

CoRE Working Group
Internet-Draft
Intended status: Standards Track
Expires: 8 September 2022

C. Amsüss
M. Tiloca
RISE AB
7 March 2022

Cacheable OSCORE
draft-amsuess-core-cachable-oscore-04

Abstract

Group communication with the Constrained Application Protocol (CoAP) can be secured end-to-end using Group Object Security for Constrained RESTful Environments (Group OSCORE), also across untrusted intermediary proxies. However, this sidesteps the proxies' abilities to cache responses from the origin server(s). This specification restores cacheability of protected responses at proxies, by introducing consensus requests which any client in a group can send to one server or multiple servers in the same group.

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/>.

Source for this draft and an issue tracker can be found at <https://gitlab.com/chrysn/core-cachable-oscore/>.

Status of This Memo

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

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

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

This Internet-Draft will expire on 8 September 2022.

Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	3
1.1. Use cases	4
1.2. Terminology	4
2. OSCORE processing without source authentication	5
3. Deterministic Requests	7
3.1. Deterministic Unprotected Request	7
3.2. Design Considerations	8
3.3. Request-Hash	9
3.4. Use of Deterministic Requests	10
3.4.1. Pre-Conditions	10
3.4.2. Client Processing of Deterministic Request	11
3.4.3. Server Processing of Deterministic Request	13
3.4.4. Response to a Deterministic Request	15
3.4.5. Deterministic Requests to Multiple Servers	16
4. Obtaining Information about the Deterministic Client	17
5. Security Considerations	18
6. IANA Considerations	18
6.1. CoAP Option Numbers Registry	18
6.2. OSCORE Security Context Parameters Registry	19
7. References	20
7.1. Normative References	20
7.2. Informative References	21
Appendix A. Change log	23
Appendix B. Padding	24
B.1. Definition of the Padding Option	24
B.2. Using and processing the Padding option	25
Appendix C. Simple Cacheability using Ticket Requests	25
Appendix D. Application for More Efficient End-to-End Protected Multicast Notifications	27
Appendix E. Open questions	27
Appendix F. Unsorted further ideas	28
Acknowledgments	28

Authors' Addresses	28
--------------------	----

1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] supports also group communication, for instance over UDP and IP multicast [I-D.ietf-core-groupcomm-bis]. In a group communication environment, exchanged messages can be secured end-to-end by using Group Object Security for Constrained RESTful Environments (Group OSCORE) [I-D.ietf-core-oscore-groupcomm].

Requests and responses protected with the group mode of Group OSCORE can be read by all group members, i.e., not only by the intended recipient(s), thus achieving group-level confidentiality.

This allows a trusted intermediary proxy which is also a member of the OSCORE group to populate its cache with responses from origin servers. Later on, the proxy can possibly reply to a request in the group with a response from its cache, if recognized as an eligible server by the client.

However, an untrusted proxy which is not member of the OSCORE group only sees protected responses as opaque, uncacheable ciphertext. In particular, different clients in the group that originate a same plain CoAP request would send different protected requests, as a result of their Group OSCORE processing. Such protected requests cannot yield a cache hit at the proxy, which makes the whole caching of protected responses pointless.

This document addresses this complication and enables cacheability of protected responses, also for proxies that are not members of the OSCORE group and are unaware of OSCORE in general. To this end, it builds on the concept of "consensus request" initially considered in [I-D.ietf-core-observe-multicast-notifications], and defines "deterministic request" as a convenient incarnation of such concept.

All clients wishing to send a particular GET or FETCH request are able to deterministically compute the same protected request, using a variation on the pairwise mode of Group OSCORE. It follows that cache hits become possible at the proxy, which can thus serve clients in the group from its cache. Like in [I-D.ietf-core-observe-multicast-notifications], this requires that clients and servers are already members of a suitable OSCORE group.

Cacheability of protected responses is useful also in applications where several clients wish to retrieve the same object from a single server. Some security properties of OSCORE are dispensed with to gain other desirable properties.

In order to clearly handle the protocol's security properties, and to broaden applicability to group situations outside the deterministic case, the technical implementation is split in two halves:

- * maintaining request-response bindings in absence of request source authentication, and
- * building and processing of deterministic requests (which have no source authentication, and thus require the former).

1.1. Use cases

When firmware updates are delivered using CoAP, many similar devices fetch large representations at the same time. Collecting them at a proxy not only keeps the traffic low, but also lets the clients ride single file to hide their numbers[SW-EPIV] and identities.

When relying on intermediaries to fan out the delivery of multicast data protected end-to-end as in [I-D.ietf-core-observe-multicast-notifications], deterministic requests allow for a more efficient setup, by reducing the amount of message exchanges and enabling early population of cache entries (see Appendix D).

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.

Readers are expected to be familiar with terms and concepts of CoAP [RFC7252] and its method FETCH [RFC8132], group communication for CoAP [I-D.ietf-core-groupcomm-bis], COSE [I-D.ietf-cose-rfc8152bis-struct][I-D.ietf-cose-rfc8152bis-algs], OSCORE [RFC8613], and Group OSCORE [I-D.ietf-core-oscore-groupcomm].

This document introduces the following new terms.

- * **Consensus Request:** a CoAP request that multiple clients use to repeatedly access a particular resource. In this document, it exclusively refers to requests protected with Group OSCORE to a resource hosted at one or more servers in the OSCORE group.

A Consensus Request has all the properties relevant to caching, but its transport dependent properties (e.g., Token or Message ID) are not defined. Thus, different requests on the wire can be said to "be the same Consensus Request" even if they have different Tokens or source addresses.

The Consensus Request is the reference for request-response binding. In general, a client processing a response to a consensus request did not generate (and thus sign) the consensus request. The client not only needs to decrypt the Consensus Request to understand a response to it (for example to tell which path was requested), it also needs to verify that this is the only Consensus Request that could elicit this response.

- * **Deterministic Client:** a fictitious member of an OSCORE group, having no Sender Sequence Number, no asymmetric key pair, and no Recipient Context.

The Group Manager sets up the Deterministic Client, and assigns it a unique Sender ID as for other group members. Furthermore, the Deterministic Client has only the minimum common set of privileges shared by all group members.

- * **Deterministic Request:** a Consensus Request generated by the Deterministic Client. The use of Deterministic Requests is defined in Section 3.
- * **Ticket Request:** a Consensus Request generated by the server itself.

This term is not used in the main document, but is useful in comparison with other applications of consensus requests that are generated in a different way than as a Deterministic Request. The prototypical Ticket Request is the Phantom Request defined in [I-D.ietf-core-observe-multicast-notifications].

In Appendix C, the term is used to bridge the gap to that draft.

2. OSCORE processing without source authentication

The request-response binding of OSCORE is achieved by the `request_kid` / `request_piv` items (and, in group OSCORE, `request_kid_context`) present in the response's AAD. Its security depends on the server obtaining source authentication for the request: Without, a malicious group member could alter a request to the server (without altering the `request_` details above), and the client would still accept the response as if it were a response to its request.

Source authentication is thus a precondition to secure use of OSCORE. However, it is hard to provide when:

- * Requests are built exclusively using shared key material (as in a Deterministic Client).
- * Requests are sent without source authentication, or where the source authentication is not checked. (This was part of [I-D.ietf-core-oscore-groupcomm] in revisions before -12).

This document does not [yet?] give full guidance on how to restore request-response binding for the general case, but currently only offers suggestions:

- * The response can contain the full request. An option that allows doing that was presented in [I-D.bormann-core-responses].
- * The response can contain a cryptographic hash of the full request. This is used in Section 3.3.
- * The above details can be transported in a Class E option (encrypted) or a a Class I option (unencrypted but part of the AAD). The latter has the advantage that it can be removed in transit and reconstructed at the receiver.
- * Alternatively, the agreed-on request data can be placed in a different position in the AAD, or be part of the security context derivation. In the latter case, care needs to be taken to never initialize a security context twice with the same input, as that would lead to nonce reuse.

[Suggestion for any OSCORE v2: avoid request details in the request's AAD as individual elements. Rather than having 'request_kid', 'request_piv' and (in Group OSCORE) 'request_kid_context' as separate fields, they can better be something more pluggable. This would avoid the need to make up an option before processing, and would allow just plugging in the hash or request in there replacing the request_ items.]

Additional care has to be taken that details not expressed in the request itself (like the security context from which it is assumed to have originated) are captured.

Processing of requests without source authentication has to be done assuming only the minimal possible privilege of the requester [which currently described as the authorization of the Deterministic Client, and may be moved up here in later versions of this document]. If a response is built to such a request that contains data more sensitive

than that (which might be justified if the response is protected for an authorized group member in pairwise response mode), special consideration for any side channels like response size or timing is required.

