rekeying (see Section 3.2), then the following applies.

  - The client MUST store the value of the 'kid context' parameter from the request, and retain it until the long exchange is terminated. Upon establishing a new Security Context with a new Gid as ID Context (see Section 2.6.3.2), the client MUST NOT update the stored 'kid context' parameter value associated with the long exchange and the corresponding request.

  - The client MUST store an invariant identifier of the group, which is immutable even in case the Security Context of the group is re-established. For example, this invariant identifier can be the "group name" in [I-D.ietf-ace-key-groupcomm-oscore], where it is used for joining the group and retrieving the current group keying material from the Group Manager.

    After a group rekeying, such an invariant information makes it simpler for the client to retrieve the current group keying material from the Group Manager, in case the client has missed both the rekeying messages and the first response protected with the new Security Context (see Section 8.3).

8.2. Verifying the Request

Upon receiving a protected request with the Group Flag set to 1, following the procedure in Section 7, a server SHALL proceed as described in Section 8.2 of [RFC8613], with the following modifications.

* In step 2, the decoding of the compressed COSE object follows Section 5 of this document. In particular:

  - If the server discards the request due to not retrieving a Security Context associated with the OSCORE group, the server MAY respond with a 4.01 (Unauthorized) error message. When doing so, the server MAY set an Outer Max-Age option with value zero, and MAY include a descriptive string as diagnostic payload.

- If the received 'kid context' matches an existing ID Context (Gid) but the received 'kid' does not match any Recipient ID in this Security Context, then the server MAY create a new Recipient Context for this Recipient ID and initialize it according to Section 3 of [RFC8613], and also retrieve the authentication credential associated with the Recipient ID to be stored in the new Recipient Context. Such a configuration is application specific. If the application does not specify dynamic derivation of new Recipient Contexts, then the server SHALL stop processing the request.

* In step 4, the Additional Authenticated Data is modified as described in Section 4 of this document.

* In step 6, the server also verifies the countersignature, by using the public key from the client's authentication credential stored in the associated Recipient Context. In particular:

  - If the server does not have the public key of the client yet, the server MUST stop processing the request and MAY respond with a 5.03 (Service Unavailable) response. The response MAY include a Max-Age Option, indicating to the client the number of seconds after which to retry. If the Max-Age Option is not present, a retry time of 60 seconds will be assumed by the client, as default value defined in Section 5.10.5 of [RFC7252].

  - The server MUST perform signature verification before decrypting the COSE object, as defined below. Implementations that cannot perform the two steps in this order MUST ensure that no access to the plaintext is possible before a successful signature verification and MUST prevent any possible leak of time-related information that can yield side-channel attacks.

  - The server retrieves the encrypted countersignature ENC_SIGNATURE from the message payload, and computes the original countersignature SIGNATURE as

    SIGNATURE = ENC_SIGNATURE XOR KEYSTREAM

    where KEYSTREAM is derived as per Section 5.1.

  - The server verifies the original countersignature SIGNATURE as described in Sections 3.2 and 3.3 of [RFC9338], by using the client's public key and according to the Signature Algorithm in the Security Context.

In particular, the Countersign_structure contains the context text string "CounterSignature0", the external_aad as defined in Section 4.4 of this document, and the ciphertext of the COSE object as payload.

- If the signature verification fails, the server SHALL stop processing the request, SHALL NOT update the Replay Window, and MAY respond with a 4.00 (Bad Request) response. Such a response MAY include an Outer Max-Age option with value zero, and its diagnostic payload MAY contain a string, which, if present, MUST be "Decryption failed" as if the decryption of the COSE object had failed.

- When decrypting the COSE object using the Recipient Key, the Group Encryption Algorithm from the Common Context MUST be used.

* Additionally, if the used Recipient Context was created upon receiving this request and the message is not verified successfully, the server MAY delete that Recipient Context. Such a configuration, which is specified by the application, mitigates attacks that aim at overloading the server's storage.

A server SHOULD NOT process a request if the received Recipient ID ('kid') is equal to its own Sender ID in its own Sender Context. For an example where this is not fulfilled, see Section 9.2.1 of [I-D.ietf-core-observe-multicast-notifications].

In addition, the following applies if the request establishes a long exchange and the server intends to reply with multiple responses.

* The server MUST store the value of the 'kid' parameter from the request, and retain it until the last response has been sent. The server MUST NOT update the stored value of the 'kid' parameter associated with the request, even in case the client is individually rekeyed and starts using a new Sender ID received from the Group Manager (see Section 2.6.3.1).

* The server MUST store the value of the 'kid context' parameter from the request, and retain it until the last response has been sent, i.e., beyond a possible change of ID Context following a group rekeying (see Section 3.2). That is, upon establishing a new Security Context with a new Gid as ID Context (see Section 2.6.3.2), the server MUST NOT update the stored value of a 'kid context' parameter associated with the request.

8.3. Protecting the Response

When using the group mode to protect a response, a server SHALL proceed as described in Section 8.3 of [RFC8613], with the following modifications.

Note that the server always protects a response with the Sender Context from its latest Security Context, and that establishing a new Security Context resets the Sender Sequence Number to 0 (see Section 3.2).

* In step 2, the Additional Authenticated Data is modified as described in Section 4 of this document.

  In addition, the following applies if the server intends to reply with multiple responses, within the long exchange established by the corresponding request.

  - The server MUST use the stored value of the 'kid' parameter from the request (see Section 8.2), as value for the 'request_kid' parameter in the external_aad structure (see Section 4.4).

  - The server MUST use the stored value of the 'kid context' parameter from the request (see Section 8.2), as value for the 'request_kid_context' parameter in the external_aad structure (see Section 4.4).

* In step 3, if any of the following two conditions holds, the server MUST include its Sender Sequence Number as Partial IV in the response and use it to build the AEAD nonce to protect the response. This prevents the server from reusing the AEAD nonce from the request together with the same encryption key.

  - The response is not the first response that the server sends to the request.

  - The server is using a different Security Context for the response compared to what was used to verify the request (see Section 3.2).

* In step 4, the encryption of the COSE object is modified as described in Section 4 of this document. The encoding of the compressed COSE object is modified as described in Section 5 of this document. In particular, the Group Flag MUST be set to 1. The Group Encryption Algorithm from the Common Context MUST be used.

In addition, the following applies.

- If the server is using a different ID Context (Gid) for the response compared to what was used to verify the request (see Section 3.2) and this is the first response from the server to that request, then the new ID Context MUST be included in the 'kid context' parameter of the response.

- The server may be replying to a request that was protected with an old Security Context. After completing the establishment of a new Security Context, the server MUST protect all the responses to that request with the Sender Context of the new Security Context.

  For each ongoing long exchange, the server can help the client to synchronize, by including also the 'kid context' parameter in responses following a group rekeying, with value set to the ID Context (Gid) of the new Security Context.

  If there is a known upper limit to the duration of a group rekeying, the server SHOULD include the 'kid context' parameter during that time. Otherwise, the server SHOULD include it until the Max-Age has expired for the last response sent before the installation of the new Security Context.

- The server can obtain a new Sender ID from the Group Manager, when individually rekeyed (see Section 2.6.3.1) or when re-joining the group. In such a case, the server can help the client to synchronize, by including the 'kid' parameter in a response protected in group mode, even when the request was protected in pairwise mode (see Section 9.3).

  That is, when responding to a request protected in pairwise mode, the server SHOULD include the 'kid' parameter in a response protected in group mode, if it is replying to that client for the first time since the assignment of its new Sender ID.

* In step 5, the countersignature is computed and the format of the OSCORE message is modified as described in Section 4 and Section 5 of this document. In particular the payload of the Group OSCORE message includes also the encrypted countersignature (see Section 5).

8.4. Verifying the Response

Upon receiving a protected response with the Group Flag set to 1, following the procedure in Section 7, a client SHALL proceed as described in Section 8.4 of [RFC8613], with the modifications described in this section.

Note that a client may receive a response protected with a Security Context different from the one used to protect the corresponding request, and that, upon the establishment of a new Security Context, the client re-initializes its Replay Windows in its Recipient Contexts (see Section 3.2).

* In step 2, the decoding of the compressed COSE object is modified as described in Section 5 of this document. In particular, a 'kid' may not be present, if the response is a reply to a request protected in pairwise mode. In such a case, the client assumes the response 'kid' to be the Recipient ID for the server to which the request protected in pairwise mode was intended for.

  If the response 'kid context' matches an existing ID Context (Gid) but the received/assumed 'kid' does not match any Recipient ID in this Security Context, then the client MAY create a new Recipient Context for this Recipient ID and initialize it according to Section 3 of [RFC8613], and also retrieve the authentication credential associated with the Recipient ID to be stored in the new Recipient Context. If the application does not specify dynamic derivation of new Recipient Contexts, then the client SHALL stop processing the response.

* In step 3, the Additional Authenticated Data is modified as described in Section 4 of this document.

  In addition, the following applies if the client processes a response to a request within a long exchange.

  - The client MUST use the stored value of the 'kid' parameter from the request (see Section 8.1), as value for the 'request_kid' parameter in the external_aad structure (see Section 4.4).

  - The client MUST use the stored value of the 'kid context' parameter from the request (see Section 8.1), as value for the 'request_kid_context' parameter in the external_aad structure (see Section 4.4).

This ensures that, throughout a long exchange, the client can correctly verify the received responses, even in case the client is individually rekeyed and starts using a new Sender ID received from the Group Manager (see Section 2.6.3.1), as well as when it installs a new Security Context with a new ID Context (Gid) following a group rekeying (see Section 3.2).

* In step 5, the client also verifies the countersignature, by using the public key from the server's authentication credential stored in the associated Recipient Context. In particular:

  - The client MUST perform signature verification as defined below, before decrypting the COSE object. Implementations that cannot perform the two steps in this order MUST ensure that no access to the plaintext is possible before a successful signature verification and MUST prevent any possible leak of time-related information that can yield side-channel attacks.

  - The client retrieves the encrypted countersignature ENC_SIGNATURE from the message payload, and computes the original countersignature SIGNATURE as

    SIGNATURE = ENC_SIGNATURE XOR KEYSTREAM

    where KEYSTREAM is derived as per Section 5.1.

    The client verifies the original countersignature SIGNATURE.

  - If the verification of the countersignature fails, the client: i) SHALL stop processing the response; and ii) SHALL NOT update the Response Number associated with the server.

  - After a successful verification of the countersignature, the client performs also the following actions in case the request was protected in pairwise mode (see Section 9.3).

    o If the 'kid' parameter is present in the response, the client checks whether this received 'kid' is equal to the expected 'kid', i.e., the known Recipient ID for the server to which the request was intended for.

o If the 'kid' parameter is not present in the response, the client checks whether the server that has sent the response is the same one to which the request was intended for. This can be done by checking that the public key used to verify the countersignature of the response is equal to the public key included in the authentication credential Recipient Auth Cred, which was taken as input to derive the Pairwise Sender Key used for protecting the request (see Section 2.5.1).

In either case, if the client determines that the response has come from a different server than the expected one, then the client: i) SHALL discard the response and SHALL NOT deliver it to the application; ii) SHALL NOT update the Response Number associated with the server.

Otherwise, the client hereafter considers the received 'kid' as the current Recipient ID for the server.

* In step 5, when decrypting the COSE object using the Recipient Key, the Group Encryption Algorithm from the Common Context MUST be used.

  In addition, the client performs the following actions if the response is received within a long exchange.

  - The ordering and the replay protection of responses received from the server during the long exchange are performed as per Section 6.3.1 of this document, by using the Response Number associated with that server within that long exchange. In case of unsuccessful decryption and verification of a response, the client SHALL NOT update the Response Number associated with the server.

  - When receiving the first valid response from the server within the long exchange, the client MUST store the current kid "kid1" of that server for that long exchange. If the 'kid' field is included in the OSCORE option of the response, its value specifies "kid1". If the request was protected in pairwise mode (see Section 9.3), the 'kid' field may not be present in the OSCORE option of the response (see Section 4.2). In this case, the client assumes "kid1" to be the Recipient ID for the server to which the request was intended for.

- When receiving another valid response to the same request from the same server - which can be identified and recognized through the same public key used to verify the countersignature and included in the server's authentication credential - the client determines the current kid "kid2" of the server as above for "kid1", and MUST check whether "kid2" is equal to the stored "kid1".

  If "kid1" and "kid2" are different, the client SHOULD NOT accept the response as valid to be delivered to the application, and SHOULD NOT update the Response Number associated with the server. Exceptions can apply if the client has a means of preserving the order of responses in a more elaborate way out of the scope of this document, or the client application does not require response ordering altogether.

Note that, if "kid2" is different from "kid1" and the 'kid' field is omitted from the response - which is possible if the request was protected in pairwise mode - then the client will compute a wrong keystream to decrypt the countersignature (i.e., by using "kid1" rather than "kid2" in the 'id' field of the 'info' array in Section 5.1), thus subsequently failing to verify the countersignature and discarding the response.

This ensures that the client remains able to correctly perform the ordering and replay protection of responses within a long exchange, even in case the server legitimately starts using a new Sender ID, as received from the Group Manager when individually rekeyed (see Section 2.6.3.1) or when re-joining the group.

* In step 8, 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, mitigates attacks that aim at overloading the client's storage.

8.5. External Signature Checkers

When receiving a message protected in group mode, a signature checker (see Section 3.3) proceeds as follows.

* The signature checker retrieves the encrypted countersignature ENC_SIGNATURE from the message payload, and computes the original countersignature SIGNATURE as

  SIGNATURE = ENC_SIGNATURE XOR KEYSTREAM

  where KEYSTREAM is derived as per Section 5.1.

* The signature checker verifies the original countersignature SIGNATURE, by using the public key of the sender endpoint as included in that endpoint's authentication credential. The signature checker determines the right authentication credential based on the ID Context (Gid) and the Sender ID of the sender endpoint.

Note that the following applies when attempting to verify the countersignature of a response message.

* The response may not include a Partial IV and/or an ID Context. In such a case, the signature checker considers the same values from the corresponding request, i.e., the request matching with the response by CoAP Token value.

* The response may not include a Sender ID. This can happen when the response protected in group mode matches a request protected in pairwise mode (see Section 9.1), with a case in point provided by [I-D.amsuess-core-cachable-oscore]. In such a case, the signature checker needs to use other means (e.g., source addressing information of the server endpoint) to identify the correct authentication credential including the public key to use for verifying the countersignature of the response.

The particular actions following a successful or unsuccessful verification of the countersignature are application specific and out of the scope of this document.

9. Message Processing in Pairwise Mode

When using the pairwise mode of Group OSCORE, messages are protected and processed as in [RFC8613], with the modifications described in this section. The security objectives of the pairwise mode are discussed in Appendix A.2.

The possible use of the pairwise mode is indicated by the Group Manager as part of the group data provided to candidate group members when joining the group, according to which the parameters AEAD Algorithm and Pairwise Key Agreement Algorithm in the Security Context are set (see Section 2).

The pairwise mode takes advantage of an existing Security Context to establish keying material shared exclusively with any other member. For encryption and decryption operations in pairwise mode, the AEAD Algorithm from the Common Context is used (see Section 2.1.1).

In order to use the pairwise mode in a group where the group mode is also used (i.e., Group Encryption Algorithm and Signature Algorithm in the Security Context are set), the signature scheme of the group mode MUST support a combined signature and encryption scheme. For example, this can rely on signing operations using ECDSA, and encryption operations using AES-CCM with keying material derived through ECDH.

The pairwise mode does not support external verifiers of source authentication and message integrity like the group mode does (see Section 8.5).

An endpoint implementing only a silent server does not support the pairwise mode.

Endpoints using the CoAP Echo Option [RFC9175] and/or block-wise transfers [RFC7959] in a group where the AEAD Algorithm and Pairwise Key Agreement Algorithm are set MUST support the pairwise mode. This applies, for example, to block-wise exchanges after a first block-wise request which targets all servers in the group and includes the CoAP Block2 option (see Section 3.8 of [I-D.ietf-core-groupcomm-bis]). This prevents the attack described in Section 13.9, which leverages requests sent over unicast to a single group member and protected in group mode.

The pairwise mode cannot be used to protect messages intended for multiple recipients. In fact, the keying material used for the pairwise mode is shared only between two endpoints.

However, a sender can use the pairwise mode to protect a message sent to (but not intended for) multiple recipients, if interested in a response from only one of them. For instance, this is useful to support the address discovery service defined in Section 9.1, when a single 'kid' value is indicated in the payload of a request sent to multiple recipients, e.g., over multicast.

9.1. Pre-Conditions

In order to protect an outgoing message in pairwise mode, the sender needs to know the authentication credential and the Recipient ID for the recipient endpoint, as stored in the Recipient Context associated with that endpoint (see Section 2.5.4).

Furthermore, the sender needs to know the individual address of the recipient endpoint. This information may not be known at any given point in time. For instance, right after having joined the group, a client may know the authentication credential and Recipient ID for a given server, but not the addressing information required to reach it with an individual, one-to-one request.

In order to make addressing information of individual endpoints available, servers in the group MAY expose a resource to which a client can send a request targeting a set of servers, identified by their 'kid' values specified in the request payload, or implicitly if the request is sent in pairwise mode. Further details of such an interface are out of scope for this document.

9.2. Main Differences from OSCORE

The pairwise mode protects messages between two members of a group, essentially following [RFC8613], but with the following notable differences.

* The 'kid' and 'kid context' parameters of the COSE object are used as defined in Section 4.2 of this document.

* The external_aad defined in Section 4.4 of this document is used for the encryption and decryption process.

* The Pairwise Sender/Recipient Keys used as Sender/Recipient keys are derived as defined in Section 2.5 of this document.

9.3. Protecting the Request

When using the pairwise mode to protect a request, a client SHALL proceed as described in Section 8.1 of [RFC8613], with the differences summarized in Section 9.2 of this document.

Furthermore, when sending a request that establishes a long exchange, what is specified in Section 8.1 of this document holds, with respect to storing the value of the 'kid' and 'kid context' parameters, and to storing an invariant identifier of the group.

9.4. Verifying the Request

Upon receiving a protected request with the Group Flag set to 0, following the procedure in Section 7, a server SHALL proceed as described in Section 8.2 of [RFC8613], with the differences summarized in Section 9.2 of this document. The following differences also apply.

* If the server discards the request due to not retrieving a Security Context associated with the OSCORE group or to not supporting the pairwise mode, the server MAY respond with a 4.01 (Unauthorized) error message or a 4.02 (Bad Option) error message, respectively. When doing so, the server MAY set an Outer Max-Age option with value zero, and MAY include a descriptive string as diagnostic payload.

* If a new Recipient Context is created for this Recipient ID, new Pairwise Sender/Recipient Keys are also derived (see Section 2.5.1). The new Pairwise Sender/Recipient Keys are deleted if the Recipient Context is deleted as a result of the message not being successfully verified.

* What is specified in Section 8.2 of this document holds with respect to the following points.

  - The possible, dynamic creation and configuration of a Recipient Context upon receiving the request.

  - The possible deletion of a Recipient Context created upon receiving the request, in case the request is not verified successfully.

  - The rule about processing the request where the received Recipient ID ('kid') is equal to the server's Sender ID.

  - The storing of the value of the 'kid' and 'kid context' parameters from the request, if the server intends to reply with multiple responses within the long exchange established by the request.

9.5. Protecting the Response

When using the pairwise mode to protect a response, a server SHALL proceed as described in Section 8.3 of [RFC8613], with the differences summarized in Section 9.2 of this document. The following differences also apply.

* What is specified in Section 8.3 of this document holds with respect to the following points.

  - The protection of a response when using a different Security Context than the one used to protect the corresponding request. That is, the server always protects a response with the Sender Context from its latest Security Context, and establishing a new Security Context resets the Sender Sequence Number to 0 (see Section 3.2).

- The use of the stored value of the 'kid' and 'kid context' parameters, if the server intends to reply with multiple responses within the long exchange established by the request.

- The rules for the inclusion of the server's Sender Sequence Number as Partial IV in a response, as used to build the AEAD nonce to protect the response.

- The rules for the inclusion of the ID Context (Gid) in the 'kid context' parameter of a response, if the ID Context used for the response differs from the one used to verify the request (see Section 3.2), also for helping the client to synchronize.

- The rules for the inclusion of the Sender ID in the 'kid' parameter of a response to a request that was protected in pairwise mode, if the server has obtained a new Sender ID from the Group Manager when individually rekeyed (see Section 2.6.3.1), thus helping the client to synchronize.

9.6. Verifying the Response

Upon receiving a protected response with the Group Flag set to 0, following the procedure in Section 7, a client SHALL proceed as described in Section 8.4 of [RFC8613], with the differences summarized in Section 9.2 of this document. The following differences also apply.

* The client may receive a response protected with a Security Context different from the one used to protect the corresponding request. Also, upon the establishment of a new Security Context, the client re-initializes its Replay Windows in its Recipient Contexts (see Section 2.2).

* The same as described in Section 8.4 holds with respect to handling the 'kid' parameter of the response, when received as a reply to a request protected in pairwise mode. The client can also in this case check whether the replying server is the expected one, by relying on the server's public key. However, since the response is protected in pairwise mode, the public key is not used for verifying a countersignature as in Section 8.4. Instead, the expected server's authentication credential - namely Recipient Auth Cred and including the server's public key - was taken as input to derive the Pairwise Recipient Key used to decrypt and verify the response (see Section 2.5.1).

* If a new Recipient Context is created for this Recipient ID, new Pairwise Sender/Recipient Keys are also derived (see Section 2.5.1). The new Pairwise Sender/Recipient Keys are deleted if the Recipient Context is deleted as a result of the message not being successfully verified.

* What is specified in Section 8.4 of this document holds with respect to the following points.

  - The possible, dynamic creation and configuration of a Recipient Context upon receiving the response.

  - The use of the stored value of the 'kid' and 'kid context' parameters, when processing a response received within a long exchange.

  - The performing of ordering and replay protection for responses received within a long exchange.

  - The possible deletion of a Recipient Context created upon receiving the response, in case the response is not verified successfully.

10. Challenge-Response Based Freshness and Synchronization

This section describes how a server endpoint can verify freshness of a request and synchronize with Sender Sequence Numbers of client endpoints in the group. Similarly to what is defined in Appendix B.1.2 of [RFC8613], the server performs a challenge-response exchange with a client, by using the Echo Option for CoAP specified in Section 2 of [RFC9175].

Upon receiving a request from a particular client for the first time, the server processes the message as described in this document, but, even if valid, does not deliver it to the application. Instead, the server replies to the client with a Group OSCORE protected 4.01 (Unauthorized) response message, including only the Echo Option and no diagnostic payload. The server MUST use its Partial IV when generating the AEAD nonce for protecting the response conveying the Echo Option, and MUST include the Partial IV in the response.

The Echo option value SHOULD NOT be reused; if it is reused, it MUST be highly unlikely to have been recently used with this client. Since this response is protected with the Security Context used in the group, the client will consider the response valid upon successfully decrypting and verifying it.