3. Deterministic Requests

This section defines a method for clients starting from a same plain CoAP request to independently arrive at a same Deterministic Request protected with Group OSCORE.

3.1. Deterministic Unprotected Request

Clients build the unprotected Deterministic Request in a way which is as much reproducible as possible. This document does not set out full guidelines for minimizing the variation, but considered starting points are:

- * Set the inner Observe option to 0 even if no observation is intended (and hence no outer Observe is set). Thus, both observing and non-observing requests can be aggregated into a single request, that is upstreamed as an observation at the latest when any observing request reaches a caching proxy.

In this case, following a Deterministic Request that includes only an inner Observe option, servers include an inner Observe option (but no outer Observe option) in a successful response sent as reply. Also, when receiving a response to such a Deterministic Request previously sent, clients have to silently ignore the inner Observe option in that response.

- * Avoid setting the ETag option in requests on a whim. Only set it when there was a recent response with that ETag. When obtaining later blocks, do not send the known-stale ETag.
- * In block-wise transfer, maximally sized large inner blocks (szx=6) should be selected. This serves not only to align the clients on consistent cache entries, but also helps amortize the additional data transferred in the per-message signatures.

Outer block-wise transfer can then be used if these messages exceed a hop's efficiently usable MTU size.

(If BERT [RFC8323] is usable with OSCORE, its use is fine as well; in that case, the server picks a consistent block size for all clients anyway).

- * The Padding option defined in Appendix B can be used to limit an adversary's ability to deduce the content and the target resource of Deterministic Requests from their length. In particular, all Deterministic Requests of the same class (ideally, all requests to a particular server) can be padded to reach the same total length, that should be agreed on among all users of the same OSCORE Security Context.
- * Clients should not send any inner Echo options [RFC9175] in deterministic requests.

This limits the use of the Echo option in combination with deterministic requests to unprotected (outer) options, and thus is limited to testing the reachability of the client. This is not practically limiting, as the use as an inner option would be to prove freshness, which is something deterministic requests simply cannot provide anyway.

These only serve to ensure that cache entries are utilized; failure to follow them has no more severe consequences than decreasing the utility and effectiveness of a cache.

3.2. Design Considerations

The hard part is arriving at a consensus pair (key, nonce) to be used with the AEAD cipher for encrypting the Deterministic Request, while also avoiding reuse of the same (key, nonce) pair across different requests.

Diversity can conceptually be enforced by applying a cryptographic hash function to the complete input of the encryption operation over the plain CoAP request (i.e., the AAD and the plaintext of the COSE object), and then using the result as source of uniqueness. Any non-malleable cryptographically secure hash of sufficient length to make collisions sufficiently unlikely is suitable for this purpose.

A tempting possibility is to use a fixed (group) key, and use the hash as a deterministic AEAD nonce for each Deterministic Request through the Partial IV component (see Section 5.2 of [RFC8613]). However, the 40 bit available for the Partial IV are by far insufficient to ensure that the deterministic nonce is not reused across different Deterministic Requests. Even if the full deterministic AEAD nonce could be set, the sizes used by common algorithms would still be too small.

As a consequence, the proposed method takes the opposite approach, by considering a fixed deterministic AEAD nonce, while generating a different deterministic encryption key for each Deterministic

Request. That is, the hash computed over the plain CoAP request is taken as input to the key generation. As an advantage, this approach does not require to transport the computed hash in the OSCORE option.

[Note: This has a further positive side effect arising with version -11 of Group OSCORE. That is, since the full encoded OSCORE option is part of the AAD, it avoids a circular dependency from feeding the AAD into the hash computation, which in turn needs crude workarounds like building the full AAD twice, or zeroing out the hash-to-be.]

3.3. Request-Hash

In order to transport the hash of the plain CoAP request, a new CoAP option is defined, which MUST be supported by clients and servers that support Deterministic Requests.

The option is called Request-Hash. As summarized in Figure 1, the Request-Hash option is elective, safe to forward, part of the cache key and repeatable.

No.	C	U	N	R	Name	Format	Length	Default
TBD1				x	Request-Hash	opaque	any	(none)

Figure 1: Request-Hash Option

The Request-Hash option is identical in all its properties to the Request-Tag option defined in [RFC9175], with the following exceptions:

- * It may be arbitrarily long.

Implementations can limit its length to that of the longest output of the supported hash functions.

- * It may be present in responses (TBD: Does this affect any other properties?).

A response's Request-Hash is, as a matter of default value, equal to the request's. The response is only valid if its Request-Hash is equal to the matching request's.

Servers (including proxies) thus generally SHOULD NOT need to send the Request-Hash option explicitly in responses, especially as a matter of bandwidth efficiency.

A reason (and, currently, the only known) to actually send a Request-Hash in a response are non-traditional responses as described in [I-D.bormann-core-responses], which in terms of that document are non-matching to the request (and thus easily usable); the request hash in the response allows populating caches (see below) and decryption of the response in deterministic request contexts. In the context of non-traditional responses, a matching request's Request-Hash can be inferred from its value in the response.

- * A proxy MAY use any fresh cached response from the selected server to respond to a request with the same Request-Hash; this may save it some memory.

A proxy can add or remove the request's Request-Tag value to / from a response.

- * When used with a Deterministic Request, this option is created at message protection time by the sender, and used before message unprotection by the recipient. Therefore, in this use case, it is treated as Class U for OSCORE [RFC8613] in requests. In the same application, for responses, it is treated as Class I, and often elided from sending (but reconstructed at the receiver). Other uses of this option can put it into different classes for the OSCORE processing.

This option achieves request-response binding described in Section 2.

3.4. Use of Deterministic Requests

This section defines how a Deterministic Request is built on the client side and then processed on the server side.

3.4.1. Pre-Conditions

The use of Deterministic Requests in an OSCORE group requires that the interested group members are aware of the Deterministic Client in the group. In particular, they need to know:

- * The Sender ID of the Deterministic Client, to be used as 'kid' parameter for the Deterministic Requests. This allows all group members to compute the Sender Key of the Deterministic Client.

The Sender ID of the Deterministic Client is immutable throughout the lifetime of the OSCORE group. That is, it is not relinquished and it does not change upon changes of the group keying material following a group rekeying performed by the Group Manager.

- * The hash algorithm to use for computing the hash of a plain CoAP request, when producing the associated Deterministic Request.

Group members have to obtain this information from the Group Manager. A group member can do that, for instance, when obtaining the group keying material upon joining the OSCORE group, or later on as an active member by sending a request to a dedicated resource at the Group Manager.

The joining process based on the Group Manager defined in [I-D.ietf-ace-key-groupcomm-oscore] can be easily extended to support the provisioning of information about the Deterministic Client. Such an extension is defined in Section 4 of this document.

3.4.2. Client Processing of Deterministic Request

In order to build a Deterministic Request, the client protects the plain CoAP request using the pairwise mode of Group OSCORE (see Section 9 of [I-D.ietf-core-oscore-groupcomm]), with the following alterations.

1. When preparing the OSCORE option, the `external_aad` and the AEAD nonce:

- * The used Sender ID is the Deterministic Client's Sender ID.

- * The used Partial IV is 0.

When preparing the `external_aad`, the element `'sender_public_key'` in the `aad_array` takes the empty CBOR byte string.

2. The client uses the hash function indicated for the Deterministic Client, and computes a hash `H` over the following input: the Sender Key of the Deterministic Client, concatenated with the `external_aad` from step 1, concatenated with the COSE plaintext.

Note that the payload of the plain CoAP request (if any) is not self-delimiting, and thus hash functions are limited to non-malleable ones.

3. The client derives the deterministic Pairwise Sender Key `K` as defined in Section 2.3.1 of [I-D.ietf-core-oscore-groupcomm], with the following differences:

- * The Sender Key of the Deterministic Client is used as first argument of the HKDF.

- * The hash H from step 2 is used as second argument of the HKDF, i.e., as a pseudo IKM-Sender computable by all the group members.

Note that an actual IKM-Sender cannot be obtained, since there is no authentication credential (and public key included therein) associated with the Deterministic Client, to be used as Sender Authentication Credential and for computing an actual Diffie-Hellman Shared Secret.

- * The Sender ID of the Deterministic Client is used as value for the 'id' element of the 'info' parameter used as third argument of the HKDF.
4. The client includes a Request-Hash option in the request to protect, with value set to the hash H from Step 2.
 5. The client MAY include an inner Observe option set to 0 to be protected with OSCORE, even if no observation is intended (see Section 3.1).
 6. The client protects the request using the pairwise mode of Group OSCORE as defined in Section 9.3 of [I-D.ietf-core-oscore-groupcomm], using the AEAD nonce from step 1, the deterministic Pairwise Sender Key K from step 3 as AEAD encryption key, and the finalized AAD.
 7. The client MUST NOT include an unprotected (outer) Observe option if no observation is intended, even in case an inner Observe option was included at step 5 above.
 8. The client sets FETCH as the outer code of the protected request to make it usable for a proxy's cache, even if no observation is requested [RFC7641].