The server stores the Echo Option value included in the response together with the pair (gid,kid), where 'gid' is the Group Identifier of the OSCORE group and 'kid' is the Sender ID of the client in the group. These are specified in the 'kid context' and 'kid' fields of the OSCORE Option of the request, respectively. After a group rekeying has been completed and a new Security Context has been established in the group, which results also in a new Group Identifier (see Section 3.2), the server MUST delete all the stored Echo values associated with members of the group.

Upon receiving a 4.01 (Unauthorized) response that includes an Echo Option and originates from a verified group member, the subsequent client request echoing the Echo Option value MUST be sent as a unicast message to the same server.

If in the group the AEAD Algorithm and Pairwise Key Agreement Algorithm are set in the Security Context, the client MUST use the pairwise mode to protect the request, as per Section 9.3. Note that, as defined in Section 9, endpoints that are members of such a group and that use the Echo Option support the pairwise mode. In a group where the AEAD Algorithm and Pairwise Key Agreement Algorithm are not set, only the group mode can be used. Hence, requests including the Echo option can be protected only with the Group Mode, with the caveat due to the risk for those requests to be redirected to a different server than the intended one, as discussed in Section 13.9.

The client does not necessarily resend the same request, but can instead send a more recent one, if the application permits it. This allows the client to not retain previously sent requests for full retransmission, unless the application explicitly requires otherwise. In either case, the client uses a fresh Sender Sequence Number value from its own Sender Context. If the client stores requests for possible retransmission with the Echo Option, it should not store a given request for longer than a preconfigured time interval. Note that the unicast request echoing the Echo Option is correctly treated and processed, 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 4).

Upon receiving the unicast request including the Echo Option, the server performs the following verifications.

* If the server does not store an Echo Option value for the pair (gid,kid), it considers: i) the time t1 when it has established the Security Context used to protect the received request; and ii) the time t2 when the request has been received. Since a valid request cannot be older than the Security Context used to protect it, the server verifies that (t2 - t1) is less than the largest amount of time acceptable to consider the request fresh.

* If the server stores an Echo Option value for the pair (gid,kid) associated with that same client in the same group, the server verifies that the option value equals that same stored value previously sent to that client.

If the verifications above fail, the server MUST NOT process the request further and MAY send a 4.01 (Unauthorized) response including an Echo Option, hence performing a new challenge-response exchange.

If the verifications above are successful, the server considers the Recipient Context associated with the sender client and proceeds as follows.

* In case the Replay Window is invalid, the steps below occur.

  1. The server updates the Replay Window, by marking as received the Sender Sequence Number from the latest received request. This becomes the lower limit of the Replay Window, while all the other values are marked as not received.

  2. The server makes the Replay Window valid, and accepts the request as fresh.

* In case the Replay Window is already valid, the server discards the verification result and accepts the request as fresh or treats it as a replay, according to the existing Replay Window.

A server should not deliver 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. A server may perform the challenge-response described above at any time, if synchronization with Sender Sequence Numbers of clients is lost, e.g., after a device reboot occurred in an unprepared way. A client has to be ready to perform the challenge-response based on the Echo Option if a server starts it.

Message freshness is further discussed in Section 13.14.

11. Implementation Compliance

Like in [RFC8613], HKDF SHA-256 is the mandatory to implement HKDF.

An endpoint may support only the group mode, or only the pairwise mode, or both.

For endpoints that support the group mode, the following applies.

* For endpoints that use authenticated encryption, the AEAD algorithm AES-CCM-16-64-128 defined in Section 4.2 of [RFC9053] is mandatory to implement as Group Encryption Algorithm (see Section 2.1.6).

* For many constrained IoT devices it is problematic to support more than one signature algorithm. Existing devices can be expected to support either EdDSA or ECDSA. In order to enable as much interoperability as we can reasonably achieve, the following applies with respect to the Signature Algorithm (see Section 2.1.7).

  Less constrained endpoints SHOULD implement both: the EdDSA signature algorithm together with the elliptic curve Ed25519 [RFC8032]; and the ECDSA signature algorithm together with the elliptic curve P-256.

  Constrained endpoints SHOULD implement: the EdDSA signature algorithm together with the elliptic curve Ed25519 [RFC8032]; or the ECDSA signature algorithm together with the elliptic curve P-256.

* Endpoints that implement the ECDSA signature algorithm MAY use "deterministic ECDSA" as specified in [RFC6979]. Pure deterministic elliptic-curve signature algorithms such as deterministic ECDSA and EdDSA have the advantage of not requiring access to a source of high-quality randomness. However, these signature algorithms have been shown vulnerable to some side-channel and fault injection attacks due to their determinism, which can result in extracting a device’s private key. As suggested in Section 2.1.1 of [RFC9053], this can be addressed by combining both randomness and determinism [I-D.irtf-cfrg-det-sigs-with-noise].

For endpoints that support the pairwise mode, the following applies.

* The AEAD algorithm AES-CCM-16-64-128 defined in Section 4.2 of [RFC9053] is mandatory to implement as AEAD Algorithm (see Section 2.1.1).

* The ECDH-SS + HKDF-256 algorithm specified in Section 6.3.1 of [RFC9053] is mandatory to implement as Pairwise Key Agreement Algorithm (see Section 2.1.9).

* In order to enable as much interoperability as we can reasonably achieve in the presence of constrained devices (see above), the following applies.

  Less constrained endpoints SHOULD implement both the X25519 curve [RFC7748] and the P-256 curve as ECDH curves.

  Constrained endpoints SHOULD implement the X25519 curve [RFC7748] or the P-256 curve as ECDH curve.

Constrained IoT devices may alternatively represent Montgomery curves and (twisted) Edwards curves [RFC7748] in the short-Weierstrass form Wei25519, with which the algorithms ECDSA25519 and ECDH25519 can be used for signature operations and Diffie-Hellman secret calculation, respectively [I-D.ietf-lwig-curve-representations].

12. Web Linking

The use of Group OSCORE or OSCORE [RFC8613] MAY be indicated by a target "gosc" attribute in a web link [RFC8288] to a resource, e.g., using a link-format document [RFC6690] if the resource is accessible over CoAP.

The "gosc" attribute is a hint indicating that the destination of that link is only accessible using Group OSCORE or 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 Group OSCORE or OSCORE.

A value MUST NOT be given for the "gosc" attribute; any present value MUST be ignored by parsers. The "gosc" attribute MUST NOT appear more than once in a given link-value; occurrences after the first MUST be ignored by parsers.

When a link-value includes the "gosc" attribute, the link-value MUST also include the "osc" attribute defined in Section 9 of [RFC8613]. If the endpoint parsing the link-value supports Group OSCORE and understands the "gosc" attribute, then the parser MUST ignore the "osc" attribute, which is overshadowed by the "gosc" attribute.

The example in Figure 4 shows a use of the "gosc" attribute: the client does resource discovery on a server and gets back a list of resources, one of which includes the "gosc" attribute indicating that the resource is protected with Group OSCORE or OSCORE. The link-format notation (see Section 5 of [RFC6690]) is used.

```
REQ: GET /.well-known/core

RES: 2.05 Content
     </sensors/temp>;gosc;osc,
     </sensors/light>;if="sensor"
```

Figure 4: Example of using the "gosc" attribute in a web link.

13. Security Considerations

The same threat model discussed for OSCORE in Appendix D.1 of [RFC8613] holds for Group OSCORE.

For Group OSCORE, the Sender Context and Recipient Context additionally contain asymmetric keys, which are used to provide source authentication: in group mode, by means of countersignatures (see Section 13.1); in pairwise mode, by using Diffie-Hellman (see Section 13.2). The key pair can, for example, be generated by the endpoint or provisioned during manufacturing.

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.

The same considerations on supporting Proxy operations discussed for OSCORE in Appendix D.2 of [RFC8613] hold for Group OSCORE.

The same considerations on protected message fields for OSCORE discussed in Appendix D.3 of [RFC8613] hold for Group OSCORE.

The same considerations on uniqueness of (key, nonce) pairs for OSCORE discussed in Appendix D.4 of [RFC8613] hold for Group OSCORE. This is further discussed in Section 13.3 of this document.

The same considerations on unprotected message fields for OSCORE discussed in Appendix D.5 of [RFC8613] hold for Group OSCORE, with the following differences. First, the 'kid context' of request

messages is part of the Additional Authenticated Data, thus safely enabling to keep long exchanges active beyond a possible change of ID Context (Gid), following a group rekeying (see Section 4.4). Second, the countersignature included in a Group OSCORE message protected in group mode is computed also over the value of the OSCORE option, which is also part of the Additional Authenticated Data used in the signing process. This is further discussed in Section 13.7 of this document.

As discussed in Section 6.2.3 of [I-D.ietf-core-groupcomm-bis], Group OSCORE addresses security attacks against CoAP listed in Sections 11.2-11.6 of [RFC7252], especially when run 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 (see Section 12 of [RFC8613]), and discusses how they hold when Group OSCORE is used.

13.1. Security of the Group Mode

The group mode defined in Section 8 relies on commonly shared group keying material to protect communication within a group. Using the group mode has the implications discussed below. The following refers to group members as the endpoints in the group storing the latest version of the group keying material.

* Source authentication of messages sent to a group is ensured through a countersignature, which is computed by the sender using its own private key and then appended to the message payload. Also, the countersignature is encrypted by using a keystream derived from the group keying material (see Section 5). This ensures group privacy, i.e., an attacker cannot track an endpoint over two groups by linking messages between the two groups, unless being also a member of those groups.

* 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.

* If the used Group Encryption Algorithm provides integrity protection, then it also ensures group authentication and proof of group membership, but not source authentication. That is, it ensures that a message sent to a group has been sent by a member of that group, but not necessarily by the alleged sender. In fact, any group member is able to derive the Sender Key used by the actual sender endpoint, and thus can compute a valid authentication tag. Therefore, the message content could originate from any of the current group members.

  Furthermore, if the used Group Encryption Algorithm does not provide integrity protection, then it does not ensure any level of message authentication or proof of group membership.

  On the other hand, proof of group membership is always ensured by construction through the strict management of the group keying material (see Section 3.2). That is, the group is rekeyed in case of members' leaving, and the current group members are informed of former group members. Thus, a current group member storing the latest group keying material does not store the authentication credential of any former group member.

  This allows a recipient endpoint to rely on the stored authentication credentials and public keys included therein, in order to always confidently assert the group membership of a sender endpoint when processing an incoming message, i.e., to assert that the sender endpoint was a group member when it signed the message. In turn, this prevents a former group member to possibly re-sign and inject in the group a stored message that was protected with old keying material.

  A case in point is a group where the Group Encryption Algorithm does not provide integrity protection; a group member leaves the group; and, after the group rekeying, associates with the group as external signature checker (see Section 8.5). When doing so, it obtains from the Group Manager the new Signature Encryption Key, from which it can derive keystreams for encrypting and decrypting the countersignatures of messages protected in group mode.

If, when participating in the group rekeying, the current group members had not deleted the Recipient Context and authentication credential of the former group member, then the signature checker would be able to successfully inject messages protected in group mode, as encrypted with the old group keying material, signed with its own private key, and with the countersignature encrypted by means of the latest Signature Encryption Key. Then, the group members, as still retaining the authentication credential of the signature checker, will verify and accept the message, even though the sender was not a group member when signing the message.

The security properties of the group mode are summarized below.

1. Asymmetric source authentication, by means of a countersignature.

2. Symmetric group authentication, by means of an authentication tag (only for Group Encryption Algorithms providing integrity protection).

3. Symmetric group confidentiality, by means of symmetric encryption.

4. Proof of group membership, by strictly managing the group keying material, as well as by means of integrity tags when using a Group Encryption Algorithm that provides also integrity protection.

5. Group privacy, by encrypting the countersignature.

The group mode fulfills the security properties above while also displaying the following benefits. First, the use of a Group Encryption Algorithm that does not provide integrity protection results in a minimal communication overhead, by limiting the message payload to the ciphertext without integrity tag together with the encrypted countersignature. Second, it is possible to deploy semi-trusted entities such as signature checkers (see Section 3.3), which can break property 5, but cannot break properties 1, 2, 3, and 4.

13.2. Security of the Pairwise Mode

The pairwise mode defined in Section 9 protects messages by using pairwise symmetric keys, derived from the static-static Diffie-Hellman shared secret computed from the asymmetric keys of the sender and recipient endpoint (see Section 2.5).

The used AEAD Algorithm MUST provide integrity protection. Therefore, the pairwise mode ensures both pairwise data-confidentiality and source authentication of messages, without using

countersignatures. Furthermore, the recipient endpoint achieves proof of group membership for the sender endpoint, since only current group members have the required keying material to derive a valid Pairwise Sender/Recipient Key.

The long-term storing of the Diffie-Hellman shared secret is a potential security issue. In fact, if the shared secret of two group members is leaked, a third group member can exploit it to impersonate any of those two group members, by deriving and using their pairwise key. The possibility of such leakage should be contemplated, as more likely to happen than the leakage of a private key, which could be rather protected at a significantly higher level than generic memory, e.g., by using a Trusted Platform Module. Therefore, there is a trade-off between the maximum amount of time a same shared secret is stored and the frequency of its re-computing.

13.3. Uniqueness of (key, nonce)

The proof for uniqueness of (key, nonce) pairs in Appendix D.4 of [RFC8613] is also valid in group communication scenarios. That is, given an OSCORE group:

* Uniqueness of Sender IDs within the group is enforced by the Group Manager. In fact, from the moment when a Gid is assigned to a group until the moment when a new Gid is assigned to that same group, the Group Manager does not reassign a Sender ID within the group (see Section 3.2).

* The case A in Appendix D.4 of [RFC8613] concerns all requests as well as all responses including a Partial IV (e.g., Observe notifications [RFC7641] or any other subsequent responses after the first one). In this case, same considerations from [RFC8613] apply here as well.

* The case B in Appendix D.4 of [RFC8613] concerns responses not including a Partial IV (e.g., a single response to a request). In this case, same considerations from [RFC8613] 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.

13.4. Management of Group Keying Material

The protocol described in this document 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 MUST be adopted.

[I-D.ietf-ace-key-groupcomm-oscore] specifies a simple rekeying scheme for renewing the Security Context in a group.

Alternative rekeying schemes that are more scalable with the group size may be needed in dynamic, large groups where endpoints can join and leave at any time, in order to limit the impact on performance due to the Security Context and keying material update.

13.5. 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.

This may result in a client using an old Security Context to protect a request, and a server using a different new Security Context to protect a corresponding response. As a consequence, clients may receive a response protected with a Security Context different from the one used to protect the corresponding request.

In particular, a server may first get a request protected with the old Security Context, then install the new Security Context, and only after that produce a response to send back to the client. In such a case, as specified in Section 8.3, the server MUST protect the potential response using the new Security Context. Specifically, the server MUST include its Sender Sequence Number as Partial IV in the response and use it to build the AEAD nonce to protect the response. This prevents the AEAD nonce from the request from being reused with the new Security Context.

The client will process that response using the new Security Context, provided that it has installed the new security parameters and keying material before the message processing.

In case block-wise transfer [RFC7959] is used, the same considerations from Section 10.3 of [I-D.ietf-ace-key-groupcomm] hold.

Furthermore, as described below, a group rekeying may temporarily result in misaligned Security Contexts between the sender and recipient of a same message.

13.5.1. Late Update on the Sender

In this 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, and is thus unable to correctly process it.

A possible way to ameliorate this issue is to preserve the old retained 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.

This makes particular sense when the recipient is a client, that would hence be able to process incoming responses protected with the old retained Security Context used to protect the associated request. If, as typically expected, the old Gid is not included in the response, then the client will first fail to process the response using the latest Security Context, and then use the old retained Security Context as a second attempt.

Instead, a recipient server can immediately process an incoming request with the old retained Security Context, as signaled by the old Gid that is always included in requests. However, the server would better and more simply discard such an incoming request.

This tolerance preserves the processing of secure messages throughout a long-lasting key rotation, as group rekeying processes may likely take a long time to complete, especially in large groups. On the other hand, a former (compromised) group member can abusively take advantage of this, and send messages protected with the old retained Security Context. Therefore, a conservative application policy should not admit the retention of old Security Contexts.

13.5.2. Late Update on the Recipient

In this case, the sender protects a message using the new Security Context, but the recipient receives that message before having installed the new Security Context. Therefore, the recipient would not be able to correctly process the message and hence discards it.

If the recipient installs the new Security Context shortly after that and the sender endpoint retransmits the message, 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.

13.6. 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.

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, AEAD keys are different among different groups.

In case multiple groups use the same IP multicast address, the entity assigning that address may help limiting the chances to experience such collisions of Group Identifiers. In particular, it may allow the Group Managers of those groups using the same IP multicast address to share their respective list of assigned Group Identifiers currently in use.

13.7. Cross-group Message Injection

A same endpoint is allowed to and would likely use the same pair (private key, authentication credential) in multiple OSCORE groups, possibly administered by different Group Managers.

When a sender endpoint sends a message protected in pairwise mode to a recipient endpoint in an OSCORE group, a malicious group member may attempt to inject the message to a different OSCORE group also including the same endpoints (see Section 13.7.1).

This practically relies on altering the content of the OSCORE option, and having the same MAC in the ciphertext still correctly validating, which has a success probability depending on the size of the MAC.

As discussed in Section 13.7.2, the attack is practically infeasible if the message is protected in group mode, thanks to the countersignature also bound to the OSCORE option through the Additional Authenticated Data used in the signing process (see Section 4.4).

13.7.1. Attack Description

Let us consider:

* 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.

* A sender endpoint X which is member of both G1 and G2, and uses the same pair (private key, authentication credential) in both groups. The endpoint X has Sender ID Sid1 in G1 and Sender ID Sid2 in G2. The pairs (Sid1, Gid1) and (Sid2, Gid2) identify the same authentication credential of X in G1 and G2, respectively.

* A recipient endpoint Y which is member of both G1 and G2, and uses the same pair (private key, authentication credential) in both groups. The endpoint Y has Sender ID Sid3 in G1 and Sender ID Sid4 in G2. The pairs (Sid3, Gid1) and (Sid4, Gid2) identify the same authentication credential of Y in G1 and G2, respectively.

* A malicious endpoint Z is also member of both G1 and G2. Hence, Z is able to derive the Sender Keys used by X in G1 and G2.

When X sends a message M1 addressed to Y in G1 and protected in pairwise mode, Z can intercept M1, and attempt to forge a valid message M2 to be injected in G2, making it appear as still sent by X to Y and valid to be accepted.

More in detail, Z intercepts and stops message M1, and forges a message M2 by changing the value of the OSCORE option from M1 as follows: the 'kid context' is set to G2 (rather than G1); and the 'kid' is set to Sid2 (rather than Sid1). Then, Z injects message M2 as addressed to Y in G2.

Upon receiving M2, there is a probability equal to 2^-64 that Y successfully verifies the same unchanged MAC by using the Pairwise Recipient Key associated with X in G2.

Note that Z does not know the pairwise keys of X and Y, since it does not know and is not able to compute their shared Diffie-Hellman secret. Therefore, Z is not able to check offline if a performed forgery is actually valid, before sending the forged message to G2.

13.7.2. Attack Prevention in Group Mode

When a Group OSCORE message is protected in group mode, the countersignature 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 4.4).

That is, other than over the ciphertext, the countersignature is computed over: the ID Context (Gid) and the Partial IV, which are always present in requests; as well as the Sender ID of the message originator, which is always present in requests as well as in responses to requests protected in group mode.

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 in Section 13.7.1, since it would require the adversary to additionally forge a valid countersignature that replaces the original one in the forged message M2.

If, hypothetically, the countersignature did not cover the OSCORE option:

* The attack described in Section 13.7.1 would still be possible against response messages protected in group mode, since the same unchanged countersignature from message M1 would be also valid in message M2.

* A simplification would also be possible in performing the attack, since Z is able to derive the Sender/Recipient Keys of X and Y in G1 and G2. That is, Z can also set a convenient Partial IV in the response, until the same unchanged MAC is successfully verified by using G2 as 'request_kid_context', Sid2 as 'request_kid', and the symmetric key associated with X in G2.

  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. The probability that a single given message M1 can be used to forge a response M2 for a given request would be 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 the same group G1.

13.8. Prevention of Group Cloning Attack

Both when using the group mode and the pairwise mode, the message protection covers also the Group Manager's authentication credential. This is included in the Additional Authenticated Data used in the signing process and/or in the integrity-protected encryption process (see Section 4.4).

By doing so, an endpoint X member of a group G1 cannot perform the following attack.

1. X sets up a group G2 where it acts as Group Manager.

2. X makes G2 a "clone" of G1, i.e., G1 and G2 use the same algorithms and have the same Master Secret, Master Salt and ID Context.

3. X collects a message M sent to G1 and injects it in G2.

4. Members of G2 accept M and believe it to be originated in G2.

The attack above is effectively prevented, since message M is protected by including the authentication credential of G1's Group Manager in the Additional Authenticated Data. Therefore, members of G2 do not successfully verify and decrypt M, since they correctly use the authentication credential of X as Group Manager of G2 when attempting to.

13.9. Group OSCORE for Unicast Requests

If a request is intended to be sent over unicast as addressed to a single group member, it is NOT RECOMMENDED for the client to protect the request by using the group mode as defined in Section 8.1.

This does not include the case where the client sends a request over unicast to a proxy, to be forwarded to multiple intended recipients over multicast [I-D.ietf-core-groupcomm-bis]. In this case, the client typically protects the request with the group mode, even though it is sent to the proxy over unicast (see Section 8).

If the client uses the group mode with its own Sender Key to protect a unicast request to a group member, an on-path adversary can, right then or later on, redirect that request to one/many different group member(s) over unicast, or to the whole OSCORE group over multicast. By doing so, the adversary can induce the target group member(s) to perform actions intended for one group member only. Note that the adversary can be external, i.e., they do not need to also be a member of the OSCORE group.

This is due to the fact that the client is not able to indicate the single intended recipient in a way which is secure and possible to process for Group OSCORE on the server side. In particular, Group OSCORE does not protect network addressing information such as the IP address of the intended recipient server. It follows that the server(s) receiving the redirected request cannot assert whether that was the original intention of the client, and would thus simply assume so.

The impact of such an attack depends especially on the REST method of the request, i.e., the Inner CoAP Code of the OSCORE request message. In particular, safe methods such as GET and FETCH would trigger (several) unintended responses from the targeted server(s), while not resulting in destructive behavior. On the other hand, non safe methods such as PUT, POST, and PATCH/iPATCH would result in the target server(s) taking active actions on their resources and possible cyber-physical environment, with the risk of destructive consequences and possible implications for safety.

A client can instead use the pairwise mode as defined in Section 9.3, in order to protect a request sent to a single group member by using pairwise keying material (see Section 2.5). This prevents the attack discussed above by construction, as only the intended server is able to derive the pairwise keying material used by the client to protect the request. In a group where the AEAD Algorithm and Pairwise Key Agreement Algorithm are set in the Security Context, a client supporting the pairwise mode SHOULD use it to protect requests sent to a single group member over unicast. For an example where this is not fulfilled, see Section 9.2.1 of [I-D.ietf-core-observe-multicast-notifications].

The use of block-wise transfers [RFC7959] with group communication for CoAP is as discussed in Section 3.8 of [I-D.ietf-core-groupcomm-bis].

Additional considerations are discussed in Section 10, with respect to requests including a CoAP Echo Option [RFC9175] that have to be sent over unicast, as a challenge-response method for servers to achieve synchronization of clients' Sender Sequence Number.

13.10. End-to-end Protection

The same considerations from Section 12.1 of [RFC8613] hold for Group OSCORE.

Additionally, (D)TLS and Group OSCORE can be combined for protecting message exchanges occurring over unicast. However, it is not possible to combine (D)TLS and Group OSCORE for protecting message exchanges where messages are sent over multicast.

13.11. Master Secret

Group OSCORE derives the Security Context using the same construction used by 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 [RFC8613] 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].

13.12. Replay Protection

As in OSCORE [RFC8613], also Group OSCORE relies on Sender Sequence Numbers included in the COSE message field 'Partial IV' and used to build AEAD nonces.

Note that the Partial IV of an endpoint does not necessarily grow monotonically. For instance, upon exhaustion of the endpoint's Sender Sequence Number space, the endpoint's Partial IV space also gets exhausted. As discussed in Section 2.6.3, this results either in the endpoint being individually rekeyed and getting a new Sender ID, or in the establishment of a new Security Context in the group. Therefore, uniqueness of (key, nonce) pairs (see Section 13.3) is preserved also when a new Security Context is established.

Since one-to-many communication such as multicast usually involves unreliable transports, the simplification of the Replay Window to a size of 1 suggested in Section 7.4 of [RFC8613] is not viable with Group OSCORE, unless exchanges in the group rely only on unicast messages.

A server may not be synchronized with a client's Sender Sequence Number, or a Replay Window may be initialized as invalid (see Section 2.6.1). The server can either retrieve a new Group OSCORE Security Context, or synchronize with the clients' Sender Sequence Numbers before resuming to accept incoming messages from other group members.

13.13. Message Ordering

Assuming that the other endpoint is honest, Group OSCORE provides relative ordering of received messages. For a given Group OSCORE Security Context, the received Partial IV (when included) allows the recipient endpoint to determine the order in which requests or responses were sent by the other endpoint.

In case the Partial IV was omitted in a response, this indicates that it was the oldest response from the sender endpoint to the corresponding request (like notification responses in OSCORE, see Section 7.4.1 of [RFC8613]). A received response is not older than the corresponding request.

13.14. Message Freshness

As in OSCORE, Group OSCORE provides only the guarantee that the request is not older than the Group OSCORE Security Context used to protect it. Other aspects of freshness are discussed in Section 6.2.

The challenge-response approach described in Section 10 provides an assurance of freshness of the request without depending on the honesty of the client. 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.

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 4.01 (Unauthorized) response to the client. Thus, silent servers should adopt alternative approaches to achieve and maintain synchronization with Sender Sequence Numbers of clients.

Since requests including the Echo Option are sent over unicast, a server can be victim of the attack discussed in Section 13.9, in case such requests are protected in group mode. Instead, protecting those requests with the pairwise mode prevents the attack above. In fact, only the very server involved in the challenge-response exchange is able to derive the pairwise key used by the client to protect the request including the Echo Option.

In either case, an internal on-path adversary would not be able to mix up the Echo Option value of two different unicast requests, sent by a same client to any two different servers in the group. In fact, even if the group mode was used, this would require the adversary to forge the countersignature of both requests. As a consequence, each of the two servers remains able to selectively accept a request with the Echo Option only if it is waiting for that exact integrity-protected Echo Option value, and is thus the intended recipient.

13.15. Client Aliveness

Like in OSCORE (see Section 12.5 of [RFC8613]), a server may verify the aliveness of the client by using the CoAP Echo Option [RFC9175] as described in Section 10.

In the interest of avoiding otherwise unnecessary uses of such an approach, the server can exploit the fact that the received request cannot be older than the Security Context used to protect it. This effectively allows the server to verify the client aliveness relative to the installation of the latest group keying material.

13.16. Cryptographic Considerations

The same considerations from Section 12.6 of [RFC8613] about the maximum Sender Sequence Number hold for Group OSCORE.

As discussed in Section 2.6.2, an endpoint that experiences an exhaustion of its own Sender Sequence Number space MUST NOT protect further messages including a Partial IV, until it has derived a new Sender Context. This prevents the endpoint to reuse the same AEAD nonce 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 Security Context by means of group rekeying, or provide only that endpoint with a new Sender ID value. In 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 [RFC8613] hold for Group OSCORE, about building the AEAD nonce and the secrecy of the Security Context parameters.

The group mode uses the "encrypt-then-sign" construction, i.e., the countersignature is computed over the COSE_Encrypt0 object (see Section 4.1). This is motivated by enabling signature checkers (see Section 3.3), which do not join a group as members but are allowed to verify countersignatures of messages protected in group mode without being able to decrypt them (see Section 8.5).

If the Group Encryption Algorithm used in group mode provides integrity protection, countersignatures of COSE_Encrypt0 with short authentication tags do not provide the security properties associated with the same algorithm used in COSE_Sign (see Section 6 of [RFC9338]). To provide 128-bit security against collision attacks, the tag length MUST be at least 256-bits. A countersignature of a COSE_Encrypt0 with AES-CCM-16-64-128 provides at most 32 bits of integrity protection.

The derivation of pairwise keys defined in Section 2.5.1 is compatible with ECDSA and EdDSA asymmetric keys, but is not compatible with RSA asymmetric keys.

For the public key translation from Ed25519 (Ed448) to X25519 (X448) specified in Section 2.5.1, variable time methods can be used since the translation operates on public information. Any byte string of appropriate length is accepted as a public key for X25519 (X448) in [RFC7748]. It is therefore not necessary for security to validate the translated public key (assuming the translation was successful).

The security of using the same key pair for Diffie-Hellman and for signing (by considering the ECDH procedure in Section 2.5 as a Key Encapsulation Mechanism (KEM)) is demonstrated in [Degabriele] and [Thormarker].

Applications using ECDH (except X25519 and X448) based KEM in Section 2.5 are assumed to verify that a peer endpoint's public key is on the expected curve and that the shared secret is not the point at infinity. The KEM in [Degabriele] checks that the shared secret is different from the point at infinity, as does the procedure in Section 5.7.1.2 of [NIST-800-56A] which is referenced in Section 2.5.

By extending Theorem 2 of [Degabriele], [Thormarker] shows that the same key pair can be used with X25519 and Ed25519 (X448 and Ed448) for the KEM specified in Section 2.5. By symmetry in the KEM used in this document, both endpoints can consider themselves to have the recipient role in the KEM - as discussed in Section 7 of [Thormarker] - and rely on the mentioned proofs for the security of their key pairs.

Theorem 3 in [Degabriele] shows that the same key pair can be used for an ECDH based KEM and ECDSA. The KEM uses a different KDF than in Section 2.5, but the proof only depends on that the KDF has certain required properties, which are the typical assumptions about HKDF, e.g., that output keys are pseudorandom. In order to comply with the assumptions of Theorem 3, received public keys MUST be successfully validated, see Section 5.6.2.3.4 of [NIST-800-56A]. The validation MAY be performed by a trusted Group Manager. For

[Degabriele] to apply as it is written, public keys need to be in the expected subgroup. For this, we rely on cofactor Diffie-Hellman as per Section 5.7.1.2 of [NIST-800-56A], which is referenced in Section 2.5.1.

HashEdDSA variants of Ed25519 and Ed448 are not used by COSE (see Section 2.2 of [RFC9053]), and are not covered by the analysis in [Thormarker]. Hence, they MUST NOT be used with the public keys used to derive pairwise keys as specified in this document.

13.17. Message Segmentation

The same considerations from Section 12.7 of [RFC8613] hold for Group OSCORE.

13.18. Privacy Considerations

Group OSCORE ensures end-to-end integrity protection and encryption of the message payload and of all the 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 [RFC8613] hold for Group OSCORE, with the following addition.

* When protecting a message in group mode, the countersignature is encrypted by using a keystream derived from the group keying material (see Section 5 and Section 5.1). This ensures group privacy. That is, an attacker cannot track an endpoint over two groups by linking messages between the two groups, unless being also a member of those groups.

Furthermore, the following privacy considerations hold about the OSCORE option, which may reveal information on the communicating endpoints.

* The 'kid' parameter, which is intended to help a recipient endpoint to find the right Recipient Context, may reveal information about the Sender Endpoint. When both a request and the corresponding responses include the 'kid' parameter, this may reveal information about both a client sending a request and all the possibly replying servers sending their own individual response.

* The 'kid context' parameter, which is intended to help a recipient endpoint to find the right Security 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.

When receiving a group request, each of the recipient endpoints can reply with a response that includes its Sender ID as 'kid' parameter. All these responses will be matchable with the request through the CoAP Token. Thus, even if these responses do not include a 'kid context' parameter, it becomes possible to understand that the responder endpoints are in the same group of the requester endpoint.

Furthermore, using the approach described in Section 10 to achieve synchronization with a client's Sender Sequence Number 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.

Finally, the approach described in Section 13.6 to prevent collisions of Group Identifiers from different Group Managers may reveal information about events in the respective OSCORE groups. In particular, a Group Identifier changes when the corresponding group is rekeyed. Thus, Group Managers might use the shared list of Group Identifiers to infer the rate and patterns of group membership changes triggering a group rekeying, e.g., due to newly joined members or evicted (compromised) members. In order to alleviate this privacy concern, it should be hidden from the Group Managers which exact Group Manager has currently assigned which Group Identifiers in its OSCORE groups.

14. IANA Considerations

Note to RFC Editor: Please replace "[RFC-XXXX]" with the RFC number of this document and delete this paragraph.

This document has the following actions for IANA.

14.1. OSCORE Flag Bits Registry

IANA is asked to add the following entry to the "OSCORE Flag Bits" registry within the "Constrained RESTful Environments (CoRE) Parameters" registry group.

Bit Position	Name	Description	Reference
2	Group Flag	For using a Group OSCORE Security Context, set to 1 if the message is protected with the group mode	[RFC-XXXX]

14.2. Target Attributes Registry

IANA is asked to add the following entry to the "Target Attributes" registry within the "Constrained RESTful Environments (CoRE) Parameters" registry group.

```
Attribute Name: gosc
Brief Description: Hint: resource only accessible
                   using Group OSCORE or OSCORE
Change Controller: IETF
Reference: [RFC-XXXX]
```

15. References

15.1. Normative References

[I-D.ietf-core-groupcomm-bis]
Dijk, E., Wang, C., and M. Tiloca, "Group Communication for the Constrained Application Protocol (CoAP)", Work in Progress, Internet-Draft, draft-ietf-core-groupcomm-bis-10, 23 October 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-core-groupcomm-bis-10>.

[NIST-800-56A]
Barker, E., Chen, L., Roginsky, A., Vassilev, A., and R. Davis, "Recommendation for Pair-Wise Key-Establishment Schemes Using Discrete Logarithm Cryptography - NIST Special Publication 800-56A, Revision 3", April 2018, <https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-56Ar3.pdf>.

[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/rfc/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/rfc/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/rfc/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/rfc/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/rfc/rfc7641>.

[RFC7748] Langley, A., Hamburg, M., and S. Turner, "Elliptic Curves for Security", RFC 7748, DOI 10.17487/RFC7748, January 2016, <https://www.rfc-editor.org/rfc/rfc7748>.

[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/rfc/rfc8032>.

[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/rfc/rfc8174>.

[RFC8288] Nottingham, M., "Web Linking", RFC 8288, DOI 10.17487/RFC8288, October 2017, <https://www.rfc-editor.org/rfc/rfc8288>.

[RFC8610] Birkholz, H., Vigano, C., and C. Bormann, "Concise Data Definition Language (CDDL): A Notational Convention to Express Concise Binary Object Representation (CBOR) and JSON Data Structures", RFC 8610, DOI 10.17487/RFC8610, June 2019, <https://www.rfc-editor.org/rfc/rfc8610>.

[RFC8613] Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security for Constrained RESTful Environments (OSCORE)", RFC 8613, DOI 10.17487/RFC8613, July 2019, <https://www.rfc-editor.org/rfc/rfc8613>.

[RFC8949] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", STD 94, RFC 8949, DOI 10.17487/RFC8949, December 2020, <https://www.rfc-editor.org/rfc/rfc8949>.

[RFC9052] Schaad, J., "CBOR Object Signing and Encryption (COSE): Structures and Process", STD 96, RFC 9052, DOI 10.17487/RFC9052, August 2022, <https://www.rfc-editor.org/rfc/rfc9052>.

[RFC9053] Schaad, J., "CBOR Object Signing and Encryption (COSE): Initial Algorithms", RFC 9053, DOI 10.17487/RFC9053, August 2022, <https://www.rfc-editor.org/rfc/rfc9053>.

[RFC9175] Amsüss, C., Preuß Mattsson, J., and G. Selander, "Constrained Application Protocol (CoAP): Echo, Request-Tag, and Token Processing", RFC 9175, DOI 10.17487/RFC9175, February 2022, <https://www.rfc-editor.org/rfc/rfc9175>.

[RFC9338] Schaad, J., "CBOR Object Signing and Encryption (COSE): Countersignatures", STD 96, RFC 9338, DOI 10.17487/RFC9338, December 2022, <https://www.rfc-editor.org/rfc/rfc9338>.

15.2. Informative References

[Degabriele]
Degabriele, J. P., Lehmann, A., Paterson, K. G., Smart, N. P., and M. Strefler, "On the Joint Security of Encryption and Signature in EMV", December 2011, <https://eprint.iacr.org/2011/615>.

[I-D.amsuess-core-cachable-oscore]
Amsüss, C. and M. Tiloca, "Cacheable OSCORE", Work in Progress, Internet-Draft, draft-amsuess-core-cachable-oscore-08, 10 January 2024, <https://datatracker.ietf.org/doc/html/draft-amsuess-core-cachable-oscore-08>.

[I-D.ietf-ace-key-groupcomm]
Palombini, F. and M. Tiloca, "Key Provisioning for Group Communication using ACE", Work in Progress, Internet-Draft, draft-ietf-ace-key-groupcomm-18, 16 January 2024, <https://datatracker.ietf.org/doc/html/draft-ietf-ace-key-groupcomm-18>.

[I-D.ietf-ace-key-groupcomm-oscore]
Tiloca, M., Park, J., and F. Palombini, "Key Management for OSCORE Groups in ACE", Work in Progress, Internet-Draft, draft-ietf-ace-key-groupcomm-oscore-16, 6 March 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-ace-key-groupcomm-oscore-16>.

[I-D.ietf-core-observe-multicast-notifications]
Tiloca, M., Höglund, R., Amsüss, C., and F. Palombini, "Observe Notifications as CoAP Multicast Responses", Work in Progress, Internet-Draft, draft-ietf-core-observe-

multicast-notifications-07, 23 October 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-core-observe-multicast-notifications-07>.

[I-D.ietf-cose-cbor-encoded-cert]
Mattsson, J. P., Selander, G., Raza, S., Höglund, J., and M. Furuhed, "CBOR Encoded X.509 Certificates (C509 Certificates)", Work in Progress, Internet-Draft, draft-ietf-cose-cbor-encoded-cert-09, 4 March 2024, <https://datatracker.ietf.org/doc/html/draft-ietf-cose-cbor-encoded-cert-09>.

[I-D.ietf-iotops-security-protocol-comparison]
Mattsson, J. P., Palombini, F., and M. Vuini, "Comparison of CoAP Security Protocols", Work in Progress, Internet-Draft, draft-ietf-iotops-security-protocol-comparison-03, 23 October 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-iotops-security-protocol-comparison-03>.

[I-D.ietf-lwig-curve-representations]
Struik, R., "Alternative Elliptic Curve Representations", Work in Progress, Internet-Draft, draft-ietf-lwig-curve-representations-23, 21 January 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-lwig-curve-representations-23>.

[I-D.irtf-cfrg-det-sigs-with-noise]
Mattsson, J. P., Thormarker, E., and S. Ruohomaa, "Hedged ECDSA and EdDSA Signatures", Work in Progress, Internet-Draft, draft-irtf-cfrg-det-sigs-with-noise-02, 1 March 2024, <https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-det-sigs-with-noise-02>.

[I-D.tiloca-core-groupcomm-proxy]
Tiloca, M. and E. Dijk, "Proxy Operations for CoAP Group Communication", Work in Progress, Internet-Draft, draft-tiloca-core-groupcomm-proxy-09, 31 August 2023, <https://datatracker.ietf.org/doc/html/draft-tiloca-core-groupcomm-proxy-09>.

[I-D.tiloca-core-oscore-capable-proxies]
Tiloca, M. and R. Höglund, "OSCORE-capable Proxies", Work in Progress, Internet-Draft, draft-tiloca-core-oscore-capable-proxies-07, 10 July 2023, <https://datatracker.ietf.org/doc/html/draft-tiloca-core-oscore-capable-proxies-07>.

[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/rfc/rfc4944>.

[RFC4949] Shirey, R., "Internet Security Glossary, Version 2", FYI 36, RFC 4949, DOI 10.17487/RFC4949, August 2007, <https://www.rfc-editor.org/rfc/rfc4949>.

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <https://www.rfc-editor.org/rfc/rfc5280>.

[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/rfc/rfc5869>.

[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/rfc/rfc6282>.

[RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <https://www.rfc-editor.org/rfc/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/rfc/rfc7228>.

[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/rfc/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/rfc/rfc8075>.

[RFC8392] Jones, M., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "CBOR Web Token (CWT)", RFC 8392, DOI 10.17487/RFC8392, May 2018, <https://www.rfc-editor.org/rfc/rfc8392>.

[RFC9147] Rescorla, E., Tschofenig, H., and N. Modadugu, "The Datagram Transport Layer Security (DTLS) Protocol Version 1.3", RFC 9147, DOI 10.17487/RFC9147, April 2022, <https://www.rfc-editor.org/rfc/rfc9147>.

[RFC9200] Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "Authentication and Authorization for Constrained Environments Using the OAuth 2.0 Framework (ACE-OAuth)", RFC 9200, DOI 10.17487/RFC9200, August 2022, <https://www.rfc-editor.org/rfc/rfc9200>.

[Thormarker] Thormarker, E., "On using the same key pair for Ed25519 and an X25519 based KEM", April 2021, <https://eprint.iacr.org/2021/509>.

Appendix A. Assumptions and Security Objectives

This section presents a set of assumptions and security objectives for the protocol described in this document. The rest of this section refers to three types of groups:

* Application group, i.e., a set of CoAP endpoints that share a common pool of resources.

* Security group, as defined in Section 1.1 of this document. There can be a one-to-one or a one-to-many relation between security groups and application groups, and vice versa.

* CoAP group, i.e., a set of CoAP endpoints where each endpoint is configured to receive one-to-many CoAP requests, e.g., sent to the group's associated IP multicast address and UDP port as defined in [I-D.ietf-core-groupcomm-bis]. An endpoint may be a member of multiple CoAP groups. There can be a one-to-one or a one-to-many relation between application groups and CoAP groups. Note that a device sending a CoAP request to a CoAP group is not necessarily itself a member of that group: it is a member only if it also has a CoAP server endpoint listening to requests for this CoAP group, sent to the associated IP multicast address and port. In order to provide secure group communication, all members of a CoAP group as well as all further endpoints configured only as clients sending CoAP (multicast) requests to the CoAP group have to be member of a security group. There can be a one-to-one or a one-to-many relation between security groups and CoAP groups, and vice versa.

A.1. Assumptions

The following points are assumed to be already addressed and are out of the scope of this document.

* 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 CoAP 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 clients transmit data to the CoAP group. According to [I-D.ietf-core-groupcomm-bis], any possible proxy entity is supposed to know about the clients. Also, every client expects and is able to handle multiple response messages associated with a same request sent to the CoAP group.

* Group size: security solutions for group communication should be able to adequately support different and possibly large security groups. The group size is the current number of members in a security 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 hundreds to thousands of devices, with large groups easier to manage if including several silent servers. Security groups larger than that should be divided into smaller independent groups. One should not assume that the set of members of a security group remains fixed. That is, the group membership is subject to changes, possibly on a frequent basis.

* Communication with the Group Manager: an endpoint must use a secure dedicated channel when communicating with the Group Manager, also when not registered as a member of the security group.

* Provisioning and management of Security Contexts: a Security Context must be established among the members of the security group. A secure mechanism must be used to generate, revoke and (re-)distribute keying material, communication policies and

security parameters in the security group. The actual provisioning and management of the Security Context is out of the scope of this document.

* Multicast data security cipher suite: all members of a security group must use the same cipher suite to provide authenticity, integrity and confidentiality of messages in the group. The cipher suite is specified as part of the Security Context.

* Ensuring backward security: a new device joining the security group should not have access to any old Security Contexts used before its joining. This ensures that a new member of the security group is not able to decrypt confidential data sent before it has joined the security 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 in the security group upon a new member's joining has to be defined as part of the group key management scheme.

* Ensuring forward security: entities that leave the security group should not have access to any future Security Contexts or message exchanged within the security group after their leaving. This ensures that a former member of the security group is not able to decrypt confidential data sent within the security group anymore. Also, it ensures that a former member is not able to send protected messages to the security group anymore. The actual mechanism to update the Security Context and renew the group keying material in the security group upon a member's leaving has to be defined as part of the group key management scheme.

A.2. Security Objectives

The protocol described in this document aims at fulfilling the following security objectives:

* Data replay protection: request messages or response messages replayed within the security group must be detected.

* Data confidentiality: messages sent within the security group shall be encrypted.

* Group-level data confidentiality: the group mode provides 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 security group, but not by an external adversary or other external entities.

* Pairwise data confidentiality: the pairwise mode especially provides pairwise data confidentiality, since messages are encrypted using pairwise keying material shared between any two group members, hence they can be decrypted only by the intended single recipient.

* Source message authentication: messages sent within the security group shall be authenticated. That is, it is essential to ensure that a message is originated by a member of the security group in the first place, and in particular by a specific, identifiable member of the security group.

* Message integrity: messages sent within the security group shall be integrity protected. That is, it is essential to ensure that a message has not been tampered with, either by a group member, or by an external adversary or other external entities which are not members of the security group.

* Message ordering: it must be possible to determine the ordering of messages coming from a single sender. Like in OSCORE [RFC8613], a recipient endpoint can determine the relative order of requests or responses from another sender endpoint by means of their Partial IV. It is not required to determine ordering of messages from different senders.