The result is the Deterministic Request to be sent.

Since the encryption key K is derived using material from the whole plain CoAP request, this (key, nonce) pair is only used for this very message, which is deterministically encrypted unless there is a hash collision between two Deterministic Requests.

The deterministic encryption requires the used AEAD algorithm to be deterministic in itself. This is the case for all the AEAD algorithms currently registered with COSE in [COSE.Algorithms]. For future algorithms, a flag in the COSE registry is to be added.

Note that, while the process defined above is based on the pairwise mode of Group OSCORE, no information about the server takes part to the key derivation or is included in the AAD. This is intentional, since it allows for sending a deterministic request to multiple servers at once (see Section 3.4.5). On the other hand, it requires later checks at the client when verifying a response to a Deterministic Request (see Section 3.4.4).

3.4.3. Server Processing of Deterministic Request

Upon receiving a Deterministic Request, a server performs the following actions.

A server that does not support Deterministic Requests would not be able to create the necessary Recipient Context, and thus will fail decrypting the request.

1. If not already available, the server retrieves the information about the Deterministic Client from the Group Manager, and derives the Sender Key of the Deterministic Client.
2. The server actually recognizes the request to be a Deterministic Request, due to the presence of the Request-Hash option and to the 'kid' parameter of the OSCORE option set to the Sender ID of the Deterministic Client.

If the 'kid' parameter of the OSCORE option specifies a different Sender ID than the one of the Deterministic Client, the server MUST NOT take the following steps, and instead processes the request as per Section 9.4 of [I-D.ietf-core-oscore-groupcomm].

3. The server retrieves the hash H from the Request-Hash option.
4. The server derives a Recipient Context for processing the Deterministic Request. In particular:
 - * The Recipient ID is the Sender ID of the Deterministic Client.
 - * The Recipient Key is derived as the key K in step 3 of Section 3.4.2, with the hash H retrieved at the previous step.
5. The server verifies the request using the pairwise mode of Group OSCORE, as defined in Section 9.4 of [I-D.ietf-core-oscore-groupcomm], using the Recipient Context from step 4, with the difference that the server does not perform replay checks against a Replay Window (see below).

In case of successful verification, the server MUST also perform the following actions, before possibly delivering the request to the application.

- * Starting from the recovered plain CoAP request, the server MUST recompute the same hash that the client computed at step 2 of Section 3.4.2.

If the recomputed hash value differs from the value retrieved from the Request-Hash option at step 3, the server MUST treat the request as invalid and MAY reply with an unprotected 4.00 (Bad Request) error response. The server MAY set an Outer Max-Age option with value zero. The diagnostic payload MAY contain the string "Decryption failed".

This prevents an attacker that guessed a valid authentication tag for a given Request-Hash value to poison caches with incorrect responses.

- * The server MUST verify that the unprotected request is safe to be processed in the REST sense, i.e., that it has no side effects. If verification fails, the server MUST discard the message and SHOULD reply with a protected 4.01 (Unauthorized) error response.

Note that some CoAP implementations may not be able to prevent that an application produces side effects from a safe request. This may incur checking whether the particular resource handler is explicitly marked as eligible for processing deterministic requests. An implementation may also have a configured list of requests that are known to be side effect free, or even a pre-built list of valid hashes for all sensible requests for them, and reject any other request.

These checks replace the otherwise present requirement that the server needs to check the Replay Window of the Recipient Context (see step 5 above), which is inapplicable with the Recipient Context derived at step 4 from the value of the Request-Hash option. The reasoning is analogous to the one in [I-D.amsuess-lwig-oscore] to treat the potential replay as answerable, if the handled request is side effect free.

3.4.4. Response to a Deterministic Request

When treating a response to a deterministic request, the Request-Hash option is treated as a Class I option (but usually not sent). This creates the request-response binding ensuring that no mismatched responses can be successfully unprotected (see Section 2). The client MUST reject responses with a Request-Hash not matching the one it sent in the request.

When preparing the response, the server performs the following actions.

1. The server sets a non-zero Max-Age option, thus making the Deterministic Request usable for the proxy cache.
2. The server preliminarily sets the Request-Hash option with the full request hash.
3. If the Deterministic Request included an inner Observe option but not an outer Observe option, the server MUST include an inner Observe option in the response.
4. The server MUST protect the response using the group mode of Group OSCORE, as defined in Section 8.3 of [I-D.ietf-core-oscore-groupcomm]. This is required to ensure that the client can verify source authentication of the response, since the "pairwise" key used for the Deterministic Request is actually shared among all the group members.

Note that the Request-Hash option is treated as Class I here.

5. The server MUST use its own Sender Sequence Number as Partial IV to protect the response, and include it as Partial IV in the OSCORE option of the response. This is required since the server does not perform replay protection on the Deterministic Request (see Section 3.4.4).
6. The server uses 2.05 (Content) as outer code even though it is not necessarily an Observe notification [RFC7641], in order to make the response cacheable.
7. The server SHOULD remove the Request-Hash option from the message before sending as per the general option mechanism of Section 3.3.
8. If the Deterministic Request included an inner Observe option but not an outer Observe option, the server MUST NOT include an outer Observe option in the response.

Upon receiving the response, the client performs the following actions.

1. In case the response includes a 'kid' in the OSCORE option and unless responses from multiple servers are expected (see Section 3.4.5), the client MUST verify it to be exactly the 'kid' of the server to which the Deterministic Request was sent.
2. The client sets the Request-Hash option with the full request hash on the response. If an option of a different value is already present, it rejects the response.
3. The client verifies the response using the group mode of Group OSCORE, as defined in Section 8.4 of [I-D.ietf-core-oscore-groupcomm]. In particular, the client verifies the counter signature in the response, based on the 'kid' of the server it sent the request to. When verifying the response, the Request-Hash option is treated as a Class I option.
4. If the Deterministic Request included an inner Observe option but not an outer Observe option (see Section 3.1), the client MUST silently ignore the inner Observe option in the response and MUST NOT stop processing the response.

[Note: This deviates from Section 4.1.3.5.2 of RFC 8613, but it is limited to a very specific situation, where the client and server both know exactly what happens. This does not affect the use of OSCORE in other situations.]

3.4.5. Deterministic Requests to Multiple Servers

A Deterministic Request can be sent to a CoAP group, e.g., over UDP and IP multicast [I-D.ietf-core-groupcomm-bis], thus targeting multiple servers at once.

To simplify key derivation, such a Deterministic Request is still created in the same way as a one-to-one request and still protected with the pairwise mode of Group OSCORE, as defined in Section 3.4.2.

[Note: If it was protected with the group mode, the request hash would need to be fed into a group key derivation just for this corner case. Furthermore, there would need to be a signature in spite of no authentication credential (and public key included therein) associated with the Deterministic Client.]

When a server receives a request from the Deterministic Client as addressed to a CoAP group, the server proceeds as defined in Section 3.4.3, with the difference that it MUST include its own Sender ID in the response, as 'kid' parameter of the OSCORE option.

Although it is normally optional for the server to include its Sender ID when replying to a request protected in pairwise mode, it is required in this case for allowing the client to retrieve the Recipient Context associated with the server originating the response.

4. Obtaining Information about the Deterministic Client

This section extends the Joining Process defined in [I-D.ietf-ace-key-groupcomm-oscore], and based on the ACE framework for Authentication and Authorization [I-D.ietf-ace-oauth-authz]. Upon joining the OSCORE group, this enables a new group member to obtain from the Group Manager the required information about the Deterministic Client (see Section 3.4.1).

With reference to the 'key' parameter of the Joining Response defined in Section 6.4 of [I-D.ietf-ace-key-groupcomm-oscore], the Group_OSCORE_Input_Material object specified as its value contains also the two additional parameters 'det_senderId' and 'det_hash_alg'. These are defined in Section 6.2 of this document. In particular:

- * The 'det_senderId' parameter, if present, has as value the OSCORE Sender ID assigned to the Deterministic Client by the Group Manager. This parameter MUST be present if the OSCORE group uses deterministic requests as defined in this document. Otherwise, this parameter MUST NOT be present.
- * The 'det_hash_alg' parameter, if present, has as value the hash algorithm to use for computing the hash of a plain CoAP request, when producing the associated Deterministic Request. This parameter takes values from the "Value" column of the "COSE Algorithms" Registry [COSE.Algorithms]. This parameter MUST be present if the OSCORE group uses deterministic requests as defined in this document. Otherwise, this parameter MUST NOT be present.