Appendix B. List of Use Cases

Group Communication for CoAP [I-D.ietf-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 [I-D.ietf-core-groupcomm-bis] to understand the non-security related details. This section discusses a number of use cases that benefit from secure group communication, and refers to the three types of groups from Appendix A. Specific security requirements for these use cases are discussed in Appendix A.

* Lighting control: consider a building equipped with IP-connected lighting devices, switches, and border routers. The lighting devices acting as servers are organized into application groups and CoAP groups, according to their physical location in the building. For instance, lighting devices in a room or corridor can be configured as members of a single application group and corresponding CoAP group. Those lighting devices together with the switches acting as clients in the same room or corridor can be configured as members of the corresponding security group. Switches are then used to control the lighting devices by sending on/off/dimming commands to all lighting devices in the CoAP 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 with a role in the lighting application 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 set of connected lights, ensuring that the light preset (e.g., dimming level or color) of a large set 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 set of lighting devices. In more advanced lighting control use cases, an 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 set 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.

* Integrated building control: enabling Building Automation and Control Systems (BACSs) to control multiple heating, ventilation and air-conditioning units to predefined presets. Controlled units can be organized into application groups and CoAP 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 application group and corresponding CoAP 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.

* Software and firmware updates: software and firmware updates often comprise quite a large amount of data. This can overload a Low-power and Lossy Network (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 set 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.

* Parameter and configuration update: by means of multicast communication, it is possible to update the settings of a set of similar devices, both simultaneously and efficiently. Possible parameters are related, for instance, to network load management or network access control. 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.

* Commissioning of Low-power and Lossy Network (LLN) 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 application group and corresponding CoAP 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.

* 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-tolerant multicast algorithm, such as Multicast Protocol for Low-Power and Lossy Networks (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 with 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 each time new keying material, together with a new Gid, is distributed to the group in order to establish a new Security Context (see Section 3.2).

As an example, a 3-byte Gid 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 Common Context 61532 times in the group, its Gid will assume value '0xb1f05c'.

Using an immutable Group Prefix for a group with a Group Manager that cannot reassign Gid values (see Section 3.2) limits the total number of rekeying instances. With a Group Manager that does reassign Gid values, it limits the maximum active number of rekeying instances that a CoAP observation [RFC7641] can persist through. In either case, the group epoch size needs to be chosen depending on the expected rate of rekeying instances.

As discussed in Section 13.6, 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 Security Contexts to process an incoming message, until the right one is found and the message is correctly verified. Therefore, it is favorable 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. Document Updates

RFC EDITOR: PLEASE REMOVE THIS SECTION.

D.1. Version -20 to -21

* Updated author list.

* Terminology: improved definition of "group request".

* Editorial: removed quotation marks when using the CBOR simple values true, false, and null.

* Editorial: expanded name of the "CoRE Parameters" registry group.

D.2. Version -19 to -20

* Change Controller for the target attribute "gosc" set to "IETF".

D.3. Version -18 to -19

* Unified presentation of handling of multiple responses.

* Added Rikard Hoeglund as Contributor.

D.4. Version -17 to -18

* Changed document title.

* Possible use with CoAP-mappable HTTP.

* Added Common Context parameter "Authentication Credential Format".

* Renamed "Group Encryption Key" to "Signature Encryption Key". Consistent fixes in its derivation.

* Renamed "Signature Encryption Algorithm" to "Group Encryption Algorithm".

* Ensured a single Common IV, also when the two encryption algorithms have different AEAD nonce sizes.

* Guidelines on the Pairwise Key Agreement Algorithm and derivation of the Diffie-Hellman secret.

* The possible use of a mode follows from the set parameters.

* The Group Manager is always present; 'gm_cred' in the external_aad cannot be null anymore.

* The authentication credential of the Group Manager can have a different format than that of the group members'.

* Set-up of new endpoints moved to document body.

* The encrypted countersignature is a result of the header compression, not of COSE.

* Revised examples of compressed and non-compressed COSE object.
* Removed excessive requirements on group rekeying scheduling.
* More considerations on the strictness of group key management.
* Clearer alternatives on retaining an old Security Context.
* Revised used of terminology on freshness.
* Clarifications, fixes and editorial improvements.

D.5. Version -16 to -17

* Definition and registration of the target attribute "gosc".
* Reference update and editorial fixes.

D.6. Version -15 to -16

* Clients "SHOULD" use the group mode for one-to-many requests.
* Handling of multiple non-notification responses.
* Revised presentation of security properties.
* Improved listing of operations defined for the group mode that are inherited by the pairwise mode.
* Editorial improvements.

D.7. Version -14 to -15

* Updated references and editorial fixes.

D.8. Version -13 to -14

* Replaced "node" with "endpoint" where appropriate.
* Replaced "owning" with "storing" (of keying material).
* Distinction between "authentication credential" and "public key".
* Considerations on storing whole authentication credentials.
* Considerations on Denial of Service.

* Recycling of Group IDs by tracking the "Birth Gid" of each group member is now optional to support and use for the Group Manager.

* Fine-grained suppression of error responses.

* Changed section title "Mandatory-to-Implement Compliance Requirements" to "Implementation Compliance".

* "Challenge-Response Synchronization" moved to the document body.

* RFC 7641 and draft-ietf-core-echo-request-tag as normative references.

* Clarifications and editorial improvements.

D.9. Version -12 to -13

* Fixes in the derivation of the Group Encryption Key.

* Added Mandatory-to-Implement compliance requirements.

* Changed UCCS to CCS.

D.10. Version -11 to -12

* No mode of operation is mandatory to support.

* Revised parameters of the Security Context, COSE object and external_aad.

* Revised management of keying material for the Group Manager.

* Informing of former members when rekeying the group.

* Admit encryption-only algorithms in group mode.

* Encrypted countersignature through a keystream.

* Added public key of the Group Manager as key material and protected data.

* Clarifications about message processing, especially notifications.

* Guidance for message processing of external signature checkers.

* Updated derivation of pairwise keys, with more security considerations.

* Termination of ongoing observations as client, upon leaving or before re-joining the group.

* Recycling Group IDs by tracking the "Birth Gid" of each group member.

* Expanded security and privacy considerations about the group mode.

* Removed appendices on skipping signature verification and on COSE capabilities.

* Fixes and editorial improvements.

D.11. Version -10 to -11

* Loss of Recipient Contexts due to their overflow.

* Added diagram on keying material components and their relation.

* Distinction between anti-replay and freshness.

* Preservation of Sender IDs over rekeying.

* Clearer cause-effect about reset of SSN.

* The GM provides public keys of group members with associated Sender IDs.

* Removed 'par_countersign_key' from the external_aad.

* One single format for the external_aad, both for encryption and signing.

* Presence of 'kid' in responses to requests protected in pairwise mode.

* Inclusion of 'kid_context' in notifications following a group rekeying.

* Pairwise mode presented with OSCORE as baseline.

* Revised examples with signature values.

* Decoupled growth of clients' Sender Sequence Numbers and loss of synchronization for server.

* Sender IDs not recycled in the group under the same Gid.

* Processing and description of the Group Flag bit in the OSCORE option.

* Usage of the pairwise mode for multicast requests.

* Clarifications on synchronization using the Echo option.

* General format of context parameters and external_aad elements, supporting future registered COSE algorithms (new Appendix).

* Fixes and editorial improvements.

D.12. Version -09 to -10

* Removed 'Counter Signature Key Parameters' from the Common Context.

* New parameters in the Common Context covering the DH secret derivation.

* New countersignature header parameter from draft-ietf-cose-countersign.

* Stronger policies non non-recycling of Sender IDs and Gid.

* The Sender Sequence Number is reset when establishing a new Security Context.

* Added 'request_kid_context' in the aad_array.

* The server can respond with 5.03 if the client's public key is not available.

* The observer client stores an invariant identifier of the group.

* Relaxed storing of original 'kid' for observer clients.

* Both client and server store the 'kid_context' of the original observation request.

* The server uses a fresh PIV if protecting the response with a Security Context different from the one used to protect the request.

* Clarifications on MTI algorithms and curves.

* Removed optimized requests.

* Overall clarifications and editorial revision.

D.13. Version -08 to -09

* Pairwise keys are discarded after group rekeying.

* Signature mode renamed to group mode.

* The parameters for countersignatures use the updated COSE registries. Newly defined IANA registries have been removed.

* Pairwise Flag bit renamed as Group Flag bit, set to 1 in group mode and set to 0 in pairwise mode.

* Dedicated section on updating the Security Context.

* By default, sender sequence numbers and replay windows are not reset upon group rekeying.

* An endpoint implementing only a silent server does not support the pairwise mode.

* Separate section on general message reception.

* Pairwise mode moved to the document body.

* Considerations on using the pairwise mode in non-multicast settings.

* Optimized requests are moved as an appendix.

* Normative support for the signature and pairwise mode.

* Revised methods for synchronization with clients' sender sequence number.

* Appendix with example values of parameters for countersignatures.

* Clarifications and editorial improvements.

D.14. Version -07 to -08

* Clarified relation between pairwise mode and group communication (Section 1).

* Improved definition of "silent server" (Section 1.1).

* Clarified when a Recipient Context is needed (Section 2).

* Signature checkers as entities supported by the Group Manager (Section 2.3).

* Clarified that the Group Manager is under exclusive control of Gid and Sender ID values in a group, with Sender ID values under each Gid value (Section 2.3).

* Mitigation policies in case of recycled 'kid' values (Section 2.4).

* More generic exhaustion (not necessarily wrap-around) of sender sequence numbers (Sections 2.5 and 10.11).

* Pairwise key considerations, as to group rekeying and Sender Sequence Numbers (Section 3).

* Added reference to static-static Diffie-Hellman shared secret (Section 3).

* Note for implementation about the external_aad for signing (Sectino 4.3.2).

* Retransmission by the application for group requests over multicast as Non-confirmable (Section 7).

* A server MUST use its own Partial IV in a response, if protecting it with a different context than the one used for the request (Section 7.3).

* Security considerations: encryption of pairwise mode as alternative to group-level security (Section 10.1).

* Security considerations: added approach to reduce the chance of global collisions of Gid values from different Group Managers (Section 10.5).

* Security considerations: added implications for block-wise transfers when using the signature mode for requests over unicast (Section 10.7).

* Security considerations: (multiple) supported signature algorithms (Section 10.13).

* Security considerations: added privacy considerations on the approach for reducing global collisions of Gid values (Section 10.15).

* Updates to the methods for synchronizing with clients' sequence number (Appendix E).

* Simplified text on discovery services supporting the pairwise mode (Appendix G.1).

* Editorial improvements.

D.15. Version -06 to -07

* Updated abstract and introduction.

* Clarifications of what pertains a group rekeying.

* Derivation of pairwise keying material.

* Content re-organization for COSE Object and OSCORE header compression.

* Defined the Pairwise Flag bit for the OSCORE option.

* Supporting CoAP Observe for group requests and responses.

* Considerations on message protection across switching to new keying material.

* New optimized mode based on pairwise keying material.

* More considerations on replay protection and Security Contexts upon key renewal.

* Security considerations on Group OSCORE for unicast requests, also as affecting the usage of the Echo option.

* Clarification on different types of groups considered (application/security/CoAP).

* New pairwise mode, using pairwise keying material for both requests and responses.

D.16. Version -05 to -06

* Group IDs mandated to be unique under the same Group Manager.

* Clarifications on parameter update upon group rekeying.

* Updated external_aad structures.

* Dynamic derivation of Recipient Contexts made optional and application specific.

* Optional 4.00 response for failed signature verification on the server.

* Removed client handling of duplicated responses to multicast requests.

* Additional considerations on public key retrieval and group rekeying.

* Added Group Manager responsibility on validating public keys.

* Updates IANA registries.

* Reference to RFC 8613.

* Editorial improvements.

D.17. Version -04 to -05

* Added references to draft-dijk-core-groupcomm-bis.

* New parameter Counter Signature Key Parameters (Section 2).

* Clarification about Recipient Contexts (Section 2).

* Two different external_aad for encrypting and signing (Section 3.1).

* Updated response verification to handle Observe notifications (Section 6.4).

* Extended Security Considerations (Section 8).

* New "Counter Signature Key Parameters" IANA Registry (Section 9.2).

D.18. Version -03 to -04

* Added the new "Counter Signature Parameters" in the Common Context (see Section 2).

* Added recommendation on using "deterministic ECDSA" if ECDSA is used as countersignature algorithm (see Section 2).

* Clarified possible asynchronous retrieval of keying material from the Group Manager, in order to process incoming messages (see Section 2).

* Structured Section 3 into subsections.

* Added the new 'par_countersign' to the aad_array of the external_aad (see Section 3.1).

* Clarified non reliability of 'kid' as identity identifier for a group member (see Section 2.1).

* Described possible provisioning of new Sender ID in case of Partial IV wrap-around (see Section 2.2).

* The former signature bit in the Flag Byte of the OSCORE option value is reverted to reserved (see Section 4.1).

* 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).

* Relaxed statements on sending error messages (see Section 6).

* Added explicit step on computing the countersignature for outgoing messages (see Sections 6.1 and 6.3).

* Handling of just created Recipient Contexts in case of unsuccessful message verification (see Sections 6.2 and 6.4).

* Handling of replied/repeated responses on the client (see Section 6.4).

* New IANA Registry "Counter Signature Parameters" (see Section 9.1).

D.19. Version -02 to -03

* Revised structure and phrasing for improved readability and better alignment with draft-ietf-core-object-security.

* Added discussion on wrap-Around of Partial IVs (see Section 2.2).

* Separate sections for the COSE Object (Section 3) and the OSCORE Header Compression (Section 4).

* The countersignature is now appended to the encrypted payload of the OSCORE message, rather than included in the OSCORE Option (see Section 4).

* Extended scope of Section 5, now titled " Message Binding, Sequence Numbers, Freshness and Replay Protection".

* Clarifications about Non-confirmable messages in Section 5.1 "Synchronization of Sender Sequence Numbers".

* Clarifications about error handling in Section 6 "Message Processing".

* Compacted list of responsibilities of the Group Manager in Section 7.

* Revised and extended security considerations in Section 8.

* Added IANA considerations for the OSCORE Flag Bits Registry in Section 9.

* Revised Appendix D, now giving a short high-level description of a new endpoint set-up.

D.20. Version -01 to -02

* 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".

* Section 2 has been updated to have the Group Identifier stored in the 'ID Context' parameter defined in draft-ietf-core-object-security.

* Section 3 has been updated with the new format of the Additional Authenticated Data.

* Major rewriting of Section 4 to better highlight the differences with the message processing in draft-ietf-core-object-security.

* Added Sections 7.2 and 7.3 discussing security considerations about uniqueness of (key, nonce) and collision of group identifiers, respectively.

* Minor updates to Appendix A.1 about assumptions on multicast communication topology and group size.

* Updated Appendix C on format of group identifiers, with practical implications of possible collisions of group identifiers.

* Updated Appendix D.2, adding a pointer to draft-palombini-ace-key-groupcomm about retrieval of nodes' public keys through the Group Manager.

* 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.

D.21. Version -00 to -01

* Section 1.1 has been updated with the definition of group as "security group".

* Section 2 has been updated with:

  - Clarifications on establishment/derivation of Security Contexts.

  - A table summarizing the the additional context elements compared to OSCORE.

* 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.

* Added Section 6, listing the responsibilities of the Group Manager.

* Added Appendix A (former section), including assumptions and security objectives.

* Appendix B has been updated with more details on the use cases.

* Added Appendix C, providing an example of Group Identifier format.

* Appendix D has been updated to be aligned with draft-palombini-ace-key-groupcomm.

Acknowledgments

Jiye Park contributed as a co-author of initial versions of this document.

The authors sincerely thank Christian Amsüss, Stefan Beck, Rolf Blom, Carsten Bormann, Esko Dijk, Martin Gunnarsson, Klaus Hartke, Richard Kelsey, Dave Robin, Jim Schaad, Ludwig Seitz, Peter van der Stok, and Erik Thormarker for their feedback and comments.

The work on this document has been partly supported by VINNOVA and the Celtic-Next project CRITISEC; the H2020 projects SIFIS-Home (Grant agreement 952652) and ARCADIAN-IoT (Grant agreement 101020259); the SSF project SEC4Factory under the grant RIT17-0032; and the EIT-Digital High Impact Initiative ACTIVE.

Authors' Addresses

Marco Tiloca
RISE AB
Isafjordsgatan 22
SE-16440 Stockholm Kista
Sweden
Email: marco.tiloca@ri.se

Göran Selander
Ericsson AB
Torshamnsgatan 23
SE-16440 Stockholm Kista
Sweden
Email: goran.selander@ericsson.com

Francesca Palombini
Ericsson AB
Torshamnsgatan 23
SE-16440 Stockholm Kista
Sweden
Email: francesca.palombini@ericsson.com

John Preuß Mattsson
Ericsson AB
Torshamnsgatan 23
SE-16440 Stockholm Kista
Sweden
Email: john.mattsson@ericsson.com

Rikard Höglund
RISE AB
Isafjordsgatan 22
SE-16440 Stockholm Kista
Sweden
Email: rikard.hoglund@ri.se

Network Working Group
Internet-Draft
Intended status: Informational
Expires: 8 August 2022

J. Preuß Mattsson
J. Fornehed
G. Selander
F. Palombini
Ericsson
C. Amsüss
Energy Harvesting Solutions

4 February 2022

Attacks on the Constrained Application Protocol (CoAP)
draft-mattsson-core-coap-attacks-03

Abstract

Being able to securely read information from sensors, to securely control actuators, and to not enable distributed denial-of-service attacks are essential in a world of connected and networking things interacting with the physical world. This document summarizes a number of known attacks on CoAP and show that just using CoAP with a security protocol like DTLS, TLS, or OSCORE is not enough for secure operation. Several of the discussed attacks can be mitigated with the solutions in draft-ietf-core-echo-request-tag.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction 2
2. Attacks on CoAP 4
  2.1. The Block Attack 4
  2.2. The Request Delay Attack 6
  2.3. The Response Delay and Mismatch Attack 9
  2.4. The Request Fragment Rearrangement Attack 12
    2.4.1. Completing an Operation with an Earlier Final Block 13
    2.4.2. Injecting a Withheld First Block 14
    2.4.3. Attack difficulty 15
  2.5. The Relay Attack 16
3. Security Considerations 17
4. IANA Considerations 17
5. Informative References 17
Acknowledgements 19
Authors' Addresses 19

1. Introduction

Being able to securely read information from sensors and to securely control actuators are essential in a world of connected and networking things interacting with the physical world. One protocol used to interact with sensors and actuators is the Constrained Application Protocol (CoAP) [RFC7252]. Any Internet-of-Things (IoT) deployment valuing security and privacy would use a security protocol such as DTLS [I-D.ietf-tls-dtls13], TLS [RFC8446], or OSCORE [RFC8613] to protect CoAP, where the choice of security protocol depends on the transport protocol and the presence of intermediaries. The use of CoAP over UDP and DTLS is specified in [RFC7252] and the use of CoAP over TCP and TLS is specified in [RFC8323]. OSCORE protects CoAP end-to-end with the use of COSE [RFC8152] and the CoAP Object-Security option [RFC8613], and can therefore be used over any transport.

The Constrained Application Protocol (CoAP) [RFC7252] was designed with the assumption that security could be provided on a separate layer, in particular by using DTLS [RFC6347]. The four properties traditionally provided by security protocols are:

* Data confidentiality

* Data origin authentication

* Data integrity checking

* Replay protection

In this document we show that protecting CoAP with a security protocol on another layer is not nearly enough to securely control actuators (and in many cases sensors) and that secure operation often demands far more than the four properties traditionally provided by security protocols. We describe several serious attacks any on-path attacker (i.e., not only "trusted intermediaries") can do and discusses tougher requirements and mechanisms to mitigate the attacks. In general, secure operation of actuators also requires the three properties:

* Data-to-data binding

* Data-to-space binding

* Data-to-time binding

"Data-to-data binding" is e.g., binding of responses to a request or binding of data fragments to each other. "Data-to-space binding" is the binding of data to an absolute or relative point in space (i.e., a location) and may in the relative case be referred to as proximity. "Data-to-time binding" is the binding of data to an absolute or relative point in time and may in the relative case be referred to as freshness. The two last properties may be bundled together as "Data-to-spacetime binding".

Freshness is a measure of when a message was sent on a timescale of the recipient. A client or server that receives a message can either verify that the message is fresh or determine that it cannot be verified that the message is fresh. What is considered fresh is application dependent. Freshness is completely different from replay protection, but most replay protection mechanism use a sequence number. Assuming the client is well-behaving, such a sequence number that can be used by the server as a relative measure of when a message was sent on a timescale of the sender. Replay protection is mandatory in TLS and OSCORE and optional in DTLS. DTLS and TLS use

sequence numbers for both requests and responses. In TLS the sequence numbers are implicit and not sent in the record. OSCORE use sequence numbers for requests and some responses. Most OSCORE responses are bound to the request and therefore, enable the client to determine if the response is fresh or not.

The request delay attack (valid for DTLS, TLS, and OSCORE and described in Section 2.2) lets an attacker control an actuator at a much later time than the client anticipated. The response delay and mismatch attack (valid for DTLS and TLS and described in Section 2.3) lets an attacker respond to a client with a response meant for an older request. The request fragment rearrangement attack (valid for DTLS, TLS, and OSCORE and described in Section 2.4) lets an attacker cause unauthorized operations to be performed on the server, and responses to unauthorized operations to be mistaken for responses to authorized operations.

The goal with this document is motivating generic and protocol-specific recommendations on the usage of CoAP. Mechanisms mitigating some of the attacks discussed in this document can be found in [I-D.ietf-core-echo-request-tag]. This document is a companion document to [I-D.ietf-core-echo-request-tag] giving more information on the attacks motivating the mechanisms.

2. Attacks on CoAP

Internet-of-Things (IoT) deployments valuing security and privacy, need to use a security protocol such as DTLS, TLS, or OSCORE to protect CoAP. This is especially true for deployments of actuators where attacks often (but not always) have serious consequences. The attacks described in this section are made under the assumption that CoAP is already protected with a security protocol such as DTLS, TLS, or OSCORE, as an attacker otherwise can easily forge false requests and responses.