The same extension above applies also to the 'key' parameter when included in a Key Distribution Response (see Sections 8.1 and 8.2 of [I-D.ietf-ace-key-groupcomm-oscore]) and in a Signature Verification Data Response (see Section 13 of [I-D.ietf-ace-key-groupcomm-oscore]).

5. Security Considerations

The same security considerations from [RFC7252][I-D.ietf-core-groupcomm-bis][RFC8613][I-D.ietf-core-oscore-groupcomm] hold for this document.

Compared to Group OSCORE, deterministic requests dispense with some of OSCORE's security properties by just so much as to make caching possible:

- * Receiving a response to a deterministic request does not mean that the response was generated after the request was sent.

It still contains two freshness statements, though:

- It is more recent than any other response from the same group member that has a smaller sequence number.
- It is more recent than the original creation of the deterministic security context's key material.

- * Request privacy is limited.

An intermediary can determine that two requests from different clients are identical, and associate the different responses generated for them. A server producing responses of varying size to a request can use the Padding option to hide when the response is changing.

- * Source authentication for requests is lost.

Instead, the server must verify that the request (precisely: its handler) is side effect free. The distinct semantics of the CoAP request codes can help the server make that assessment.

[More on the verification of the Deterministic Request]

6. IANA Considerations

This document has the following actions for IANA.

6.1. CoAP Option Numbers Registry

IANA is asked to enter the following option numbers to the "CoAP Option Numbers" registry defined in [RFC7252] within the "CoRE Parameters" registry.

Number	Name	Reference
TBD1	Request-Hash	[[this document]]
TBD2	Padding	[[this document]]

Figure 2: CoAP Option Numbers

[

For the Request-Hash option, the number suggested to IANA is 548.

For the Padding option, the option number is picked to be the highest number in the Experts Review range; the high option number allows it to follow practically all other options, and thus to be set when the final unpadded message length including all options is known. Therefore, the number suggested to IANA is 64988.

Applications that make use of the "Experimental use" range and want to preserve that property are invited to pick the largest suitable experimental number (65532)

Note that unless other high options are used, this means that padding a message adds an overhead of at least 3 bytes, i.e., 1 byte for option delta/length and two more bytes of extended option delta. This is considered acceptable overhead, given that the application has already chosen to prefer the privacy gains of padding over wire transfer length.

]

6.2. OSCORE Security Context Parameters Registry

IANA is asked to register the following entries in the "OSCORE Security Context Parameters" Registry defined in Section 9.4 of [I-D.ietf-ace-oscore-profile].

- * Name: det_senderId
- * CBOR Label: TBD3
- * CBOR Type: bstr
- * Registry: -

- * Description: OSCORE Sender ID assigned to the Deterministic Client of an OSCORE group
- * Reference: [[this document]] (Section 4)
- * Name: det_hash_alg
- * CBOR Label: TBD4
- * CBOR Type: int / tstr
- * Registry: -
- * Description: Hash algorithm to use in an OSCORE group when producing a Deterministic Request
- * Reference: [[this document]] (Section 4)

7. References

7.1. 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-05, 25 October 2021, <<https://datatracker.ietf.org/doc/html/draft-ietf-core-groupcomm-bis-05>>.

[I-D.ietf-core-oscore-groupcomm]

Tiloca, M., Selander, G., Palombini, F., Mattsson, J. P., and J. Park, "Group OSCORE - Secure Group Communication for CoAP", Work in Progress, Internet-Draft, draft-ietf-core-oscore-groupcomm-13, 25 October 2021, <<https://datatracker.ietf.org/doc/html/draft-ietf-core-oscore-groupcomm-13>>.

[I-D.ietf-cose-rfc8152bis-struct]

Schaad, J., "CBOR Object Signing and Encryption (COSE): Structures and Process", Work in Progress, Internet-Draft, draft-ietf-cose-rfc8152bis-struct-15, 1 February 2021, <<https://datatracker.ietf.org/doc/html/draft-ietf-cose-rfc8152bis-struct-15>>.

- [I-D.ietf-cose-rfc8152bis-algs]
Schaad, J., "CBOR Object Signing and Encryption (COSE): Initial Algorithms", Work in Progress, Internet-Draft, draft-ietf-cose-rfc8152bis-algs-12, 24 September 2020, <<https://datatracker.ietf.org/doc/html/draft-ietf-cose-rfc8152bis-algs-12>>.
- [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>>.
- [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>>.
- [COSE.Algorithms]
IANA, "COSE Algorithms", <<https://www.iana.org/assignments/cose/cose.xhtml#algorithms>>.

7.2. Informative References

- [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>>.
- [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>>.

[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-12, 25 October 2021, <<https://datatracker.ietf.org/doc/html/draft-ietf-ace-key-groupcomm-oscore-12>>.

[I-D.amsuess-lwig-oscore]

Amsüss, C., "OSCORE Implementation Guidance", Work in Progress, Internet-Draft, draft-amsuess-lwig-oscore-00, 29 April 2020, <<https://datatracker.ietf.org/doc/html/draft-amsuess-lwig-oscore-00>>.

[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-02, 25 October 2021, <<https://datatracker.ietf.org/doc/html/draft-ietf-core-observe-multicast-notifications-02>>.

[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-46, 8 November 2021, <<https://datatracker.ietf.org/doc/html/draft-ietf-ace-oauth-authz-46>>.

[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://datatracker.ietf.org/doc/html/draft-ietf-ace-oscore-profile-19>>.

[SW-EPIV] Lucas, G., "Star Wars", Lucasfilm Ltd. , 1977.

[I-D.bormann-core-responses]

Bormann, C. and C. Amsüss, "CoAP: Non-traditional response forms", Work in Progress, Internet-Draft, draft-bormann-core-responses-01, 3 February 2022, <<https://datatracker.ietf.org/doc/html/draft-bormann-core-responses-01>>.

[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>>.

Appendix A. Change log

Since -03:

- * Processing steps in case only inner Observe is included.
- * Clarified preserving/eliding the Request-Hash option in responses.
- * Clarified limited use of the Echo option.
- * Clarifications on using the Padding option.

Since -02:

- * Separate parts needed to respond to unauthenticated requests from the remaining deterministic response part. (Currently this is mainly an addition; the document will undergo further refactoring if that split proves helpful).
- * Inner Observe is set unconditionally in deterministic requests.
- * Clarifications around padding and security considerations.

Since -01:

- * Not meddling with `request_kid` any more.

Instead, Request-Hash in responses is treated as Class I, but typically elided.

In requests, this removes the need to compute the `external_aad` twice.

- * Derivation of the hash now uses the `external_aad`, rather than the full AAD. This is good enough because AAD is a function only of the `external_aad`, and the `external_aad` is easier to get your hands on if COSE manages all the rest.
- * The Sender ID of the Deterministic Client is immutable throughout the group lifetime. Hence, no need for any related expiration/creation time and mechanisms to perform its update in the group.

- * Extension to the ACE Group Manager of ace-key-groupcomm-oscore to provide required info about the Deterministic Client to new group members when joining the group.
- * Alignment with changes in core-oscore-groupcomm-12.
- * Editorial improvements.

Since -00:

- * More precise specification of the hashing (guided by first implementations)
- * Focus shifted to deterministic requests (where it should have been in the first place; all the build-up of Token Requests was moved to a motivating appendix)
- * Aligned with draft-tiloca-core-observe-responses-multicast-05 (not submitted at the time of submission)
- * List the security properties lost compared to OSCORE

Appendix B. Padding

As discussed in Section 5, information can be leaked by the length of a response or, in different contexts, of a request.

In order to hide such information and mitigate the impact on privacy, the following Padding option is defined, to allow increasing a message's length without changing its meaning.

The option can be used with any CoAP transport, but is especially useful with OSCORE as that does not provide any padding of its own.

Before choosing to pad a message by using the Padding option, application designers should consider whether they can arrange for common message variants to have the same length by picking a suitable content representation; the canonical example here is expressing "yes" and "no" with "y" and "n", respectively.

B.1. Definition of the Padding Option

As summarized in Figure 3, the Padding option is elective, safe to forward and not part of the cache key; these follow from the usage instructions. The option may be repeated, as that may be the only way to achieve a certain total length for the padded message.

No.	C	U	N	R	Name	Format	Length	Default
TBD2			x	x	Padding	opaque	any	(none)

Figure 3: Padding Option

When used with OSCORE, the Padding option is of Class E, this makes it indistinguishable from other Class E options or the payload to third parties.

B.2. Using and processing the Padding option

When a server produces different responses of different length for a given class of requests but wishes to produce responses of consistent length (typically to hide the variation from anyone but the intended recipient), the server can pick a length that all possible responses can be padded to, and set the Padding option with a suitable all-zero option value in all responses to that class of requests.