2.1. The Block Attack

An on-path attacker can block the delivery of any number of requests or responses. The attack can also be performed by an attacker jamming the lower layer radio protocol. This is true even if a security protocol like DTLS, TLS, or OSCORE is used. Encryption makes selective blocking of messages harder, but not impossible or even infeasible. With DTLS and TLS, proxies can read the complete CoAP message, and with OSCORE, the CoAP header and several CoAP options are not encrypted. In all three security protocols, the IP-addresses, ports, and CoAP message lengths are available to all on-path attackers, which may be enough to determine the server, resource, and command. The block attack is illustrated in Figures 1

and 2.

```
Client   Foe   Server
   |      |      |
   +----->X      |      Code: 0.03 (PUT)
   | PUT  |      |     Token: 0x47
   |      |      |  Uri-Path: lock
   |      |      |   Payload: 1 (Lock)
   |      |      |
```

Figure 1: Blocking a request

Where 'X' means the attacker is blocking delivery of the message.

```
Client   Foe   Server
   |      |      |
   +------------>|      Code: 0.03 (PUT)
   |      | PUT  |     Token: 0x47
   |      |      |  Uri-Path: lock
   |      |      |   Payload: 1 (Lock)
   |      |      |
   |      X<-----+      Code: 2.04 (Changed)
   |      | 2.04 |     Token: 0x47
   |      |      |
```

Figure 2: Blocking a response

While blocking requests to, or responses from, a sensor is just a denial-of-service attack, blocking a request to, or a response from, an actuator results in the client losing information about the server's status. If the actuator e.g., is a lock (door, car, etc.), the attack results in the client not knowing (except by using out-of-band information) whether the lock is unlocked or locked, just like the observer in the famous Schrödingers cat thought experiment. Due to the nature of the attack, the client cannot distinguish the attack from connectivity problems, offline servers, or unexpected behavior from middle boxes such as NATs and firewalls.

Remedy: Any IoT deployment of actuators where synchronized state is important need to use confirmable messages and the client need to take appropriate actions when a response is not received and it therefore loses information about the server's status.

2.2. The Request Delay Attack

An on-path attacker may not only block packets, but can also delay the delivery of any packet (request or response) by a chosen amount of time. If CoAP is used over a reliable and ordered transport such as TCP with TLS or OSCORE (with TLS-like sequence number handling), no messages can be delivered before the delayed message. If CoAP is used over an unreliable and unordered transport such as UDP with DTLS or OSCORE, other messages can be delivered before the delayed message as long as the delayed packet is delivered inside the replay window. When CoAP is used over UDP, both DTLS and OSCORE allow out-of-order delivery and uses sequence numbers together with a replay window to protect against replay attacks against requests. The replay window has a default length of 64 in DTLS and 32 in OSCORE. The attacker can influence the replay window state by blocking and delaying packets. By first delaying a request, and then later, after delivery, blocking the response to the request, the client is not made aware of the delayed delivery except by the missing response. In general, the server has no way of knowing that the request was delayed and will therefore happily process the request. Note that delays can also happen for other reasons than a malicious attacker.

If some wireless low-level protocol is used, the attack can also be performed by the attacker simultaneously recording what the client transmits while at the same time jamming the server. The request delay attack is illustrated in Figure 3.

```
Client   Foe   Server
   |      |      |
   +----->@      |      Code: 0.03 (PUT)
   | PUT  |      |     Token: 0x9c
   |      |      |  Uri-Path: lock
   |      |      |   Payload: 0 (Unlock)
   |      |      |
     ....   ....
   |      |      |
   |      @----->|      Code: 0.03 (PUT)
   |      | PUT  |     Token: 0x9c
   |      |      |  Uri-Path: lock
   |      |      |   Payload: 0 (Unlock)
   |      |      |
   |      X<-----+      Code: 2.04 (Changed)
   |      | 2.04 |     Token: 0x9c
   |      |      |
```

Figure 3: Delaying a request

Where '@' means the attacker is storing and later forwarding the message (@ may alternatively be seen as a wormhole connecting two points in time).

While an attacker delaying a request to a sensor is often not a security problem, an attacker delaying a request to an actuator performing an action is often a serious problem. A request to an actuator (for example a request to unlock a lock) is often only meant to be valid for a short time frame, and if the request does not reach the actuator during this short timeframe, the request should not be fulfilled. In the unlock example, if the client does not get any response and does not physically see the lock opening, the user is likely to walk away, calling the locksmith (or the IT-support).

If a non-zero replay window is used (the default when CoAP is used over UDP), the attacker can let the client interact with the actuator before delivering the delayed request to the server (illustrated in Figure 4). In the lock example, the attacker may store the first "unlock" request for later use. The client will likely resend the request with the same token. If DTLS is used, the resent packet will have a different sequence number and the attacker can forward it. If OSCORE is used, resent packets will have the same sequence number and the attacker must block them all until the client sends a new message with a new sequence number (not shown in Figure 4). After a while when the client has locked the door again, the attacker can deliver the delayed "unlock" message to the door, a very serious attack.

```
               Client   Foe   Server
                  |      |      |
                  +----->@      |      Code: 0.03 (PUT)
                  | PUT  |      |     Token: 0x9c
                  |      |      |  Uri-Path: lock
                  |      |      |   Payload: 0 (Unlock)
                  |      |      |
                  +------------>|      Code: 0.03 (PUT)
                  | PUT  |      |     Token: 0x9c
                  |      |      |  Uri-Path: lock
                  |      |      |   Payload: 0 (Unlock)
                  |      |      |
                  <-------------+      Code: 2.04 (Changed)
                  |      | 2.04 |     Token: 0x9c
                  |      |      |
                    ....   ....
                  |      |      |
                  +------------>|      Code: 0.03 (PUT)
                  | PUT  |      |     Token: 0x7a
                  |      |      |  Uri-Path: lock
                  |      |      |   Payload: 1 (Lock)
                  |      |      |
                  <-------------+      Code: 2.04 (Changed)
                  |      | 2.04 |     Token: 0x7a
                  |      |      |
                  |      @----->|      Code: 0.03 (PUT)
                  |      | PUT  |     Token: 0x9c
                  |      |      |  Uri-Path: lock
                  |      |      |   Payload: 0 (Unlock)
                  |      |      |
                  |      X<-----+      Code: 2.04 (Changed)
                  |      | 2.04 |     Token: 0x9c
                  |      |      |
```

Figure 4: Delaying request with reordering

While the second attack (Figure 4) can be mitigated by using a replay window of length zero, the first attack (Figure 3) cannot. A solution must enable the server to verify that the request was received within a certain time frame after it was sent or enable the server to securely determine an absolute point in time when the request is to be executed. This can be accomplished with either a challenge-response pattern, by exchanging timestamps between client and server, or by only allowing requests a short period after client authentication.

Requiring a fresh client authentication (such as a new TLS/DTLS handshake or an EDHOC key exchange [I-D.ietf-lake-edhoc]) mitigates the problem, but requires larger messages and more processing than a dedicated solution. Security solutions based on exchanging timestamps require exactly synchronized time between client and server, and this may be hard to control with complications such as time zones and daylight saving. Wall clock time is not monotonic, may reveal that the endpoints will accept expired certificates, or reveal the endpoint's location. Use of non-monotonic clocks is problematic as the server will accept requests if the clock is moved backward and reject requests if the clock is moved forward. Even if the clocks are synchronized at one point in time, they may easily get out-of-sync and an attacker may even be able to affect the client or the server time in various ways such as setting up a fake NTP server, broadcasting false time signals to radio-controlled clocks, or exposing one of them to a strong gravity field. As soon as client falsely believes it is time synchronized with the server, delay attacks are possible. A challenge response mechanism where the server does not need to synchronize its time with the client is easier to analyze but require more roundtrips. The challenges, responses, and timestamps may be sent in a CoAP option or in the CoAP payload.

Remedy: Any IoT deployment of actuators where freshness is important should use the mechanisms specified in [I-D.ietf-core-echo-request-tag] unless another application specific challenge-response or timestamp mechanism is used.

2.3. The Response Delay and Mismatch Attack

The following attack can be performed if CoAP is protected by a security protocol where the response is not bound to the request in any way except by the CoAP token. This would include most general security protocols, such as DTLS, TLS, and IPsec, but not OSCORE. CoAP [RFC7252] uses a client generated token that the server echoes to match responses to request, but does not give any guidelines for the use of token with DTLS and TLS, except that the tokens currently "in use" SHOULD (not SHALL) be unique. 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 attacker performs the attack by delaying delivery of a response until the client sends a request with the same token, the response will be accepted by the client as a valid response to the later request. If CoAP is used over a reliable and ordered transport such as TCP with TLS, no messages can be delivered before the delayed message. If CoAP is used over an unreliable and unordered transport

such as UDP with DTLS, other messages can be delivered before the delayed message as long as the delayed packet is delivered inside the replay window. 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.

The attack can be performed by an attacker on the wire, or an attacker simultaneously recording what the server transmits while at the same time jamming the client. As (D)TLS encrypts the Token, the attacker needs to predict when the Token is resused. How hard that is depends on the CoAP library, but some implementations are known to omit the Token as much as possible and others lets the application chose the Token. If the response is a "piggybacked response", the client may additionally check the Message ID and drop it on mismatch. That doesn’t make the attack impossible, but lowers the probability.

The response delay and mismatch attack is illustrated in Figure 5.

```
            Client   Foe   Server
               |      |      |
               +------------>|      Code: 0.03 (PUT)
               | PUT  |      |     Token: 0x77
               |      |      |  Uri-Path: lock
               |      |      |   Payload: 0 (Unlock)
               |      |      |
               |      @<-----+      Code: 2.04 (Changed)
               |      | 2.04 |     Token: 0x77
               |      |      |
                 ....   ....
               |      |      |
               +----->X      |      Code: 0.03 (PUT)
               | PUT  |      |     Token: 0x77
               |      |      |  Uri-Path: lock
               |      |      |   Payload: 0 (Lock)
               |      |      |
               <------@      |      Code: 2.04 (Changed)
               | 2.04 |      |     Token: 0x77
               |      |      |
```

Figure 5: Delaying and mismatching response to PUT

If we once again take a lock as an example, the security consequences may be severe as the client receives a response message likely to be interpreted as confirmation of a locked door, while the received response message is in fact confirming an earlier unlock of the door. As the client is likely to leave the (believed to be locked) door unattended, the attacker may enter the home, enterprise, or car protected by the lock.

The same attack may be performed on sensors. As illustrated in Figure 6, an attacker may convince the client that the lock is locked, when it in fact is not. The "Unlock" request may be also be sent by another client authorized to control the lock.

```
Client   Foe   Server
   |      |      |
   +------------>|      Code: 0.01 (GET)
   | GET  |      |     Token: 0x77
   |      |      |  Uri-Path: lock
   |      |      |
   |      @<-----+      Code: 2.05 (Content)
   |      | 2.05 |     Token: 0x77
   |      |      |   Payload: 1 (Locked)
   |      |      |
   +------------>|      Code: 0.03 (PUT)
   | PUT  |      |     Token: 0x34
   |      |      |  Uri-Path: lock
   |      |      |   Payload: 1 (Unlock)
   |      |      |
   |      X<-----+      Code: 2.04 (Changed)
   |      | 2.04 |     Token: 0x34
   |      |      |
   +----->X      |      Code: 0.01 (GET)
   | GET  |      |     Token: 0x77
   |      |      |  Uri-Path: lock
   |      |      |
   <------@      |      Code: 2.05 (Content)
   | 2.05 |      |     Token: 0x77
   |      |      |   Payload: 1 (Locked)
   |      |      |
```

Figure 6: Delaying and mismatching response to GET

As illustrated in Figure 7, an attacker may even mix responses from different resources as long as the two resources share the same (D)TLS connection on some part of the path towards the client. This can happen if the resources are located behind a common gateway, or are served by the same CoAP proxy. An on-path attacker (not necessarily a (D)TLS endpoint such as a proxy) may e.g., deceive a client that the living room is on fire by responding with an earlier delayed response from the oven (temperatures in degree Celsius).

```
           Client   Foe   Server
              |      |      |
              +------------>|      Code: 0.01 (GET)
              | GET  |       |     Token: 0x77
              |      |       |  Uri-Path: oven/temperature
              |      |      |
              |      @<-----+      Code: 2.05 (Content)
              |       | 2.05 |     Token: 0x77
              |      |       |   Payload: 225
              |      |      |
              .....  ....
              |      |      |
              +----->X      |      Code: 0.01 (GET)
              | GET  |       |     Token: 0x77
              |      |       |  Uri-Path: livingroom/temperature
              |      |      |
              <------@      |      Code: 2.05 (Content)
              | 2.05 |       |     Token: 0x77
              |      |       |   Payload: 225
              |      |      |
```

Figure 7: Delaying and mismatching response from other resource

Remedy: Section 4.2 of [I-D.ietf-core-echo-request-tag] formally updates the client token processing for CoAP [RFC7252]. Following this updated processing mitigates the attack.

2.4. The Request Fragment Rearrangement Attack

These attack scenarios show that the Request Delay and Block Attacks can be used against block-wise transfers to cause unauthorized operations to be performed on the server, and responses to unauthorized operations to be mistaken for responses to authorized operations. The combination of these attacks is described as a separate attack because it makes the Request Delay Attack relevant to systems that are otherwise not time-dependent, which means that they could disregard the Request Delay Attack.

This attack works even if the individual request/response pairs are encrypted, authenticated and protected against the Response Delay and Mismatch Attack, provided the attacker is on the network path and can correctly guess which operations the respective packages belong to.

The attacks can be performed on any security protocol where the attacker can delay the delivery of a message unnoticed. This incluses DTLS, IPsec, and most OSCORE configurations. The attacks does not work on TCP with TLS or OSCORE (with TLS-like sequence number handling) as in these cases no messages can be delivered before the delayed message.

2.4.1. Completing an Operation with an Earlier Final Block

In this scenario (illustrated in Figure 8), blocks from two operations on a POST-accepting resource are combined to make the server execute an action that was not intended by the authorized client. This works only if the client attempts a second operation after the first operation failed (due to what the attacker made appear like a network outage) within the replay window. The client does not receive a confirmation on the second operation either, but, by the time the client acts on it, the server has already executed the unauthorized action.

```
Client   Foe   Server
   |      |      |
   +------------->    POST "incarcerate" (Block1: 0, more to come)
   |      |      |
   <-------------+    2.31 Continue (Block1: 0 received, send more)
   |      |      |
   +----->@      |    POST "valjean" (Block1: 1, last block)
   |      |      |
   +----->X      |    All retransmissions dropped
   |      |      |

(Client: Odd, but let's go on and promote Javert)

   |      |      |
   +------------->    POST "promote" (Block1: 0, more to come)
   |      |      |
   |      X<-----+    2.31 Continue (Block1: 0 received, send more)
   |      |      |
   |      @------>    POST "valjean" (Block1: 1, last block)
   |      |      |
   |      X<-----+    2.04 Valjean Promoted
   |      |      |
```

Figure 8: Completing an operation with an earlier final block

Remedy: If a client starts new block-wise operations on a security context that has lost packets, it needs to label the fragments in such a way that the server will not mix them up.

A mechanism to that effect is described as Request-Tag [I-D.ietf-core-echo-request-tag]. Had it been in place in the example and used for body integrity protection, the client would have set the Request-Tag option in the "promote" request. Depending on the server's capabilities and setup, either of four outcomes could have occurred:

1. The server could have processed the reinjected POST "valjean" as belonging to the original "incarcerate" block; that's the expected case when the server can handle simultaneous block transfers.

2. The server could respond 5.03 Service Unavailable, including a Max-Age option indicating how long it prefers not to take any requests that force it to overwrite the state kept for the "incarcerate" request.

3. The server could decide to drop the state kept for the "incarcerate" request's state, and process the "promote" request. The reinjected POST "valjean" will then fail with 4.08 Request Entity incomplete, indicating that the server does not have the start of the operation any more.

2.4.2. Injecting a Withheld First Block

If the first block of a request is withheld by the attacker for later use, it can be used to have the server process a different request body than intended by the client. Unlike in the previous scenario, it will return a response based on that body to the client.

Again, a first operation (that would go like "Girl stole apple. What shall we do with her?" - "Set her free.") is aborted by the proxy, and a part of that operation is later used in a different operation to prime the server for responding leniently to another operation that would originally have been "Evil Queen poisoned apple. What shall we do with her?" - "Lock her up.". The attack is illustrated in Figure 9.

```
Client   Foe   Server
   |      |      |
   +----->@      |     POST "Girl stole apple. Wh"
   |      |      |         (Block1: 0, more to come)
```

(Client: We'll try that one later again; for now, we have something more urgent:)

```
   |      |      |
   +------------->    POST "Evil Queen poisened apple. Wh"
   |      |      |         (Block1: 0, more to come)
   |      |      |
   |      @<-----+    2.31 Continue (Block1: 0 received, send more)
   |      |      |
   |      @------>    POST "Girl stole apple. Wh"
   |      |      |         (Block1: 0, more to come)
   |      |      |
   |      X<-----+    2.31 Continue (Block1: 0 received, send more)
   |      |      |
   <------@      |    2.31 Continue (Block1: 0 received, send more)
   |      |      |
   +------------->    POST "at shall we do with her?"
   |      |      |         (Block1: 1, last block)
   |      |      |
   <-------------+    2.05 "Set her free."
   |      |      |         (Block1: 1 received and this is the result)
```

Figure 9: Injecting a withheld first block

The remedy described in Section 2.4.1 works also for this case. Note that merely requiring that blocks of an operation should have incrementing sequence numbers would be insufficient to remedy this attack.

2.4.3. Attack difficulty

The success of any fragment rearrangement attack has multiple prerequisites:

* A client sends different block-wise requests that are only distinguished by their content.

  This is generally rare in typical CoRE applications, but can happen when the bodies of FETCH requests exceed the fragmentation threshold, or when SOAP patterns are emulated.

* A client starts later block-wise operations after an earlier one has failed.

This happens regularly as a consequence of operating in a low-power and lossy network: Losses can cause failed operation (especially when the network is unavailable for time exceeding the "few expected round-trips" they may be limited to per [RFC7959]), and the cost of reestablishing a security context.

* The attacker needs to be able to determine which packets contain which fragments.

  This can be achieved by an on-path attacker by observing request timing, or simply by observing request sizes in the case when a body is split into precisely two blocks.

It is _not_ a prerequisite that the resulting misassembled request body is syntactically correct: As the server erroneously expects the body to be integrity protected from an authorized source, it might be using a parser not suitable for untrusted input. Such a parser might crash the server in extreme cases, but might also produce a valid but incorrect response to the request the client associates the response with. Note that many constrained applications aim to minimize traffic and thus employ compact data formats; that compactness leaves little room for syntactically invalid messages.

The attack is easier if the attacker has control over the request bodies (which would be the case when a trusted proxy validates the attacker's authorization to perform two given requests, and an attack on the path between the proxy and the server recombines the blocks to a semantically different request). Attacks of that shape can easily result in reassembled bodies chosen by the attacker, but no services are currently known that operate in this way.

Summarizing, it is unlikely that an attacker can perform any of the fragment rearrangement attacks on any given system - but given the diversity of applications built on CoAP, it is easily to imagine that single applications would be vulnerable. As block-wise transfer is a basic feature of CoAP and its details are sometimes hidden behind abstractions or proxies, application authors can not be expected to design their applications with these attacks in mind, and mitigation on the protocol level is prudent.

2.5. The Relay Attack

Yet another type of attack can be performed in deployments where actuator actions are triggered automatically based on proximity and without any user interaction, e.g., a car (the client) constantly polling for the car key (the server) and unlocking both doors and engine as soon as the car key responds. An attacker (or pair of attackers) may simply relay the CoAP messages out-of-band, using for

examples some other radio technology. By doing this, the actuator (i.e., the car) believes that the client is close by and performs actions based on that false assumption. The attack is illustrated in Figure 10. In this example the car is using an application specific challenge-response mechanism transferred as CoAP payloads.

```
Client   Foe         Foe   Server
   |      |           |      |
   +----->| ......... +----->|      Code: 0.02 (POST)
   | POST |           | POST |     Token: 0x3a
   |      |           |      |  Uri-Path: lock
   |      |           |      |   Payload: JwePR2iCe8b0ux (Challenge)
   |      |           |      |
   |<-----+ ......... |<-----+      Code: 2.04 (Changed)
   | 2.04 |           | 2.04 |     Token: 0x3a
   |      |           |      |   Payload: RM8i13G8D5vfXK (Response)
   |      |           |      |
```

Figure 10: Relay attack (the client is the actuator)

The consequences may be severe, and in the case of a car, lead to the attacker unlocking and driving away with the car, an attack that unfortunately is happening in practice.

Remedy: Getting a response over a short-range radio cannot be taken as proof of proximity and can therefore not be used to take actions based on such proximity. Any automatically triggered mechanisms relying on proximity need to use other stronger mechanisms to establish proximity. Mechanisms that can be used are: measuring the round-trip time and calculating the maximum possible distance based on the speed of light, or using radio with an extremely short range like NFC (centimeters instead of meters). Another option is to include geographical coordinates (from e.g., GPS) in the messages and calculate proximity based on these, but in this case the location measurements need to be very precise and the system need to make sure that an attacker cannot influence the location estimation. Some types of global navigation satellite systems (GNSS) receivers are vulnerable to spoofing attacks.

3. Security Considerations

The whole document can be seen as security considerations for CoAP.

4. IANA Considerations

This document has no actions for IANA.

5. Informative References

[I-D.ietf-core-echo-request-tag]
Amsüss, C., Mattsson, J. P., and G. Selander, "CoAP: Echo, Request-Tag, and Token Processing", Work in Progress, Internet-Draft, draft-ietf-core-echo-request-tag-14, 4 October 2021, <https://www.ietf.org/archive/id/draft-ietf-core-echo-request-tag-14.txt>.

[I-D.ietf-lake-edhoc]
Selander, G., Mattsson, J. P., and F. Palombini, "Ephemeral Diffie-Hellman Over COSE (EDHOC)", Work in Progress, Internet-Draft, draft-ietf-lake-edhoc-12, 20 October 2021, <https://www.ietf.org/archive/id/draft-ietf-lake-edhoc-12.txt>.

[I-D.ietf-tls-dtls13]
Rescorla, E., Tschofenig, H., and N. Modadugu, "The Datagram Transport Layer Security (DTLS) Protocol Version 1.3", Work in Progress, Internet-Draft, draft-ietf-tls-dtls13-43, 30 April 2021, <https://www.ietf.org/archive/id/draft-ietf-tls-dtls13-43.txt>.

[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>.

[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>.

[RFC8152] Schaad, J., "CBOR Object Signing and Encryption (COSE)", RFC 8152, DOI 10.17487/RFC8152, July 2017, <https://www.rfc-editor.org/info/rfc8152>.

[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>.

[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>.

[RFC8613] Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security for Constrained RESTful Environments (OSCORE)", RFC 8613, DOI 10.17487/RFC8613, July 2019, <https://www.rfc-editor.org/info/rfc8613>.

Acknowledgements

The authors would like to thank Carsten Bormann, Klaus Hartke, Jaime Jiménez, Ari Keränen, Matthias Kovatsch, Achim Kraus, Sandeep Kumar, and András Méhes for their valuable comments and feedback.