Likewise, a client can decide on a class of requests that it pads to consistent length. This has considerably less efficacy and applicability when applied to Deterministic Requests. That is: an external observer can group requests even if they are of the same length; and padding would hinder convergence on a single Consensus Request, thus requiring all users of the same OSCORE Security Context to agree on the same total length in advance.

Any party receiving a Padding option **MUST** ignore it. In particular, a server **MUST NOT** make its choice of padding dependent on any padding present in the request. (An option to coordinate response padding driven by the client is out of scope for this document).

Proxies that see a padding option **MAY** discard it.

Appendix C. Simple Cacheability using Ticket Requests

Building on the concept of Phantom Requests and Informative Responses defined in [I-D.ietf-core-observe-multicast-notifications], basic caching is already possible without building a Deterministic Request.

The approach discussed in this appendix is not provided for application. In fact, it is efficient only when dealing with very large representations and no OSCORE inner Block-Wise mode (which is inefficient for other reasons), or when dealing with observe notifications (which are already well covered in [I-D.ietf-core-observe-multicast-notifications]).

Rather, it is more provided as a "mental exercise" for the authors and interested readers to bridge the gap between this document and [I-D.ietf-core-observe-multicast-notifications].

That is, instead of replying to a client with a regular response, a server can send an Informative Response, defined as a protected 5.03 (Service Unavailable) error message. The payload of the Informative Response contains the Phantom Request, which is a Ticket Request in this document's broader terminology.

Unlike a Deterministic Request, a Phantom Request is protected in Group Mode. Instead of verifying a hash, the client can see from the signature that this was indeed the request the server is answering. The client also verifies that the request URI is identical between the original request and the Ticket Request.

The remaining exchange largely plays out like in [I-D.ietf-core-observe-multicast-notifications]'s "Example with a Proxy and Group OSCORE": The client sends the Phantom Request to the proxy (but, lacking a `tp_info`, without a Listen-To-Multicast-Responses option), which forwards it to the server for lack of the option.

The server then produces a regular response and includes a non-zero Max-Age option as an outer CoAP option. Note that there is no point in including an inner Max-Age option, as the client could not pin it in time.

When a second, different client later asks for the same resource at the same server, its new request uses a different 'kid' and 'Partial IV' than the first client's. Thus, the new request produces a cache miss at the proxy and is forwarded to the server, which responds with the same Ticket Request provided to the first client. After that, when the second client sends the Ticket Request, a cache hit at the proxy will be produced, and the Ticket Request can be served from the proxy's cache.

When multiple proxies are in use, or the response has expired from the proxy's cache, the server receives the Ticket Request multiple times. It is a matter of perspective whether the server treats that as an acceptable replay (given that this whole mechanism only makes sense on requests free of side effects), or whether it is conceptualized as having an internal proxy where the request produces a cache hit.

Appendix D. Application for More Efficient End-to-End Protected Multicast Notifications

[I-D.ietf-core-observe-multicast-notifications] defines how a CoAP server can serve all clients observing a same resource at once, by sending notifications over multicast. The approach supports the possible presence of intermediaries such as proxies, also if Group OSCORE is used to protect notifications end-to-end.

However, comparing the "Example with a Proxy" in Appendix A of [I-D.ietf-core-observe-multicast-notifications] and the "Example with a Proxy and Group OSCORE" in Appendix B of [I-D.ietf-core-observe-multicast-notifications] shows that, when using Group OSCORE, more requests need to hit the server. This is because every client originally protects its Observation request individually, and thus needs a custom response served to obtain the Phantom Request as a Ticket Request.

If the clients send their requests as the same deterministic request, the server can use these requests as Ticket Requests as well. Thus, there is no need for the server to provide a same Phantom Request to each client.

Instead, the server can send a single unprotected Informative Response - very much like in the example without Group OSCORE - hence setting the proxy up and optionally providing also the latest notification along the way.

The proxy can thus be configured by the server following the first request from the clients, after which it has an active observation and a fresh cache entry in time for the second client to arrive.

Appendix E. Open questions

- * Is "deterministic encryption" something worthwhile to consider in COSE?

COSE would probably specify something more elaborate for the KDF (the current KDF round is the pairwise mode's; COSE would probably run through KDF with a KDF context structure).

COSE would give a header parameter name to the Request-Hash (which for the purpose of OSCORE deterministic requests would put back into Request-Hash by extending the option compression function across the two options).

Conceptually, they should align well, and the implementation changes are likely limited to how the KDF is run.

- * An unprotection failure from a mismatched hash will not be part of the ideally constant-time code paths that otherwise lead to AEAD unprotect failures. Is that a problem?

After all, it does tell the attacker that they did succeed in producing a valid MAC (it's just not doing it any good, because this key is only used for deterministic requests and thus also needs to pass the Request-Hash check).

Appendix F. Unsorted further ideas

- * All or none of the deterministic requests should have an inner observe option. Preferably none -- that makes messages shorter, and clients need to ignore that option either way when checking whether a Consensus Request matches their intended request.
- * We could allow clients to elide all other options than Request-Hash, and elide the payload, if it has reason to believe that it can produce a cache hit with the abbreviated request alone.

This may prove troublesome in terms of cache invalidation (the server would have to use short-lived responses to indicate that it does need the full request, or we'd need special handling for error responses, or criteria by which proxies don't even forward these if they don't have a response at hand).

That may be more trouble than it's worth without a strong use case (say, of complex but converging FETCH requests).

Hashes could also be used in truncated form for that.

Acknowledgments

The authors sincerely thank Michael Richardson, Jim Schaad and Goeran Selander for their comments and feedback.

The work on this document has been partly supported by VINNOVA and the Celtic-Next project CRITISEC; and by the H2020 project SIFIS-Home (Grant agreement 952652).

Authors' Addresses

Christian Amsüss
Austria
Email: christian@amsuess.com

Marco Tiloca
RISE AB
Isafjordsgatan 22
SE-16440 Stockholm Kista
Sweden
Email: marco.tiloca@ri.se

CORE Working Group
Internet-Draft
Intended status: Standards Track
Expires: August 13, 2022

G. Fioccola
T. Zhou
Huawei
M. Cociglio
F. Bulgarella
M. Nilo
Telecom Italia
February 9, 2022

Constrained Application Protocol (CoAP) Performance Measurement Option
draft-fz-core-coap-pm-01

Abstract

This document specifies a method for the Performance Measurement of the Constrained Application Protocol (CoAP). A new CoAP option is defined in order to enable network telemetry both end-to-end and on-path.

Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

Status of This Memo

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

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

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

This Internet-Draft will expire on August 13, 2022.

Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	2
2. Performance Measurement methods for CoAP	3
3. CoAP Performance Measurement Option	4
4. Structure of the PM Option	5
5. Application Scenarios	7
6. Security Considerations	8
7. IANA Considerations	9
8. Acknowledgements	9
9. References	9
9.1. Normative References	9
9.2. Informative References	9
Authors' Addresses	10

1. Introduction

[RFC7252] define the CoAP Protocol. In CoAP, reliability is provided by marking a message as Confirmable (CON) with ACKs. A message that does not require reliable transmission can be sent as a Non-confirmable message (NON).

In case of CoAP reliable mode there are Message IDs and ACKs, that could eventually be used to measure Round-Trip Time (RTT) and losses. But it is resource-consuming for constrained nodes since they have to look at Message IDs and take timestamps. These operations are expensive in terms of resources. In case of CoAP unreliable mode, there is no ACK and, consequently, it is not possible to measure RTT and losses.

Thus, there is no easy way to measure the performance metrics in COAP environment to satisfy the low resources of constrained nodes. And it is in any case limited to RTT and end-to-end losses.

A mechanism to measure the performance in CoAP can be useful to verify and meet the operational requirements, but it should be a simple mechanism for network diagnostic to be developed on

constrained nodes requiring just a minimal amount of collaboration from the endpoints.

[I-D.ietf-ippm-explicit-flow-measurements] describes the methodologies for Explicit Flow Measurement (EFM). The EFM techniques employ few marking bits, inside the header of each packet, for loss and delay measurement. These are relevant for encrypted protocols, e.g. QUIC [RFC9000], where there are only few bits available in the non-encrypted header in order to allow passive performance metrics from an on-path observer. These methodologies could potentially be used and extended in CoAP.

[I-D.ietf-ippm-explicit-flow-measurements] defines different combinations of bits because the number of bits in QUIC is limited and different experiments have been done. But all these methods together imply complex algorithms that do not apply well to the CoAP environment.