Authors' Addresses

John Preuß Mattsson
Ericsson AB
SE-164 80 Stockholm
Sweden

Email: john.mattsson@ericsson.com

John Fornehed
Ericsson AB
SE-164 80 Stockholm
Sweden

Email: john.fornehed@ericsson.com

Göran Selander
Ericsson AB
SE-164 80 Stockholm
Sweden

Email: goran.selander@ericsson.com

Francesca Palombini
Ericsson AB
SE-164 80 Stockholm
Sweden

Email: francesca.palombini@ericsson.com

Christian Amsüss
Energy Harvesting Solutions

Email: c.amsuess@energyharvesting.at

CoRE Working Group
Internet-Draft
Updates: 8613 (if approved)
Intended status: Standards Track
Expires: 11 January 2024

M. Tiloca
R. Höglund
RISE AB

10 July 2023

OSCORE-capable Proxies
draft-tiloca-core-oscore-capable-proxies-07

Abstract

Object Security for Constrained RESTful Environments (OSCORE) can be used to protect CoAP messages end-to-end between two endpoints at the application layer, also in the presence of intermediaries such as proxies. This document defines how to use OSCORE for protecting CoAP messages also between an origin application endpoint and an intermediary, or between two intermediaries. Also, it defines how to secure a CoAP message by applying multiple, nested OSCORE protections, e.g., both end-to-end between origin application endpoints, as well as between an application endpoint and an intermediary or between two intermediaries. Thus, this document updates RFC 8613. The same approach can be seamlessly used with Group OSCORE, for protecting CoAP messages when group communication with intermediaries is used.

Discussion Venues

This note is to be removed before publishing as an RFC.

Discussion of this document takes place on the Constrained RESTful Environments Working Group mailing list (core@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/core/.

Source for this draft and an issue tracker can be found at https://gitlab.com/crimson84/draft-tiloca-core-oscore-to-proxies.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction 3
  1.1. Terminology 4
2. Use Cases 5
  2.1. CoAP Group Communication with Proxies 5
  2.2. CoAP Observe Notifications over Multicast 6
  2.3. LwM2M Client and External Application Server 7
  2.4. LwM2M Gateway 7
  2.5. Further Use Cases 8
3. Message Processing 10
  3.1. General Rules on Protecting Options 10
  3.2. Processing an Outgoing Request 12
  3.3. Processing an Incoming Request 12
  3.4. Processing an Outgoing Response 15
  3.5. Processing an Incoming Response 15
4. Caching of OSCORE-Protected Responses 16
5. Establishment of OSCORE Security Contexts 17
6. CoAP Header Compression with SCHC 18
7. Security Considerations 19
8. IANA Considerations 20
9. References 20
  9.1. Normative References 20
  9.2. Informative References 20
Appendix A. Examples 23
  A.1. Example 1 24
  A.2. Example 2 26

A.3. Example 3 28
A.4. Example 4 30
A.5. Example 5 35
Appendix B. State Diagram of the Incoming Request Processing . . 39
Acknowledgments 41
Authors' Addresses 41

1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] supports the presence of intermediaries, such as forward-proxies and reverse-proxies, which assist origin clients by performing requests to origin servers on their behalf, and forwarding back the related responses.

CoAP supports also group communication scenarios [I-D.ietf-core-groupcomm-bis], where clients can send a one-to-many request targeting all the servers in the group, e.g., by using IP multicast. Like for one-to-one communication, group settings can also rely on intermediaries [I-D.tiloca-core-groupcomm-proxy].

The protocol Object Security for Constrained RESTful Environments (OSCORE) [RFC8613] can be used to protect CoAP messages between two endpoints at the application layer, especially achieving end-to-end security in the presence of (non-trusted) intermediaries. When CoAP group communication is used, the same can be achieved by means of the protocol Group OSCORE [I-D.ietf-core-oscore-groupcomm].

For a number of use cases (see Section 2), it is required and/or beneficial that communications are secured also between an application endpoint (i.e., a CoAP origin client/server) and an intermediary, as well as between two adjacent intermediaries in a chain. This especially applies to the communication leg between the CoAP origin client and the adjacent intermediary acting as next hop towards the CoAP origin server.

In such cases, and especially if the origin client already uses OSCORE to achieve end-to-end security with the origin server, it would be convenient that OSCORE is used also to secure communications between the origin client and its next hop. However, the original specification [RFC8613] does not define how OSCORE can be used to protect CoAP messages in such communication leg, which would require to consider also the intermediary as an "OSCORE endpoint".

This document fills this gap, and updates [RFC8613] as follows.

* It defines how to use OSCORE for protecting a CoAP message in the communication leg between: i) an origin client/server and an intermediary; or ii) two adjacent intermediaries in an intermediary chain. That is, besides origin clients/servers, it allows also intermediaries to be possible "OSCORE endpoints".

* It admits a CoAP message to be secured by multiple, nested OSCORE protections applied in sequence, as an "OSCORE-in-OSCORE" process. For instance, this is the case when the message is OSCORE-protected end-to-end between the origin client and origin server, and the result is further OSCORE-protected over the leg between the current and next hop (e.g., the origin client and the adjacent intermediary acting as next hop towards the origin server).

This document does not specify any new signaling method to guide the message processing on the different endpoints. In particular, every endpoint is always able to understand what steps to take on an incoming message depending on the presence of the OSCORE Option, as exclusively included or instead combined together with CoAP options intended for an intermediary.

The approach defined in this document can be seamlessly adopted also when Group OSCORE is used, for protecting CoAP messages in group communication scenarios that rely on intermediaries.

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 related to CoAP [RFC7252]; OSCORE [RFC8613] and Group OSCORE [I-D.ietf-core-oscore-groupcomm]. This document especially builds on concepts and mechanics related to intermediaries such as CoAP forward-proxies.

In addition, this document uses the following terms.

* Source application endpoint: an origin client producing a request, or an origin server producing a response.

* Destination application endpoint: an origin server intended to consume a request, or an origin client intended to consume a response.

* Application endpoint: a source or destination application endpoint.

* Source OSCORE endpoint: an endpoint protecting a message with OSCORE or Group OSCORE.

* Destination OSCORE endpoint: an endpoint unprotecting a message with OSCORE or Group OSCORE.

* OSCORE endpoint: a source/destination OSCORE endpoint. An OSCORE endpoint is not necessarily also an application endpoint with respect to a certain message.

* Proxy-related options: either of the following (set of) CoAP options used for proxying a CoAP request.

  - The Proxy-Uri Option. This is relevant when using a forward-proxy.

  - The set of CoAP options comprising the Proxy-Scheme Option together with any of the Uri-* Options. This is relevant when using a forward-proxy.

  - One or more Uri-Path Options, when used not together with the Proxy-Scheme Option. This is relevant when using a reverse-proxy.

* OSCORE-in-OSCORE: the process by which a message protected with (Group) OSCORE is further protected with (Group) OSCORE. This means that, if such a process is used, a successful decryption/verification of an OSCORE-protected message might yield an OSCORE-protected message.

2. Use Cases

The approach defined in this document has been motivated by a number of use cases, which are summarized below.

2.1. CoAP Group Communication with Proxies

CoAP supports also one-to-many group communication, e.g., over IP multicast [I-D.ietf-core-groupcomm-bis], which can be protected end-to-end between origin client and origin servers by using Group OSCORE [I-D.ietf-core-oscore-groupcomm].

This communication model can be assisted by intermediaries such as a CoAP forward-proxy or reverse-proxy, which relays a group request to the origin servers. If Group OSCORE is used, the proxy is

intentionally not a member of the OSCORE group. Furthermore, [I-D.tiloca-core-groupcomm-proxy] defines a signaling protocol between origin client and proxy, to ensure that responses from the different origin servers are forwarded back to the origin client within a time interval set by the client, and that they can be distinguished from one another.

In particular, it is required that the proxy identifies the origin client as allowed-listed, before forwarding a group request to the servers (see Section 4 of [I-D.tiloca-core-groupcomm-proxy]). This requires a security association between the origin client and the proxy, which would be convenient to provide with a dedicated OSCORE Security Context between the two, since the client is possibly using also Group OSCORE with the origin servers.

2.2. CoAP Observe Notifications over Multicast

The Observe extension for CoAP [RFC7641] allows a client to register its interest in "observing" a resource at a server. The server can then send back notification responses upon changes to the resource representation, all matching with the original observation request.

In some applications, such as pub-sub [I-D.ietf-core-coap-pubsub], multiple clients are interested to observe the same resource at the same server. Hence, [I-D.ietf-core-observe-multicast-notifications] defines a method that allows the server to send a multicast notification to all the observer clients at once, e.g., over IP multicast. To this end, the server synchronizes the clients by providing them with a common "phantom observation request", against which the following multicast notifications will match.

In case the clients and the server use Group OSCORE for end-to-end security and a proxy is also involved, an additional step is required (see Section 12 of [I-D.ietf-core-observe-multicast-notifications]). That is, clients are in turn required to provide the proxy with the obtained "phantom observation request", thus enabling the proxy to receive the multicast notifications from the server.

Therefore, it is preferable to have a security association also between each client and the proxy, to especially ensure the integrity of that information provided to the proxy (see Section 15.3 of [I-D.ietf-core-observe-multicast-notifications]). Like for the use case in Section 2.1, this would be conveniently achieved with a dedicated OSCORE Security Context between a client and the proxy, since the client is also using Group OSCORE with the origin server.

2.3. LwM2M Client and External Application Server

The Lightweight Machine-to-Machine (LwM2M) protocol [LwM2M-Core] enables a LwM2M Client device to securely bootstrap and then register at a LwM2M Server, with which it will perform most of its following communication exchanges. As per the transport bindings specification of LwM2M [LwM2M-Transport], the LwM2M Client and LwM2M Server can use CoAP and OSCORE to secure their communications at the application layer, including during the device registration process.

Furthermore, Section 5.5.1 of [LwM2M-Transport] specifies that: "OSCORE MAY also be used between LwM2M endpoint and non-LwM2M endpoint, e.g., between an Application Server and a LwM2M Client via a LwM2M server. Both the LwM2M endpoint and non-LwM2M endpoint MUST implement OSCORE and be provisioned with an OSCORE Security Context."

In such a case, the LwM2M Server can practically act as forward-proxy between the LwM2M Client and the external Application Server. At the same time, the LwM2M Client and LwM2M Server must continue protecting communications on their leg using their Security Context. Like for the use case in Section 2.1, this also allows the LwM2M Server to identify the LwM2M Client, before forwarding its request outside the LwM2M domain and towards the external Application Server.

2.4. LwM2M Gateway

The specification [LwM2M-Gateway] extends the LwM2M architecture by defining the LwM2M Gateway functionality. That is, a LwM2M Server can manage end IoT devices "behind" the LwM2M Gateway. While it is outside the scope of such specification, it is possible for the LwM2M Gateway to use any suitable protocol with its connected end IoT devices, as well as to carry out any required protocol translation.

Practically, the LwM2M Server can send a request to the LwM2M Gateway, asking to forward it to an end IoT device. With particular reference to the CoAP protocol and the related transport binding specified in [LwM2M-Transport], the LwM2M Server acting as CoAP client sends its request to the LwM2M Gateway acting as CoAP server.

If CoAP is used in the communication leg between the LwM2M Gateway and the end IoT devices, then the LwM2M Gateway fundamentally acts as a reverse-proxy (see Section 5.7.3 of [RFC7252]). That is, in addition to its own resources, the LwM2M Gateway serves the resources of each end IoT device behind itself, as exposed under a dedicated URI-Path. As per [LwM2M-Gateway], the first URI-Path segment is used as "prefix" to identify the specific IoT device, while the remaining URI-Path segments specify the target resource at the IoT device.

As per Section 7 of [LwM2M-Gateway], message exchanges between the LwM2M Server and the L2M2M Gateway are secured using the LwM2M-defined technologies, while the LwM2M protocol does not provide end-to-end security between the LwM2M Server and the end IoT devices. However, the approach defined in this document makes it possible to achieve both goals, by allowing the LwM2M Server to use OSCORE for protecting a message both end-to-end for the targeted end IoT device as well as for the LwM2M Gateway acting as reverse-proxy.

2.5. Further Use Cases

The approach defined in this document can be useful also in the following use cases relying on a proxy.

* A server aware of a suitable cross proxy can rely on it as a third-party service, in order to indicate transports for CoAP available to that server (see Section 4 of [I-D.ietf-core-transport-indication]).

  From a security point of view, it would be convenient if the proxy could provide suitable credentials to the client, as a general trusted proxy for the system. At the same time, it can be desirable to limit the use of such a proxy to a set of clients which have permission to use it, and that the proxy can identify through a secure communication association.

  However, in order for OSCORE to be an applicable security mechanism for this scenario, OSCORE has to be terminated at the proxy. That is, it would be required for a client and the proxy to share a dedicated OSCORE Security Context and to use it for protecting their communication leg.

* The method specified in [I-D.ietf-core-coap-pm] relies on the Performance Measurement Option to enable network telemetry for CoAP communications. This makes it possible to efficiently measure Round-Trip Time and message losses, both end-to-end and hop-by-hop. In particular, on-path probes such as intermediary proxies can be deployed to perform measurements hop-by-hop.

  When OSCORE is used in deployments including on-path probes, an inner Performance Measurement Option is protected end-to-end between the two application endpoints and enables end-to-end measurements between those. At the same time, an outer Performance Measurement Option allows also hop-by-hop measurements to be performed by reying on an on-path probe.

Therefore, it is preferable to have a secure association with an on-path probe, in order to also ensure the integrity of the hop-by-hop measurements exchanged with the probe.

* The method specified in [I-D.ietf-ace-coap-est-oscore] enables public-key certificate enrollment for Internet of Things deployments. This leverages payload formats defined in Enrollment over Secure Transport (EST) [RFC7030], while relying on CoAP for message transfer and on OSCORE for message protection.

  In real-world deployments, an EST server issuing public-key certificates may reside outside a constrained network that includes devices acting as EST clients. In particular, the EST clients are expected to support only CoAP, while the EST server in a non-constrained network is expected to support only HTTP. This requires a CoAP-to-HTTP proxy to be deployed between the EST clients and the EST server, in order to map CoAP messages with HTTP messages across the two networks.

  Even in such a scenario, the EST server and every EST client can still effectively use OSCORE to protect their communications end-to-end. At the same time, it is desirable to have an additional secure association between the EST client and the CoAP-to-HTTP proxy, especially in order for the proxy to identify the EST client before forwarding EST messages out of the CoAP boundary of the constrained network and towards the EST server.

* A proxy may be deployed to act as an entry point to a firewalled network, which only authenticated clients can join. In particular, authentication can rely on the used secure communication association between a client and the proxy. If the proxy could share a dedicated OSCORE Security Context with each client, the proxy can rely on it to identify the client, before forwarding its messages to any other member of the firewalled network.

* The approach defined in this document does not pose a limit to the number of OSCORE protections applied to the same CoAP message. This enables more privacy-oriented scenarios based on proxy chains, where the origin client protects a CoAP request using first the OSCORE Security Context shared with the origin server, and then the dedicated OSCORE Security Context shared with each of the different hops in the chain. Once received at a chain hop, the request would be stripped of the OSCORE protection associated with that hop before being forwarded to the next one.

3. Message Processing

As mentioned in Section 1, this document introduces the following two main deviations from the original OSCORE specification [RFC8613].

1. An "OSCORE endpoint", i.e., a producer/consumer of an OSCORE Option can be not only an application endpoint (i.e., an origin client or server), but also an intermediary such as a proxy.

   Hence, OSCORE can be used between an origin client/server and a proxy, as well as between two proxies in an intermediary chain.

2. A CoAP message can be secured by multiple OSCORE protections applied in sequence. Therefore, the final result is a message with nested OSCORE protections, as the output of an "OSCORE-in-OSCORE" process. Hence, following a decryption, the resulting message might legitimately include an OSCORE Option, and thus have in turn to be decrypted.

   The most common case is expected to consider a message protected with up to two OSCORE layers, i.e.: i) an inner layer, protecting the message end-to-end between the origin client and the origin server acting as application endpoints; and ii) an outer layer, protecting the message between a certain OSCORE endpoint and the other OSCORE endpoint adjacent in the intermediary chain.

   However, a message can also be protected with a higher arbitrary number of nested OSCORE layers, e.g., in scenarios relying on a longer chain of intermediaries. For instance, the origin client can sequentially apply multiple OSCORE layers to a request, each of which to be consumed and removed by one of the intermediaries in the chain, until the origin server is reached and it consumes the innermost OSCORE layer.

Appendix A provides a number of examples where the approach defined in this document is used to protect message exchanges.

3.1. General Rules on Protecting Options

Let us consider a sender endpoint that, when protecting an outgoing message, applies the i-th OSCORE layer in sequence, by using the OSCORE Security Context shared with another OSCORE endpoint X.

In addition to the CoAP options specified as class E in [RFC8613] or in the document defining them, the sender endpoint MUST encrypt and integrity-protect the following CoAP options. That is, even if they are originally specified as class U or class I for OSCORE, such options are processed like if they were specified as class E.

* Any CoAP option such that both the following conditions hold.

  1. The sender endpoint has added the option to the message either:

     - Following a message protection previously performed for an OSCORE endpoint different than X; or

     - Before a message protection previously performed for an OSCORE endpoint different than X, where the option was treated as Class U or I.

  2. The option is not intended to be consumed by the other OSCORE endpoint X.

  Examples of such CoAP options are:

  - The OSCORE Option present as the result of the OSCORE layer immediately previously applied for an OSCORE endpoint different than X, when the sender endpoint is an origin endpoint.

  - The EDHOC Option defined in [I-D.ietf-core-oscore-edhoc], when the sender endpoint is the origin client.

  - The Request-Hash Option defined in [I-D.amsuess-core-cachable-oscore], when X is not an origin endpoint).

* Any CoAP option such that all the following conditions hold.

  1. The sender endpoint has added the option to the message.

  2. The option is intended to be consumed by the other OSCORE endpoint X.

  3. At the other OSCORE endpoint X, the option does not play a role in processing the message before having removed the i-th OSCORE layer or in removing the i-th OSCORE layer altogether.

  Examples of such CoAP options are:

  - The Proxy-Uri, Proxy-Scheme, Uri-Host and Uri-Port Options defined in [RFC7252].

  - The Listen-To-Multicast-Notifications Option defined in [I-D.ietf-core-observe-multicast-notifications].

- The Multicast-Timeout, Response-Forwarding and Group-ETag Options defined in [I-D.tiloca-core-groupcomm-proxy].

* Any CoAP option such that the sender endpoint has not added the option to the message.

  Examples of such CoAP options are:

  - The OSCORE Option present as the result of the OSCORE layer immediately previously applied for an OSCORE endpoint different than X, when the sender endpoint is not an origin endpoint.

  - The EDHOC Option defined in [I-D.ietf-core-oscore-edhoc], when the sender endpoint is not the origin client.

3.2. Processing an Outgoing Request

The rules from Section 3.1 apply when processing an outgoing request message, with the following addition.

When an application endpoint applies multiple OSCORE layers in sequence to protect an outgoing request, and it uses an OSCORE Security Context shared with the other application endpoint, then the first OSCORE layer MUST be applied by using that Security Context.

3.3. Processing an Incoming Request

Upon receiving a request REQ, the recipient endpoint performs the actions described in the following steps.

1. If REQ includes proxy-related options, the endpoint moves to step 2. Otherwise, the endpoint moves to step 3.

2. The endpoint proceeds as defined below, depending on which of the two following conditions holds.

   * REQ includes either the Proxy-Uri Option, or the Proxy-Scheme Option together with any of the Uri-* Options.

     If the endpoint is not configured to be a forward-proxy, it MUST stop processing the request and MUST respond with a 5.05 (Proxying Not Supported) error response to (the previous hop towards) the origin client, as per Section 5.10.2 of [RFC7252]. This may result in protecting the error response over that communication leg, as per Section 3.4.

Otherwise, the endpoint MUST check whether forwarding this request to (the next hop towards) the origin server is an authorized operation. This check can be based, for instance, on the specific OSCORE Security Context that the endpoint used to decrypt the incoming message, before performing this step.

In case the authentication check fails, the endpoint MUST stop processing the request and MUST respond with a 4.01 (Unauthorized) error response to (the previous hop towards) the origin client, as per Section 5.10.2 of [RFC7252]. This may result in protecting the error response over that communication leg, as per Section 3.4.

Otherwise, the endpoint consumes the proxy-related options as per Section 5.7.2 of [RFC7252] and forwards REQ to (the next hop towards) the origin server, unless differently indicated in REQ, e.g., by means of any of its CoAP options. For instance, a forward-proxy does not forward a request that includes proxy-related options together with the Listen-To-Multicast-Notifications Option (see Section 12 of [I-D.ietf-core-observe-multicast-notifications]).

If the endpoint forwards REQ to (the next hop towards) the origin server, this may result in (further) protecting REQ over that communication leg, as per Section 3.2.

After that, the endpoint does not take any further action.

* REQ includes one or more Uri-Path Options but not the Proxy-Scheme Option.

  If the endpoint is not configured to be a reverse-proxy or its resource targeted by the Uri-Path Options is not intended to support reverse-proxy functionalities, then the endpoint proceeds to step 3.

  Otherwise, the endpoint MUST check whether forwarding this request to (the next hop towards) the origin server is an authorized operation. This check can be based, for instance, on the specific OSCORE Security Context that the endpoint used to decrypt the incoming message, before performing this step.

  In case the authentication check fails, the endpoint MUST stop processing the request and MUST respond with a 4.01 (Unauthorized) error response to (the previous hop towards) the origin client, as per Section 5.10.2 of [RFC7252]. This may result in protecting the error response over that communication leg, as per Section 3.4.

Otherwise, the endpoint consumes the Uri-Path options as per Section 5.7.3 of [RFC7252], and forwards REQ to (the next hop towards) the origin server, unless differently indicated in REQ, e.g., by means of any of its CoAP options.

If the endpoint forwards REQ to (the next hop towards) the origin server, this may result in (further) protecting REQ over that communication leg, as per Section 3.2.

After that, the endpoint does not take any further action.

Note that, when forwarding REQ, the endpoint might not remove all the Uri-Path Options originally present, e.g., in case the next hop towards the origin server is a further reverse-proxy.

3. The endpoint proceeds as defined below, depending on which of the two following conditions holds.

   * REQ does not include an OSCORE Option.

     If the endpoint does not have an application to handle REQ, it MUST stop processing the request and MAY respond with a 4.00 (Bad Request) error response to (the previous hop towards) the origin client. This may result in protecting the error response over that communication leg, as per Section 3.4.

     Otherwise, the endpoint delivers REQ to the application.

   * REQ includes an OSCORE Option.

     If REQ includes any URI-Path Options, the endpoint MUST stop processing the request and MAY respond with a 4.00 (Bad Request) error response to (the previous hop towards) the origin client. This may result in protecting the error response over that communication leg, as per Section 3.4.

     Otherwise, the endpoint decrypts REQ using the OSCORE Security Context indicated by the OSCORE Option, i.e., REQ* = dec(REQ). After that, the possible presence of an OSCORE Option in the decrypted request REQ* is not treated as an error situation.

If the OSCORE processing results in an error, the endpoint MUST stop processing the request and performs error handling as per Section 8.2 of [RFC8613] or Sections 8.2 and 9.4 of [I-D.ietf-core-oscore-groupcomm], in case OSCORE or Group OSCORE is used, respectively. In case the endpoint sends an error response to (the previous hop towards) the origin client, this may result in protecting the error response over that communication leg, as per Section 3.4.