This document aims to create an easy way to allow performance measurement for CoAP, by defining a new option, called Performance Measurement (PM) CoAP Option. The CoAP performance metrics (e.g. RTT and losses) can be useful for an operator or an enterprise that is managing a constrained, low-power and lossy network.

2. Performance Measurement methods for CoAP

CoAP [RFC7252] defines a number of options that can be included in a message. For this reason, a new option for CoAP, carrying Performance Measurement (PM) bits is the approach followed by this document.

The PM bits that are included in the Option are:

- o sSquare bit (Q bit), based on [RFC8321] and further described in [I-D.ietf-ippm-explicit-flow-measurements];
- o Spin bit (S bit), described in [I-D.ietf-quic-manageability] and included as optional bit in [RFC9000];
- o Loss and Delay event information for further usage.

A requirement to enable PM methods in COAP environment is that the methodologies and the algorithm needs to be kept simple. For this reason, the idea is to re-apply only the S bit and Q bit.

The sSquare bit algorithm is to create square waves of a known length (e.g. 64 packets). Each side of the connection can set the Q bit and

toggle its value every fixed number of packets. The number of packets can be easily recognized and packet loss can be measured.

The Spin bit algorithm is to create a square wave signal on the data flow, using a bit, whose length is equal to RTT. The Spin bit causes one bit to 'spin', generating one edge (a transition from 0 to 1 or from 1 to 0) once per end-to-end RTT. The Spin bit is set by both sides to the same value for as long as one round trip lasts and then it toggles the value.

The synergy between S bit and Q bit is also possible. As described above, the length of the Q bit square waves is fixed (e.g. a predefined number of packets) in this way each endpoint can detect a packet loss if it receives less packets than expected. In addition, it is possible to potentiate the Q bit signal by incorporating RTT information as well. This implies a little modification to the algorithm of the Q bit that could also be used alone:

One packet in a period of the square wave can be selected and set to the opposite value of that period. After one RTT it comes back and another packet is selected and set again to the opposite value of that period. And the process can start again. By measuring the distance between these special packets it is possible to measure the RTT in addition to packet loss. The periods with the special packets have one packet less than expected but this is easy to recognize by both endpoints.

So, with one bit, it can be possible to measure loss and delay. This can be used to reinforce the Spin Bit mechanism or to use only one bit (sQuare bit) in the Option.

The advantages of using the CoAP PM Option are:

- 1) Simplification because it is not needed to read Message IDs, indeed there is a well-defined sQuare wave, and it is not necessary to store timestamps, since the duration of the Spin Bit period is equal to RTT.
- 2) Enabling easy on-path observer (proxy, gateway) metrics.

3. CoAP Performance Measurement Option

Figure 1 shows the property of the CoAP Performance Measurement (PM) Option. The formatting of this table is reported in [RFC7252]. The C, U, N, and R columns indicate the properties Critical, Unsafe, NoCacheKey, and Repeatable as defined in [RFC7252]. None of these properties is marked for the PM options.

Number	C	U	N	R	Name	Format	Length	Default
TBD					PM	uint	1	0

Figure 1: CoAP PM Option Properties

Note that it could be possible to make use of one bit in the option to identify the mode. In this way two patterns can be defined.

4. Structure of the PM Option

The value of the PM option is a 1 byte unsigned integer. This integer value encodes the following fields:

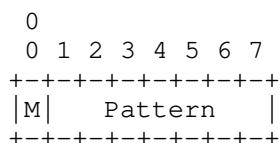


Figure 2: CoAP Performance Measurement Option

Where:

- o M bit can be set to 1 or 0 and it is used to identify whether the Option follows pattern 1 (M bit = 0) or pattern 2 (M bit = 1).
- o Pattern bits can be of two kinds as reported below.

The PM Option can employ two patterns based on the value of the M bit:

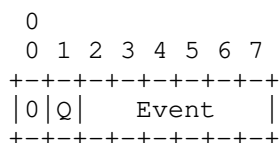


Figure 3: CoAP Performance Measurement Option pattern 1

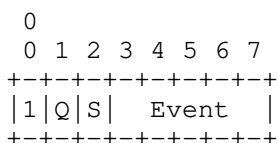


Figure 4: CoAP Performance Measurement Option pattern 2

The COAP Option could be defined with 2 PM bits (S and Q) or defined with a single PM bit (Q bit).

Where:

- o Q bit is used in both pattern 1 and pattern 2. It is described in [I-D.ietf-ippm-explicit-flow-measurements] and enhanced with the method described in Section 2;
- o S bit is used in Option 2. It is also embedded in the QUIC Protocol [RFC9000];
- o Event bits MAY encode additional Loss and Delay information based on well-defined encoding and they can also be used by on-path observers. If these Event bits are all zero, they MUST be ignored on receipt.

As an example the Event bits can be divided into two parts: loss event bits and delay event bits. Based on the average RTT, an end point can define different levels of thresholds and set the delay event bits accordingly. The same applies to loss event bits. In this way an on-path observer becomes aware of the network conditions by simply reading these Event bits.

The on-path observer can read the event signaling bits and could be the Proxy or the Gateway which interconnects disjointed CoAP networks. It MAY communicate with Client and Server to set some parameters such as timeout based on the network performance.

The CoAP PM Options described in this document can be used in both requests and responses. If a CoAP endpoint does not implement the measurement methodologies, it can simply leave the default value (all bits are zero). In this way the other CoAP endpoints become aware that the measurement cannot be executed in that case.

The fixed number of packets to create the Q bit signal is predefined and its value is configured from the beginning for all the CoAP endpoints.

5. Application Scenarios

The main usage of the CoAP PM Options is to do end-to-end measurement between the client and the server but it can also allow split measurements. The on-path measurement is the additional feature. This information can be used to monitor the network in order to check the operational performance and to employ further network optimization.

The intermediaries or on-path observers could be:

Network Functions or Probes that must be able to see deep into application.

Gateway or Proxies that, as specified in [RFC7252], are CoAP endpoints tasked by CoAP clients to perform requests on their behalf.

If the on-path observers are network functions or probes, the CoAP PM Option can be applied end-to-end between client and server. The on-path network probes can read Q bit and S bit and implement the relevant algorithms to measure losses and RTT. Otherwise they can simply read the Event bits and be informed about the performance without implementing any algorithm. The event signaling bits can be sent from the Server (that can do the performance measurement calculation) to the Client, or viceversa.

If the on-path observers are CoAP proxies, the CoAP PM Option can only be applied to the different separate connections between client and server instead of end-to-end. Indeed, CoAP proxies hide the identity of the client and could also apply caching. Thus, on the server side, the data would appear mixed in presence of more than one client, and clients would receive mixed signals in presence of cache entries. But in this case, the measurements can be segmented and done between the Proxies or between a Proxy and the Client or between a Proxy and the Server. The Server can distinguish the source client by using additional flow information such as the IP addresses. It could also be possible to bundle different clients if they are mixed. So, it is worth highlighting that an on-path observer can find useful information both on the proxy-server link and on the client-proxy link:

On the link from a proxy to the server, traffic from different clients would be mixed. In this case, the proxy can still use the PM Option to set S bit and Q bit for the bundle of clients for a specific server. The measurement can be done but it is an information related to a bundle of clients. An alternative can be

to use the Option only for a single client at once in order to avoid to do a grouped measurement.

Conversely, on the link from the client to the proxy, communication may happen with different servers, and in this case it is necessary to check the other fields to understand the server.

In summary, a typical CoAP scenario can be the following:

Devices -- Gateway/Proxy -- Transport Network -- Probe (or Proxy) -- Server/Data Center

If the CoAP PM Option is applied between devices and the server (across intermediaries), the Probe can measure the total RTT by using the Spin bit, indeed it allows RTT measurement for all the intermediate points. But, with sSquare Bit and by applying the methodologies in [RFC8321], it is also possible to do hop-by-hop measurements for loss and delay and segment where possible.

If there is a CoAP proxy, the measurement can be done between the Proxies or between a Proxy and the Client or between a Proxy and the Server. It can be done through Spin bit or by applying [RFC8321] on the sSquare Bit signal.

6. Security Considerations

Security considerations related to CoAP proxying are discussed in [RFC7252].

A CoAP endpoint can use the CoAP PM Options to affect the measures of a network into which it is making requests by maliciously modifying the value of the option. Also, the PM bits may reveal performance information outside the administrative domain. To prevent that, a CoAP proxy that is located at the boundary of an administrative domain MAY be instructed to strip the payload or part of it before forwarding the message.