Otherwise, REQ takes REQ*, and the endpoint moves to step 1.

Appendix B provides an overview of the process defined above as a state diagram.

3.4. Processing an Outgoing Response

The rules from Section 3.1 apply when processing an outgoing response message, with the following additions.

When an application endpoint applies multiple OSCORE layers in sequence to protect an outgoing response, and it uses an OSCORE Security Context shared with the other application endpoint, then the first OSCORE layer MUST be applied by using that Security Context.

The sender endpoint protects the response by applying the same OSCORE layers that it removed from the corresponding incoming request, but in the reverse order than the one they were removed.

In case the response is an error response, the sender endpoint protects it by applying the same OSCORE layers that it successfully removed from the corresponding incoming request, but in the reverse order than the one they were removed.

3.5. Processing an Incoming Response

The recipient endpoint removes the same OSCORE layers that it added when protecting the corresponding outgoing request, but in the reverse order than the one they were removed.

When doing so, the possible presence of an OSCORE Option in the decrypted response following the removal of an OSCORE layer is not treated as an error situation, unless it occurs after having removed as many OSCORE layers as were added in the outgoing request. In such a case, the endpoint MUST stop processing the response.

4. Caching of OSCORE-Protected Responses

Although not possible as per the original OSCORE specification [RFC8613], cacheability of OSCORE-protected responses at proxies can be achieved. To this end, the approach defined in [I-D.amsuess-core-cachable-oscore] can be used, as based on Deterministic Requests protected with the pairwise mode of Group OSCORE [I-D.ietf-core-oscore-groupcomm] used end-to-end between an origin client and an origin server. The applicability of this approach is limited to requests that are safe (in the RESTful sense) to process and do not yield side effects at the origin server.

In particular, this approach requires both the origin client and the origin server to have already joined the correct OSCORE group. Then, starting from the same plain CoAP request, different clients in the OSCORE group are able to deterministically generate a same request protected with Group OSCORE, which is sent to a proxy for being forwarded to the origin server. The proxy can effectively cache the resulting OSCORE-protected response from the server, since the same plain CoAP request will result again in the same Deterministic Request and thus will produce a cache hit.

When using this approach, the following also applies in addition to what is defined in Section 3, when processing incoming messages at a proxy that implements caching of responses.

* Upon receiving a request from (the previous hop towards) the origin client, the proxy checks if specifically the message available during the execution of step 2 in Section 3.3 produces a cache hit.

  That is, such a message: i) is exactly the one to be forwarded to (the next hop towards) the origin server, if no cache hit has occurred; and ii) is the result of an OSCORE decryption at the proxy, if OSCORE is used on the communication leg between the proxy and (the previous hop towards) the origin client.

* Upon receiving a response from (the next hop towards) the origin server, the proxy first removes the same OSCORE layers that it added when protecting the corresponding outgoing request, as defined in Section 3.5.

  Then, the proxy stores specifically that resulting response message in its cache. That is, such a message is exactly the one to be forwarded to (the previous hop towards) the origin client.

The specific rules about serving a request with a cached response are defined in Section 5.6 of [RFC7252], as well as in Section 7 of [I-D.tiloca-core-groupcomm-proxy] for group communication scenarios.

5. Establishment of OSCORE Security Contexts

Like the original OSCORE specification [RFC8613], this document is not devoted to any particular approach that two OSCORE endpoints use for establishing an OSCORE Security Context.

At the same time, the following applies, depending on the two peers using OSCORE or Group OSCORE [I-D.ietf-core-oscore-groupcomm] to protect their communications.

* When using OSCORE, the establishment of the OSCORE Security Context can rely on the authenticated key establishment protocol EDHOC [I-D.ietf-lake-edhoc].

  Assuming the use of OSCORE both between the two origin application endpoints as well as between the origin client and the first proxy in the chain, it is expected that the origin client first runs EDHOC with the first proxy in the chain, and then with the origin server through the chain of proxies (see the example in Appendix A.4).

  Furthermore, the additional use of the combined EDHOC + OSCORE request defined in [I-D.ietf-core-oscore-edhoc] is particularly beneficial in this case (see the example in Appendix A.5), and especially when relying on a long chain of proxies.

* The use of Group OSCORE is expected to be limited between the origin applications endpoints, e.g., between the origin client and multiple origin servers. In order to join the same OSCORE group and obtain the corresponding Group OSCORE Security Context, those endpoints can use the approach defined in [I-D.ietf-ace-key-groupcomm-oscore] and based on the ACE framework for authentication and authorization in constrained environments [RFC9200].

  For the purposes of this document, there is no need for a proxy to also be a member of the OSCORE group whose Group OSCORE Security Context is used by the origin application endpoints for protecting communications end-to-end.

6. CoAP Header Compression with SCHC

The method defined in this document enables and results in the possible protection of the same CoAP message with multiple, nested OSCORE layers. Especially when this happens, it is desirable to compress the header of protected CoAP messages, in order to improve performance and ensure that CoAP is usable also in Low-Power Wide-Area Networks (LPWANs).

To this end, it is possible to use the Static Context Header Compression and fragmentation (SCHC) framework [RFC8724]. In particular, [RFC8824] specifies how to use SCHC for compressing headers of CoAP messages, also when messages are protected with OSCORE. As further clarified in [I-D.tiloca-schc-8824-update], the SCHC Compression/Decompression is applicable also in the presence of CoAP proxies, and especially to the two following cases.

* In case OSCORE is not used at all, the SCHC processing occurs hop-by-hop, by relying on SCHC Rules that are consistently shared between two adjacent hops.

* In case OSCORE is used only end-to-end between the application endpoints, then an Inner SCHC Compression/Decompression and an Outer SCHC Compression/Decompression are performed (see Section 7.2 of [RFC8824]). In particular, the following holds.

  The SCHC processing occurs end-to-end as to the Inner SCHC Compression/Decompression. This relies on Inner SCHC Rules that are shared between the two application endpoints, which act as OSCORE endpoints and share the used OSCORE Security Context.

  The SCHC processing occurs hop-by-hop as to the Outer SCHC Compression/Decompression. This relies on Outer SCHC Rules that are shared between two adjacent hops.

When using the method defined in this document, and thus enabling also an intermediary proxy to be an OSCORE endpoint, the SCHC processing above is generalized as specified below.

When processing an outgoing CoAP message, a sender endpoint proceeds as follows.

* The sender endpoint performs one Inner SCHC Compression for each OSCORE layer applied to the outgoing message. Each Inner SCHC Compression occurs before protecting the message with that OSCORE layer, and relies on the SCHC Rules that are shared with the other OSCORE endpoint.

* The sender endpoint performs exactly one Outer SCHC Compression. This occurs after having performed all the intended OSCORE protections of the outgoing message, and relies on the SCHC Rules that are shared with the (next hop towards the) recipient application endpoint.

That is, with respect to the SCHC Compression/Decompression processing, the following holds.

* An Inner SCHC Compression is intended to a recipient OSCORE endpoint, which will: first, decrypt an incoming message with the OSCORE Security Context shared with the other OSCORE endpoint; and then, perform the corresponding Inner SCHC Decompression, by relying on the SCHC Rules shared with the other OSCORE endpoint.

* An Outer SCHC Compression is intended to the (next hop towards the) recipient application endpoint, which will: first, perform a corresponding Outer SCHC Decompression on an incoming message, by relying on the SCHC Rules shared with the (previous hop towards the) recipient application endpoint; then, perform a new Outer SCHC Compression on the result, by relying on the SCHC Rules shared with the (next hop towards the) recipient application endpoint; and, finally, send the result to the (next-hop towards the) recipient application endpoint.

Note that the generalization above does not alter the core approach, design choices, and features of the SCHC Compression/Decompression applied to CoAP headers.

7. Security Considerations

The same security considerations from CoAP [RFC7252] apply to this document. The same security considerations from [RFC8613] and [I-D.ietf-core-oscore-groupcomm] apply to this document, when using OSCORE or Group OSCORE to protect exchanged messages.

Further security considerations to take into account are inherited from the specifically used CoAP options, extensions, and methods employed when relying on OSCORE or Group OSCORE.

This document does not change the security properties of OSCORE and Group OSCORE. That is, given any two OSCORE endpoints, the method defined in this document provides them with the same security guarantees that OSCORE and Group OSCORE provide in the case where such endpoints are specifically application endpoints.

8. IANA Considerations

This document has no actions for IANA.

9. References

9.1. Normative References

[I-D.ietf-core-oscore-groupcomm]
Tiloca, M., Selander, G., Palombini, F., Mattsson, J. P., and J. Park, "Group Object Security for Constrained RESTful Environments (Group OSCORE)", Work in Progress, Internet-Draft, draft-ietf-core-oscore-groupcomm-18, 22 June 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-core-oscore-groupcomm-18>.

[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/rfc/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/rfc/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/rfc/rfc8174>.

[RFC8613] Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security for Constrained RESTful Environments (OSCORE)", RFC 8613, DOI 10.17487/RFC8613, July 2019, <https://www.rfc-editor.org/rfc/rfc8613>.

[RFC8724] Minaburo, A., Toutain, L., Gomez, C., Barthel, D., and JC. Zuniga, "SCHC: Generic Framework for Static Context Header Compression and Fragmentation", RFC 8724, DOI 10.17487/RFC8724, April 2020, <https://www.rfc-editor.org/rfc/rfc8724>.

[RFC8824] Minaburo, A., Toutain, L., and R. Andreasen, "Static Context Header Compression (SCHC) for the Constrained Application Protocol (CoAP)", RFC 8824, DOI 10.17487/RFC8824, June 2021, <https://www.rfc-editor.org/rfc/rfc8824>.

9.2. Informative References

[I-D.amsuess-core-cachable-oscore]
Amsüss, C. and M. Tiloca, "Cacheable OSCORE", Work in Progress, Internet-Draft, draft-amsuess-core-cachable-oscore-06, 11 January 2023, <https://datatracker.ietf.org/doc/html/draft-amsuess-core-cachable-oscore-06>.

[I-D.ietf-ace-coap-est-oscore]
Selander, G., Raza, S., Furuhed, M., Vuini, M., and T. Claeys, "Protecting EST Payloads with OSCORE", Work in Progress, Internet-Draft, draft-ietf-ace-coap-est-oscore-02, 9 July 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-ace-coap-est-oscore-02>.

[I-D.ietf-ace-key-groupcomm-oscore]
Tiloca, M., Park, J., and F. Palombini, "Key Management for OSCORE Groups in ACE", Work in Progress, Internet-Draft, draft-ietf-ace-key-groupcomm-oscore-16, 6 March 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-ace-key-groupcomm-oscore-16>.

[I-D.ietf-core-coap-pm]
Fioccola, G., Zhou, T., Cociglio, M., Bulgarella, F., Nilo, M., and F. Milan, "Constrained Application Protocol (CoAP) Performance Measurement Option", Work in Progress, Internet-Draft, draft-ietf-core-coap-pm-00, 19 April 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-core-coap-pm-00>.

[I-D.ietf-core-coap-pubsub]
Koster, M., Keränen, A., and J. Jimenez, "A publish-subscribe architecture for the Constrained Application Protocol (CoAP)", Work in Progress, Internet-Draft, draft-ietf-core-coap-pubsub-12, 13 March 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-core-coap-pubsub-12>.

[I-D.ietf-core-groupcomm-bis]
Dijk, E., Wang, C., and M. Tiloca, "Group Communication for the Constrained Application Protocol (CoAP)", Work in Progress, Internet-Draft, draft-ietf-core-groupcomm-bis-08, 11 January 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-core-groupcomm-bis-08>.

[I-D.ietf-core-observe-multicast-notifications]
Tiloca, M., Höglund, R., Amsüss, C., and F. Palombini, "Observe Notifications as CoAP Multicast Responses", Work

in Progress, Internet-Draft, draft-ietf-core-observe-multicast-notifications-06, 26 April 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-core-observe-multicast-notifications-06>.

[I-D.ietf-core-oscore-edhoc]
Palombini, F., Tiloca, M., Höglund, R., Hristozov, S., and G. Selander, "Using EDHOC with CoAP and OSCORE", Work in Progress, Internet-Draft, draft-ietf-core-oscore-edhoc-07, 13 March 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-core-oscore-edhoc-07>.

[I-D.ietf-core-transport-indication]
Amsüss, C., "CoAP Protocol Indication", Work in Progress, Internet-Draft, draft-ietf-core-transport-indication-02, 13 March 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-core-transport-indication-02>.

[I-D.ietf-lake-edhoc]
Selander, G., Mattsson, J. P., and F. Palombini, "Ephemeral Diffie-Hellman Over COSE (EDHOC)", Work in Progress, Internet-Draft, draft-ietf-lake-edhoc-20, 7 July 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-lake-edhoc-20>.

[I-D.tiloca-core-groupcomm-proxy]
Tiloca, M. and E. Dijk, "Proxy Operations for CoAP Group Communication", Work in Progress, Internet-Draft, draft-tiloca-core-groupcomm-proxy-08, 28 February 2023, <https://datatracker.ietf.org/doc/html/draft-tiloca-core-groupcomm-proxy-08>.

[I-D.tiloca-schc-8824-update]
Tiloca, M., Toutain, L., and I. Martinez, "Clarifications and Updates on using Static Context Header Compression (SCHC) for the Constrained Application Protocol (CoAP)", Work in Progress, Internet-Draft, draft-tiloca-schc-8824-update-00, 3 April 2023, <https://datatracker.ietf.org/doc/html/draft-tiloca-schc-8824-update-00>.

[LwM2M-Core]
Open Mobile Alliance, "Lightweight Machine to Machine Technical Specification - Core, Approved Version 1.2, OMA-TS-LightweightM2M_Core-V1_2-20201110-A", November 2020, <http://www.openmobilealliance.org/release/LightweightM2M/V1_2-20201110-A/OMA-TS-LightweightM2M_Core-V1_2-20201110-A.pdf>.

[LwM2M-Gateway]
Open Mobile Alliance, "Lightweight Machine to Machine Gateway Technical Specification - Approved Version 1.1, OMA-TS-LWM2M_Gateway-V1_1-20210518-A", May 2021, <https://www.openmobilealliance.org/release/LwM2M_Gateway/V1_1-20210518-A/OMA-TS-LWM2M_Gateway-V1_1-20210518-A.pdf>.

[LwM2M-Transport]
Open Mobile Alliance, "Lightweight Machine to Machine Technical Specification - Transport Bindings, Approved Version 1.2, OMA-TS-LightweightM2M_Transport-V1_2-20201110-A", November 2020, <http://www.openmobilealliance.org/release/LightweightM2M/V1_2-20201110-A/OMA-TS-LightweightM2M_Transport-V1_2-20201110-A.pdf>.

[RFC7030] Pritikin, M., Ed., Yee, P., Ed., and D. Harkins, Ed., "Enrollment over Secure Transport", RFC 7030, DOI 10.17487/RFC7030, October 2013, <https://www.rfc-editor.org/rfc/rfc7030>.

[RFC7641] Hartke, K., "Observing Resources in the Constrained Application Protocol (CoAP)", RFC 7641, DOI 10.17487/RFC7641, September 2015, <https://www.rfc-editor.org/rfc/rfc7641>.

[RFC8742] Bormann, C., "Concise Binary Object Representation (CBOR) Sequences", RFC 8742, DOI 10.17487/RFC8742, February 2020, <https://www.rfc-editor.org/rfc/rfc8742>.

[RFC9200] Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "Authentication and Authorization for Constrained Environments Using the OAuth 2.0 Framework (ACE-OAuth)", RFC 9200, DOI 10.17487/RFC9200, August 2022, <https://www.rfc-editor.org/rfc/rfc9200>.

Appendix A. Examples

This section provides a number of examples where the approach defined in this document is used to protect message exchanges.

The presented examples build on the example shown in Appendix A.1 of [RFC8613], and illustrate an origin client requesting the alarm status from an origin server, through a forward-proxy.

The abbreviations "REQ" and "RESP" are used to denote a request message and a response message, respectively.

A.1. Example 1

In the example shown in Figure 1, message exchanges are protected with OSCORE over the following legs.

* End-to-end, between the client and the server, using the OSCORE Security Context CTX_C_S. The client uses the OSCORE Sender ID 0x5f when using OSCORE with the server.

* Between the client and the proxy, using the OSCORE Security Context CTX_C_P. The client uses the OSCORE Sender ID 0x20 when using OSCORE with the proxy.

```
Client  Proxy  Server
  |       |       |
Encrypt   |       |
REQ with  |       |
CTX_C_S   |       |
  |       |       |
Encrypt   |       |
REQ with  |       |
CTX_C_P   |       |
  |       |       |
  +------>|       |     Code: 0.02 (POST)
  | POST  |       |    Token: 0x8c
  |       |       |   OSCORE: [kid: 0x20, Partial IV: 31]
  |       |       |     0xff
  |       |       |  Payload: {Code: 0.02,
  |       |       |            OSCORE: [kid: 0x5f, Partial IV: 42],
  |       |       |            Uri-Host: example.com,
  |       |       |            Proxy-Scheme: coap,
  |       |       |            0xff,
  |       |       |            {Code: 0.01,
  |       |       |             Uri-Path: "alarm_status"
  |       |       |            } // Encrypted with CTX_C_S
  |       |       |           } // Encrypted with CTX_C_P
  |       |       |
  |     Decrypt   |
  |     REQ with  |
  |     CTX_C_P   |
  |       |       |
  |       +------>|     Code: 0.02 (POST)
  |       | POST  |    Token: 0x7b
  |       |       |   OSCORE: [kid: 0x5f, Partial IV: 42]
  |       |       |     0xff
  |       |       |  Payload: {
  |       |       |            Code: 0.01,
  |       |       |            Uri-Path: "alarm_status"
```

```
     |        |        |           } // Encrypted with CTX_C_S
     |        |        |
     |        |     Decrypt
     |        |     REQ with
     |        |     CTX_C_S
     |        |        |
     |        |     Encrypt
     |        |     RESP with
     |        |     CTX_C_S
     |        |        |
     |        |<------+     Code: 2.04 (Changed)
     |        |  2.04 |    Token: 0x7b
     |        |        |   OSCORE: -
     |        |        |     0xff
     |        |        | Payload: {Code: 2.05,
     |        |        |           0xff,
     |        |        |           "0"
     |        |        |           } // Encrypted with CTX_C_S
     |        |        |
     |     Encrypt     |
     |     RESP with   |
     |     CTX_C_P     |
     |        |        |
     |<------+         |     Code: 2.04 (Changed)
     |  2.04  |        |    Token: 0x8c
     |        |        |   OSCORE: -
     |        |        |     0xff
     |        |        | Payload: {Code: 2.04,
     |        |        |           OSCORE: -,
     |        |        |           0xff,
     |        |        |           {Code: 2.05,
     |        |        |            0xff,
     |        |        |            "0"
     |        |        |           } // Encrypted with CTX_C_S
     |        |        |           } // Encrypted with CTX_C_P
     |        |        |
  Decrypt     |        |
  RESP with   |        |
  CTX_C_P     |        |
    |         |        |
  Decrypt     |        |
  RESP with   |        |
  CTX_C_S     |        |
    |         |        |
```

Square brackets [...] indicate content of compressed COSE object.
Curly brackets { ... } indicate encrypted data.

Figure 1: Use of OSCORE between Client-Server and Client-Proxy

A.2. Example 2

In the example shown in Figure 2, message exchanges are protected with OSCORE over the following legs.

* End-to-end between the client and the server, using the OSCORE Security Context CTX_C_S. The client uses the OSCORE Sender ID 0x5f when using OSCORE with the server.

* Between the proxy and the server, using the OSCORE Security Context CTX_P_S. The proxy uses the OSCORE Sender ID 0xd4 when using OSCORE with the server.

```
Client  Proxy  Server
  |       |       |
Encrypt   |       |
REQ with  |       |
CTX_C_S   |       |
  |       |       |
  +------>|       |         Code: 0.02 (POST)
  | POST  |       |        Token: 0x8c
  |       |       |     Uri-Host: example.com
  |       |       | Proxy-Scheme: coap
  |       |       |      OSCORE: [kid: 0x5f, Partial IV: 42]
  |       |       |        0xff
  |       |       |      Payload: {Code: 0.01,
  |       |       |                Uri-Path: "alarm_status"
  |       |       |               } // Encrypted with CTX_C_S
  |       |       |
  |     Encrypt   |
  |     REQ with  |
  |     CTX_P_S   |
  |       |       |
  |       +------>|         Code: 0.02 (POST)
  |       | POST  |        Token: 0x7b
  |       |       |      OSCORE: [kid: 0xd4, Partial IV: 31]
  |       |       |        0xff
  |       |       |      Payload: {Code: 0.02,
  |       |       |                OSCORE: [kid: 0x5f, Partial IV: 42],
  |       |       |                0xff,
  |       |       |                {Code: 0.01,
  |       |       |                 Uri-Path: "alarm_status"
  |       |       |                } // Encrypted with CTX_C_S
  |       |       |               } // Encrypted with CTX_P_S
  |       |       |
  |       |     Decrypt
```

```
   |        |     REQ with
   |        |     CTX_P_S
   |        |       |
   |        |     Decrypt
   |        |     REQ with
   |        |     CTX_C_S
   |        |       |
   |        |     Encrypt
   |        |     RESP with
   |        |     CTX_C_S
   |        |       |
   |        |     Encrypt
   |        |     RESP with
   |        |     CTX_P_S
   |        |       |
   |        |<------+         Code: 2.04 (Changed)
   |        |  2.04 |        Token: 0x7b
   |        |       |       OSCORE: -
   |        |       |         0xff
   |        |       |      Payload: {Code: 2.04,
   |        |       |                OSCORE: -,
   |        |       |                0xff,
   |        |       |                {Code: 2.05,
   |        |       |                 0xff,
   |        |       |                 "0"
   |        |       |                } // Encrypted with CTX_C_S
   |        |       |               } // Encrypted with CTX_P_S
   |        |       |
   |        |       |
   |      Decrypt   |
   |      RESP with |
   |      CTX_P_S   |
   |        |       |
   |<------+        |         Code: 2.04 (Changed)
   |  2.04 |        |        Token: 0x8c
   |       |        |       OSCORE: -
   |       |        |               0xff
   |       |        |      Payload: {Code: 2.05,
   |       |        |                0xff,
   |       |        |                "0"
   |       |        |               } // Encrypted with CTX_C_S
   |       |        |
 Decrypt   |        |
 RESP with |        |
 CTX_C_S   |        |
   |       |        |
```

Square brackets [...] indicate content of compressed COSE object.
Curly brackets { ... } indicate encrypted data.

Figure 2: Use of OSCORE between Client-Server and Proxy-Server

A.3. Example 3

In the example shown in Figure 3, message exchanges are protected with OSCORE over the following legs.

* End-to-end between the client and the server, using the OSCORE Security Context CTX_C_S. The client uses the OSCORE Sender ID 0x5f when using OSCORE with the server.

* Between the client and the proxy, using the OSCORE Security Context CTX_C_P. The client uses the OSCORE Sender ID 0x20 when using OSCORE with the proxy.

* Between the proxy and the server, using the OSCORE Security Context CTX_P_S. The proxy uses the OSCORE Sender ID 0xd4 when using OSCORE with the server.

```
Client  Proxy  Server
  |       |       |
Encrypt   |       |
REQ with  |       |
CTX_C_S   |       |
  |       |       |
Encrypt   |       |
REQ with  |       |
CTX_C_P   |       |
  |       |       |
  +------>|       |    Code: 0.02 (POST)
  | POST  |       |   Token: 0x8c
  |       |       |  OSCORE: [kid: 0x20, Partial IV: 31]
  |       |       |    0xff
  |       |       | Payload: {Code: 0.02,
  |       |       |           OSCORE: [kid: 0x5f, Partial IV: 42],
  |       |       |           Uri-Host: example.com,
  |       |       |           Proxy-Scheme: coap,
  |       |       |           0xff,
  |       |       |           {Code: 0.01,
  |       |       |            Uri-Path: "alarm_status"
  |       |       |           } // Encrypted with CTX_C_S
  |       |       |          } // Encrypted with CTX_C_P
  |       |       |
  |     Decrypt   |
  |     REQ with  |
  |     CTX_C_P   |
  |       |       |
  |     Encrypt   |
```

```
|      REQ with  |
|      CTX_P_S   |
|        |       |
|        +------>|    Code: 0.02 (POST)
|        | POST  |   Token: 0x7b
|        |       |  OSCORE: [kid: 0xd4, Partial IV: 31]
|        |       |    0xff
|        |       | Payload: {Code: 0.02,
|        |       |           OSCORE: [kid: 0x5f, Partial IV: 42],
|        |       |           0xff,
|        |       |           {Code: 0.01,
|        |       |            Uri-Path: "alarm_status"
|        |       |           } // Encrypted with CTX_C_S
|        |       |          } // Encrypted with CTX_P_S
|        |       |
|        |     Decrypt
|        |     REQ with
|        |     CTX_P_S
|        |       |
|        |     Decrypt
|        |     REQ with
|        |     CTX_C_S
|        |       |
|        |     Encrypt
|        |     RESP with
|        |     CTX_C_S
|        |       |
|        |     Encrypt
|        |     RESP with
|        |     CTX_P_S
|        |       |
|        |<------+    Code: 2.04 (Changed)
|        |  2.04 |   Token: 0x7b
|        |       |  OSCORE: -
|        |       |    0xff
|        |       | Payload: {Code: 2.04,
|        |       |           OSCORE: -,
|        |       |           0xff,
|        |       |           {Code: 2.05,
|        |       |            0xff,
|        |       |            "0"
|        |       |           } // Encrypted with CTX_C_S
|        |       |          } // Encrypted with CTX_P_S
|        |       |
|      Decrypt   |
|      RESP with |
|      CTX_P_S   |
|        |       |
```

```
     |     Encrypt   |
     |     ERSP with |
     |     CTX_C_P   |
     |        |      |
     <-------+          Code: 2.04 (Changed)
     |  2.04  |        Token: 0x8c
     |        |      OSCORE: -
     |        |        0xff
     |        |     Payload: {Code: 2.04,
     |        |               OSCORE: -,
     |        |               0xff,
     |        |               {Code: 2.05,
     |        |                0xff,
     |        |                "0"
     |        |               } // Encrypted with CTX_C_S
     |        |              } // Encrypted with CTX_C_P
     |        |
  Decrypt     |
  RESP with   |
  CTX_C_P     |
     |        |
  Decrypt     |
  RESP with   |
  CTX_C_S     |
     |        |
```

Square brackets [...] indicate content of compressed COSE object.
Curly brackets { ... } indicate encrypted data.

Figure 3: Use of OSCORE between Client-Server, Client-Proxy and Proxy-Server

A.4. Example 4

In the example shown in Figure 4, message exchanges are protected over the following legs.

* End-to-end, between the client and the server, using the OSCORE Security Context CTX_C_S. The client uses the OSCORE Sender ID 0x5f when using OSCORE with the server.

* Between the client and the proxy, using the OSCORE Security Context CTX_C_P. The client uses the OSCORE Sender ID 0x20 when using OSCORE with the proxy.

The example also shows how the client establishes an OSCORE Security Context CTX_C_P with the proxy and CTX_C_S with the server, by using the key establishment protocol EDHOC [I-D.ietf-lake-edhoc].

```
   Client  Proxy  Server
     |       |       |
     +------>|       |         Code: 0.02 (POST)
     | POST  |       |        Token: 0xf3
     |       |       |     Uri-Path: ".well-known"
     |       |       |     Uri-Path: "edhoc"
     |       |       |         0xff
     |       |       |      Payload: (true, EDHOC message_1)
     |       |       |
     |<------+       |         Code: 2.04 (Changed)
     |  2.04 |       |        Token: 0xf3
     |       |       |         0xff
     |       |       |      Payload: EDHOC message_2
     |       |       |
   Establish |       |
   CTX_C_P   |       |
     |       |       |
     +------>|       |         Code: 0.02 (POST)
     | POST  |       |        Token: 0x82
     |       |       |     Uri-Path: ".well-known"
     |       |       |     Uri-Path: edhoc
     |       |       |         0xff
     |       |       |      Payload: (C_R, EDHOC message_3)
     |       |       |
     |     Establish |
     |     CTX_C_P   |
     |       |       |
     |<------+       |
     |  ACK  |       |
     |       |       |
   Encrypt   |       |
   REQ with  |       |
   CTX_C_P   |       |
     |       |       |
     +------>|       |         Code: 0.02 (POST)
     | POST  |       |        Token: 0xbe
     |       |       |      OSCORE: [kid: 0x20, Partial IV: 00]
     |       |       |         0xff
     |       |       |      Payload: {Code: 0.02,
     |       |       |                Uri-Host: "example.com",
     |       |       |                Uri-Path: ".well-known",
     |       |       |                Uri-Path: "edhoc",
     |       |       |                Proxy-Scheme: "coap",
     |       |       |                0xff,
     |       |       |                (true, EDHOC message_1)
     |       |       |               } // Encrypted with CTX_C_P
     |       |       |
     |     Decrypt   |
```

```
      |      REQ with   |
      |      CTX_C_P    |
      |        |        |
      |        +------>|         Code: 0.02 (POST)
      |        | POST   |        Token: 0xa5
      |        |        |    Uri-Path: ".well-known"
      |        |        |    Uri-Path: "edhoc"
      |        |        |         0xff
      |        |        |     Payload: (true, EDHOC message_1)
      |        |        |
      |        |<------+         Code: 2.04 (Changed)
      |        |  2.04  |        Token: 0xa5
      |        |        |         0xff
      |        |        |     Payload: EDHOC message_2
      |        |        |
      |      Encrypt    |
      |      RESP with  |
      |      CTX_C_P    |
      |        |        |
      |<------+         |         Code: 2.04 (Changed)
      |  2.04  |        |        Token: 0xbe
      |        |        |       OSCORE: -
      |        |        |         0xff
      |        |        |     Payload: {Code: 2.04,
      |        |        |                0xff,
      |        |        |                EDHOC message_2
      |        |        |               } // Encrypted with CTX_C_P
      |        |        |
   Establish   |        |
   CTX_C_S     |        |
      |        |        |
   Encrypt     |        |
   REQ with    |        |
   CTX_C_P     |        |
      |        |        |
      +------>|         |         Code: 0.02 (POST)
      | POST   |        |        Token: 0xb9
      |        |        |       OSCORE: [kid: 0x20, Partial IV: 01]
      |        |        |         0xff
      |        |        |     Payload: {Code: 0.02,
      |        |        |                Uri-Host: "example.com",
      |        |        |                Uri-Path: ".well-known",
      |        |        |                Uri-Path: "edhoc",
      |        |        |                Proxy-Scheme: "coap",
      |        |        |                0xff,
      |        |        |                (C_R, EDHOC message_3)
      |        |        |               } // Encrypted with CTX_C_P
      |        |        |
```

```
      |     Decrypt   |
      |     REQ with  |
      |     CTX_C_P   |
      |       |       |
      |       +------>|         Code: 0.02 (POST)
      |       | POST  |        Token: 0xdd
      |       |       |     Uri-Path: ".well-known"
      |       |       |     Uri-Path: "edhoc"
      |       |       |         0xff
      |       |       |      Payload: (C_R, EDHOC message_3)
      |       |       |
      |       |     Establish
      |       |     CTX_C_S
      |       |       |
      |       |<------+
      |       |  ACK  |
      |       |       |
      |<------+       |
      |  ACK  |       |
      |       |       |
   Encrypt    |       |
   REQ with   |       |
   CTX_C_S    |       |
      |       |       |
   Encrypt    |       |
   REQ with   |       |
   CTX_C_P    |       |
      |       |       |
      +------>|           Code: 0.02 (POST)
      | POST  |          Token: 0x8c
      |       |        OSCORE: [kid: 0x20, Partial IV: 02]
      |       |          0xff
      |       |       Payload: {Code: 0.02,
      |       |                 OSCORE: [kid: 0x5f, Partial IV: 00],
      |       |                 Uri-Host: "example.com",
      |       |                 Proxy-Scheme: "coap",
      |       |                 0xff,
      |       |                 {Code: 0.01,
      |       |                  Uri-Path: "alarm_status"
      |       |                 } // Encrypted with CTX_C_S
      |       |                } // Encrypted with CTX_C_P
      |       |
      |     Decrypt   |
      |     REQ with  |
      |     CTX_C_P   |
      |       |       |
      |       +------>|    Code: 0.02 (POST)
      |       | POST  |   Token: 0x7b
```

```
     |        |        |  OSCORE: [kid: 0x5f, Partial IV: 00]
     |        |        |    0xff
     |        |        | Payload: {Code: 0.01,
     |        |        |           Uri-Path: "alarm_status"
     |        |        |          } // Encrypted with CTX_C_S
     |        |        |
     |        |     Decrypt
     |        |     REQ with
     |        |     CTX_C_S
     |        |        |
     |        |     Encrypt
     |        |     RESP with
     |        |     CTX_C_S
     |        |        |
     |        |<------+    Code: 2.04 (Changed)
     |        |  2.04 |   Token: 0x7b
     |        |        | OSCORE: -
     |        |        |    0xff
     |        |        | Payload: {Code: 2.05,
     |        |        |           0xff,
     |        |        |           "0"
     |        |        |          } // Encrypted with CTX_C_S
     |        |        |
     |     Encrypt     |
     |     RESP with   |
     |     CTX_C_P     |
     |        |        |
     |<------+         |    Code: 2.04 (Changed)
     |  2.04  |        |   Token: 0x8c
     |        |        | OSCORE: -
     |        |        |    0xff
     |        |        | Payload: {Code: 2.04,
     |        |        |           OSCORE: -,
     |        |        |           0xff,
     |        |        |           {Code: 2.05,
     |        |        |            0xff,
     |        |        |            "0"
     |        |        |           } // Encrypted with CTX_C_S
     |        |        |          } // Encrypted with CTX_C_P
     |        |        |
  Decrypt     |        |
  RESP with   |        |
  CTX_C_P     |        |
     |        |        |
  Decrypt     |        |
  RESP with   |        |
  CTX_C_S     |        |
     |        |        |
```

Square brackets [...] indicate content of compressed COSE object.
Curly brackets { ... } indicate encrypted data.
Round brackets (...) indicate a CBOR sequence [RFC 8742].

Figure 4: Use of OSCORE between Client-Server and Proxy-Server, with OSCORE Security Contexts established through EDHOC

A.5. Example 5

In the example shown in Figure 5, message exchanges are protected over the following legs.

* End-to-end, between the client and the server. The client uses the OSCORE Sender ID 0x5f when using OSCORE with the server.

* Between the client and the proxy. The client uses the OSCORE Sender ID 0x20 when using OSCORE with the proxy.

The example also shows how the client establishes an OSCORE Security Context CTX_C_P with the proxy and CTX_C_S with the server, by using the key establishment protocol EDHOC [I-D.ietf-lake-edhoc].

In particular, the client relies on the EDHOC + OSCORE request defined in [I-D.ietf-core-oscore-edhoc] and denoted as COMB_REQ, in order to transport the last EDHOC message_3 and the first OSCORE-protected application CoAP request combined together.

```
 Client  Proxy  Server
   |       |       |
   +------>|       |         Code: 0.02 (POST)
   | POST  |       |        Token: 0xf3
   |       |       |     Uri-Path: ".well-known"
   |       |       |     Uri-Path: "edhoc"
   |       |       |         0xff
   |       |       |      Payload: (true, EDHOC message_1)
   |       |       |
   |<------+       |         Code: 2.04 (Changed)
   |  2.04 |       |        Token: 0xf3
   |       |       |         0xff
   |       |       |      Payload: EDHOC message_2
   |       |       |
 Establish |       |
 CTX_C_P   |       |
   |       |       |
 Encrypt   |       |
 REQ with  |       |
 CTX_C_P   |       |
   |       |       |
```

```
 Prepare    |        |
 COMB_REQ   |        |
 for P      |        |
 from REQ   |        |
   |        |        |
   +------>|         |         Code: 0.02 (POST)
   | POST   |        |        Token: 0x82
   |        |        |      OSCORE: [kid: 0x20, Partial IV: 00]
   |        |        |       EDHOC: -
   |        |        |        0xff
   |        |        |     Payload: EDHOC message_3, // Intended for P
   |        |        |              {Code: 0.02,
   |        |        |               Uri-Host: "example.com",
   |        |        |               Uri-Path: ".well-known",
   |        |        |               Uri-Path: "edhoc",
   |        |        |               Proxy-Scheme: "coap",
   |        |        |               0xff,
   |        |        |               (true, EDHOC message_1)
   |        |        |              } // Encrypted with CTX_C_P
   |        |        |
   |      Establish  |
   |      CTX_C_P    |
   |        |        |
   |      Rebuild    |
   |      REQ from   |
   |      COMB_REQ   |
   |        |        |
   |      Decrypt    |
   |      REQ with   |
   |      CTX_C_P    |
   |        |        |
   |        +------>|         Code: 0.02 (POST)
   |        | POST   |        Token: 0xa5
   |        |        |    Uri-Path: ".well-known"
   |        |        |    Uri-Path: "edhoc"
   |        |        |        0xff
   |        |        |     Payload: (true, EDHOC message_1)
   |        |        |
   |        |<------+         Code: 2.04 (Changed)
   |        |  2.04  |        Token: 0xa5
   |        |        |        0xff
   |        |        |     Payload: EDHOC message_2
   |        |        |
   |      Encrypt    |
   |      RESP with  |
   |      CTX_C_P    |
   |        |        |
   |<------+         |    Code: 2.04 (Changed)
```

```
   |  2.04 |        |   Token: 0x82
   |       |        |  OSCORE: -
   |       |        |    0xff
   |       |        | Payload: {Code: 2.04,
   |       |        |           0xff,
   |       |        |           EDHOC message_2
   |       |        |          } // Encrypted with CTX_C_P
   |       |        |
 Decrypt   |        |
 RESP with |        |
 CTX_C_P   |        |
   |       |        |
 Establish |        |
 CTX_C_S   |        |
   |       |        |
 Encrypt   |        |
 REQ with  |        |
 CTX_C_S   |        |
   |       |        |
 Prepare   |        |
 COMB_REQ  |        |
 for S     |        |
 from REQ  |        |
   |       |        |
 Encrypt   |        |
 REQ with  |        |
 CTX_C_P   |        |
   |       |        |
   +------>|        |         Code: 0.02 (POST)
   | POST  |        |        Token: 0x83
   |       |        |       OSCORE: [kid: 0x20, Partial IV: 01]
   |       |        |         0xff
   |       |        |      Payload: {Code: 0.02,
   |       |        |                Uri-Host: "example.com",
   |       |        |                OSCORE: [kid: 0x5f, Partial IV: 00],
   |       |        |                EDHOC: -,
   |       |        |                Proxy-Scheme: "coap",
   |       |        |                0xff,
   |       |        |                EDHOC message_3 // Intended for S
   |       |        |                {
   |       |        |                 Code: 0.01,
   |       |        |                 Uri-Path:"alarm_status"
   |       |        |                } // Encrypted with CTX_C_S
   |       |        |               } // Encrypted with CTX_C_P
   |       |        |
   |     Decrypt    |
   |     REQ with   |
   |     CTX_C_P    |
```

```
   |        |        |
   |        +------>|         Code: 0.02 (POST)
   |        | POST  |        Token: 0xa6
   |        |        |       OSCORE: [kid: 0x5f, Partial IV: 00]
   |        |        |        EDHOC: -
   |        |        |         0xff
   |        |        |      Payload: EDHOC message_3 // Intended for S
   |        |        |               {
   |        |        |                Code: 0.01,
   |        |        |                Uri-Path: "alarm_status"
   |        |        |               } // Encrypted with CTX_C_S
   |        |        |
   |        |     Establish
   |        |     CTX_C_S
   |        |        |
   |        |     Rebuild
   |        |     REQ from
   |        |     COMB_REQ
   |        |        |
   |        |     Decrypt
   |        |     REQ with
   |        |     CTX_C_S
   |        |        |
   |        |     Encrypt
   |        |     RESP with
   |        |     CTX_C_S
   |        |        |
   |        |<------+    Code: 2.04 (Changed)
   |        |  2.04 |   Token: 0xa6
   |        |        |  OSCORE: -
   |        |        |    0xff
   |        |        | Payload: {Code: 2.05,
   |        |        |           0xff,
   |        |        |           "0"
   |        |        |          } // Encrypted with CTX_C_S
   |        |        |
   |     Encrypt     |
   |     RESP with   |
   |     CTX_C_P     |
   |        |        |
   |<------+         |    Code: 2.04 (Changed)
   |  2.04 |         |   Token: 0x83
   |        |        |  OSCORE: -
   |        |        |    0xff
   |        |        | Payload: {Code: 2.04,
   |        |        |           OSCORE: -,
   |        |        |           0xff,
   |        |        |           {Code: 2.05,
```

```
   |        |        |            0xff,
   |        |        |            "0"
   |        |        |           } // Encrypted with CTX_C_S
   |        |        |          } // Encrypted with CTX_C_P
   |        |        |
 Decrypt    |        |
 RESP with  |        |
 CTX_C_P    |        |
   |        |        |
 Decrypt    |        |
 RESP with  |        |
 CTX_C_S    |        |
   |        |        |
```

Square brackets [...] indicate content of compressed COSE object.
Curly brackets { ... } indicate encrypted data.
Round brackets (...) indicate a CBOR sequence [RFC 8742].

Figure 5: Use of OSCORE between Client-Server and Proxy-Server, with OSCORE Security Contexts established through EDHOC using the EDHOC + OSCORE request

Appendix B. State Diagram of the Incoming Request Processing

Figure 6 overviews the processing of an incoming request, as specified in Section 3.3. The dotted boxes indicate ending states where the processing terminates.

```
 Original    +-----------------------------------------------+
 incoming -->|        Are there proxy-related options?       |<--------+
 request     +-----------------------------------------------+         |
              |                                             |          |
             YES                                           NO          |
              |                                             |          |
              v                                             v          |
 +-------------+     +---------+    .......... +------------------+   |
 | Is there a  | YES | Am I a  | NO : Return : | Is there an      |   |
 | Proxy-Uri   |---->| forward |--->: 5.05   : | OSCORE Option?   |   |
 | Option?     |     | proxy?  |    :........: +------------------+   |
 +-------------+     +---------+                ^  |        |          |
    |                 ^       |     ..........  |  NO      YES         |
    NO                |      YES    : Return :  |  |        |          |
    |                 |       |     : 4.01   :  |  |        v          |
    |                 |       |     :........:  |  |  +------------+  |
    |                 |       |       ^         |  |  | Are there  |  |
    |                 |       |       |         |  |  | Uri-Path   |  |
    |                YES      |       NO        |  |  | Options?   |  |
    v                 |       v       |         |  |  +------------+  |
```

```
  +--------------------+ +---------------+      |  |   |          |    |
  | Is there the       | | Is forwarding |      |  |   |          |    |
  | Proxy-Scheme       | | this request  |      |  |   YES        NO   |
  | option with the    | | an authorized |      |  |   |          |    |
  | Uri-Host/Uri-Port  | | operation?    |      |  |   v          |    |
  | options?           | +---------------+      |  | .........    |    |
  +--------------------+  ^           |          |  | : Return :   |    |
    |                     |          YES         |  | : 4.00   :   |    |
    NO                    |           |          |  | :........:   |    |
    |                     |           |          |  |              v    |
    v                     |           v          |  |       +---------+ |
  +--------------+        |  .................   |  |       | Decrypt | |
  | There are    |        |  : Consume the   :   |  |       +---------+ |
  | Uri-Path     |        |  : proxy-related :   |  |         |         |
  | Options      |        |  : options and   :   |  |         |         |
  | without      |        |  : forward the   :   |  |         v         |
  | Proxy-Scheme |        |  : request       :   |  |  +----------+     |
  +--------------+        |  :...............:   |  |  | Success? |-YES -+
    |                     |                      |  |  +----------+
    |                     |                      |  |       |
    |                     |                      |  |      NO
    |                     |                      |  |       |
    |                     |                      |  |       v
    |                     |                      |  | .................
    |                     |                      |  | : OSCORE error :
    |                     |                      |  | : handling     :
    |                     |                      |  | :...............:
    |                     |                      |  |
    |                     |                      |  v
    |                     |                      | +--------------+
    |                     |                      | | Is there an  |
    |                     |                      | | application? |
    |                     |                      | +--------------+
    |                     |                      |    |        |
    |                     |                      |   YES       NO
    |                     |                      |    |        |
    |                     |                      |    |        v
    |                     |                      |    |      ..........
    |                     |                      |    |      : Return :
    |                     |                      |    |      : 4.00   :
    |                     YES                    |    |      :........:
    |                     |                      |    v
    v                     |                      |  ...................
  +------------------------+                     |  : Deliver the     :
  | Am I a reverse proxy   |                     |  : request to the  :
  | using the indicated    |--NO-----------------+  : application     :
  | resource for proxying? |                        :.................:
  +------------------------+
```

Figure 6: Processing of an incoming request.

Acknowledgments

The authors sincerely thank Christian Amsüss, Peter Blomqvist, David Navarro and Göran Selander 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 H2020 project SIFIS-Home (Grant agreement 952652).

Authors' Addresses

Marco Tiloca
RISE AB
Isafjordsgatan 22
SE-16440 Kista
Sweden
Email: marco.tiloca@ri.se

Rikard Höglund
RISE AB
Isafjordsgatan 22
SE-16440 Kista
Sweden
Email: rikard.hoglund@ri.se