It is worth highlighting what happens if devices, transport network and server are operated by different administrative domains. Security concerns need to be taken into account, but OSCORE [RFC8613] can be used and the CoAP PM options can be integrity protected end-to-end by OSCORE. Then the operators can put their measurement probes in one or more places to break down the different RTT and loss contributions where it is relevant (e.g., at the ingress/egress of their respective network segments). OSCORE ensures end-to-end integrity protection and would tell the endpoints if someone tampered, but it doesn't mean that the endpoints are not lying to the

observer. However it is possible to assume that for the typical COAP applications it is less likely that the endpoints are attackers while it is more likely that an on-path observer is the attacker.

7. IANA Considerations

IANA is requested to add the following entry to the "CoAP Option Numbers" sub-registry available at <https://www.iana.org/assignments/core-parameters/core-parameters.xhtml#option-numbers>:

Number	Name	Reference
TBD	PM Option	[This draft]

Figure 5: CoAP PM Option Numbers

8. Acknowledgements

The authors would like to thank Christian Amsuess and Thomas Fossati for the precious comments and suggestions.

9. References

9.1. Normative References

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

9.2. Informative References

[I-D.ietf-ippm-explicit-flow-measurements]
Cociglio, M., Ferrieux, A., Fioccola, G., Lubashev, I., Bulgarella, F., Hamchaoui, I., Nilo, M., Sisto, R., and D. Tikhonov, "Explicit Flow Measurements Techniques", draft-ietf-ippm-explicit-flow-measurements-00 (work in progress), October 2021.

[I-D.ietf-quic-manageability]
Kuehlewind, M. and B. Trammell, "Manageability of the QUIC Transport Protocol", draft-ietf-quic-manageability-14 (work in progress), January 2022.

- [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>>.
- [RFC8321] Fioccola, G., Ed., Capello, A., Cociglio, M., Castaldelli, L., Chen, M., Zheng, L., Mirsky, G., and T. Mizrahi, "Alternate-Marking Method for Passive and Hybrid Performance Monitoring", RFC 8321, DOI 10.17487/RFC8321, January 2018, <<https://www.rfc-editor.org/info/rfc8321>>.
- [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>>.
- [RFC9000] Iyengar, J., Ed. and M. Thomson, Ed., "QUIC: A UDP-Based Multiplexed and Secure Transport", RFC 9000, DOI 10.17487/RFC9000, May 2021, <<https://www.rfc-editor.org/info/rfc9000>>.

Authors' Addresses

Giuseppe Fioccola
Huawei
Riesstrasse, 25
Munich 80992
Germany

Email: giuseppe.fioccola@huawei.com

Tianran Zhou
Huawei
156 Beiqing Rd.
Beijing 100095
China

Email: zhoutianran@huawei.com

Mauro Cociglio
Telecom Italia
Via Reiss Romoli, 274
Torino 10148
Italy

Email: mauro.cociglio@telecomitalia.it

Fabio Bulgarella
Telecom Italia
Via Reiss Romoli, 274
Torino 10148
Italy

Email: fabio.bulgarella@guest.telecomitalia.it

Massimo Nilo
Telecom Italia
Via Reiss Romoli, 274
Torino 10148
Italy

Email: massimo.nilo@telecomitalia.it

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

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

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

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

This Internet-Draft will expire on 28 April 2022.

Copyright Notice

Copyright (c) 2021 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	2
1.1. Terminology	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 message 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	$1=2^6$ in bytes	$1=2^8$ in bytes	$1=2^{10}$ 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'.

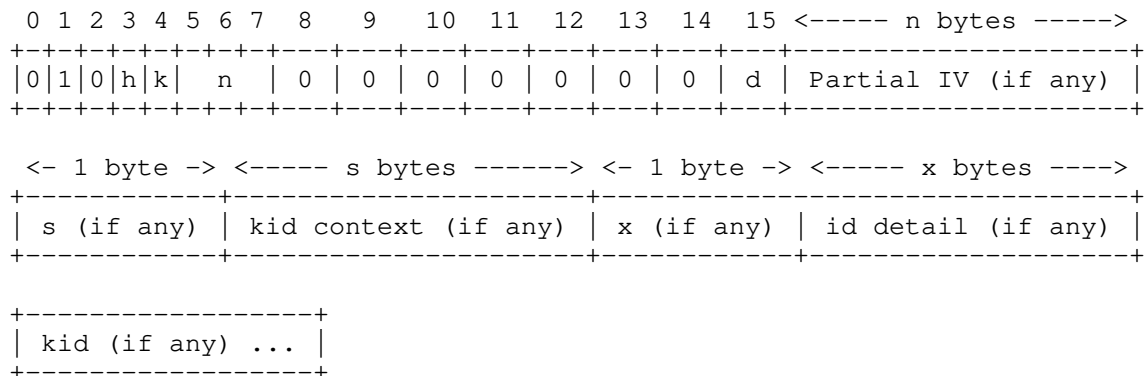


Figure 4: The OSCORE option value, including 'id detail'

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.

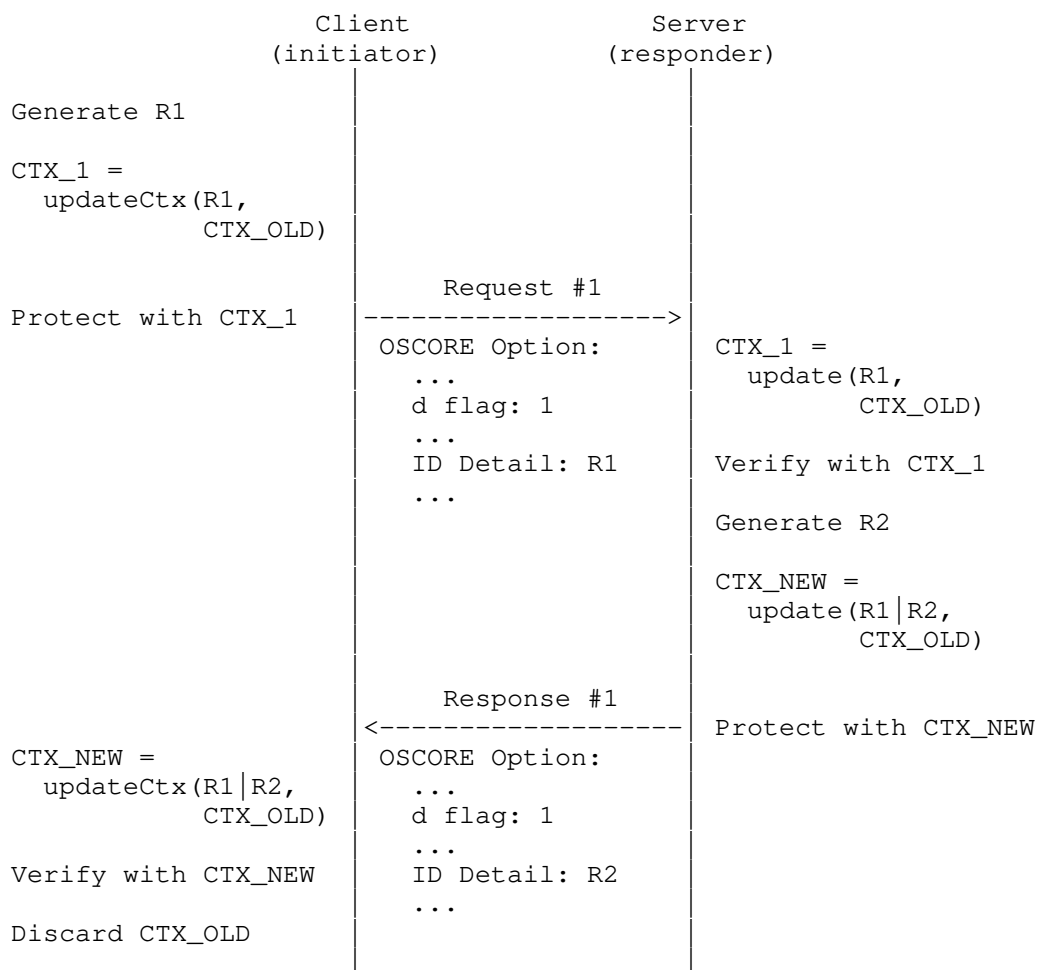
- * The initiator always offers the fresh value R1.
- * The responder always offers the fresh value R2.
- * The responder is always the first one deriving the new OSCORE Security Context CTX_NEW.
- * The initiator is always the first one achieving key confirmation, hence able to safely discard the old 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.




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

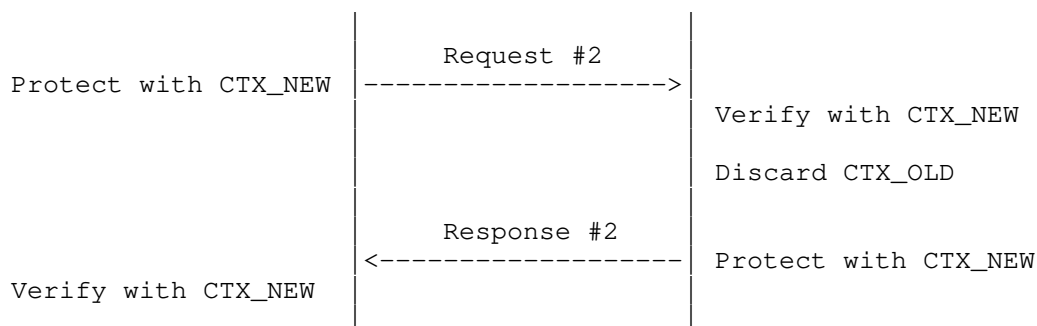


Figure 6: Client-Initiated KUDOS Workflow

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 $N = R1 \parallel R2$ 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 $N = R1 \parallel R2$ 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.

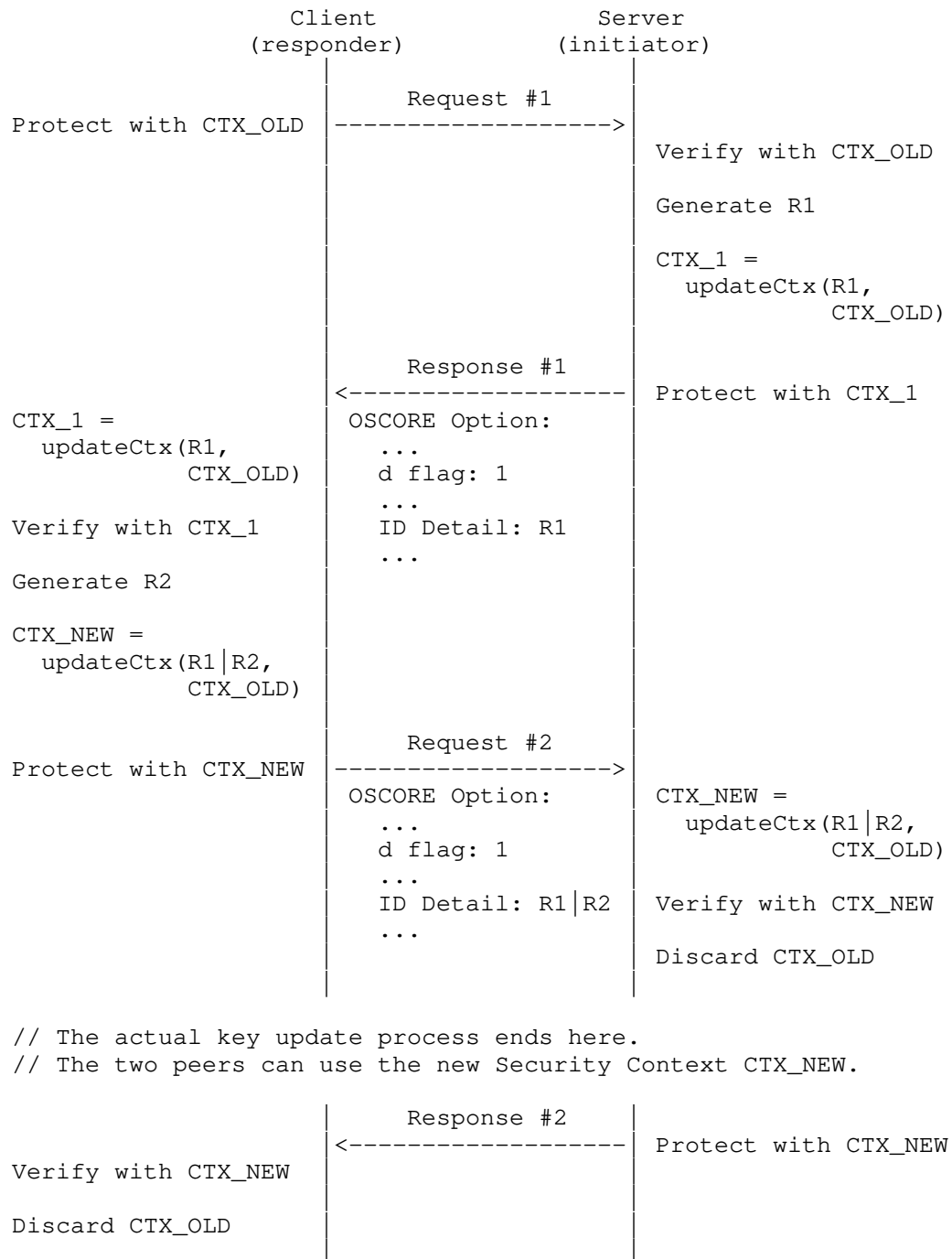


Figure 7: Server-Initiated KUDOS Workflow

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 $N = R1 \parallel R2$ 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 $R1 \parallel R2$ as 'id detail'.

Upon receiving the OSCORE request, the server retrieves the value $R1 \parallel R2$ 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 $R1 \parallel R2$ 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 $N = R1 \parallel R2$ 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: Standards Track
Expires: 8 September 2022

C. Amsüss
T. Fossati
ARM
7 March 2022

The Constrained RESTful Application Language (CoRAL)
draft-ietf-core-coral-05

Abstract

The Constrained RESTful Application Language (CoRAL) defines a data model and interaction model as well as a compact serialization formats for the description of typed connections between resources on the Web ("links"), possible operations on such resources ("forms"), and simple resource metadata.

Note to Readers

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

The issues list for this Internet-Draft can be found at <https://github.com/core-wg/coral/labels/coral>. Companion material for this Internet-Draft can be found at <https://github.com/core-wg/coral>.

Status of This Memo

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

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

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

This Internet-Draft will expire on 8 September 2022.

Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	3
1.1. Data and Interaction Model	4
1.2. Notational Conventions	4
2. Data and Interaction Model	4
2.1. Browsing Context	4
2.2. Documents	5
2.3. Data model	5
2.3.1. Observations	6
2.3.2. Possible variations	7
2.3.3. Examples	7
2.4. Serialization Format	10
2.5. Links	11
2.6. Forms	12
2.7. Form Fields	13
2.8. Navigation	13
2.9. History Traversal	15
2.10. Designing interactions in an Open World	15
3. Binary Format	16
3.1. Data Structure	16
3.1.1. Documents	16
3.1.2. Directives	17
3.1.3. URIs	17
3.1.4. Links	18
3.1.5. Forms	18
3.1.6. Form Fields	19
3.2. Dictionary Compression	19
3.2.1. Media Type Parameter	20
3.3. Export Interface	20
4. Document Semantics	21
4.1. Submitting Documents	21
4.1.1. PUT Requests	21
4.1.2. POST Requests	21
4.2. Returning Documents	22
4.2.1. Success Responses	22
4.2.2. Redirection Responses	22
4.2.3. Error Responses	23
5. Usage Considerations	23

5.1. Specifying CoRAL-based Applications	23
5.1.1. Application Interfaces	23
5.1.2. Resource Identifiers	24
5.1.3. Implementation Limits	24
5.2. Minting Vocabulary	25
5.3. Expressing Registered Link Relation Types	25
5.4. Expressing Simple RDF Statements	26
5.5. Expressing Natural Language Texts	26
5.6. Embedding Representations in CoRAL	27
6. Security Considerations	27
7. IANA Considerations	29
7.1. Media Type "application/coral+cbor"	29
7.2. CoAP Content Formats	30
8. References	30
8.1. Normative References	30
8.2. Informative References	32
Appendix A. Core Vocabulary	36
A.1. Base	36
A.2. Collections	37
A.3. HTTP	37
A.4. CoAP	38
Appendix B. Default Dictionary	39
Appendix C. Mappings to other formats	40
C.1. RDF	40
C.1.1. Example	42
C.2. CoRE Link Format	43
Appendix D. Change Log	46
Acknowledgements	48
Authors' Addresses	48

1. Introduction

The Constrained RESTful Application Language (CoRAL) is a language for the description of typed connections between resources on the Web ("links"), possible operations on such resources ("forms"), and simple resource metadata.

CoRAL is intended for driving automated software agents that navigate a Web application based on a standardized vocabulary of link relation types and operation types. It is designed to be used in conjunction with a Web transfer protocol, such as the Hypertext Transfer Protocol (HTTP) [RFC7230] or the Constrained Application Protocol (CoAP) [RFC7252].

This document defines the CoRAL data model and interaction model as well as a compact serialization format.

1.1. Data and Interaction Model

The data model is similar to the Resource Description Framework (RDF) [W3C.REC-rdf11-concepts-20140225] model, with provisions to enable form based interaction and to express data from Web Linking ([RFC8288]) based models such as [RFC6690]'s Link Format.

The interaction model derives from the processing model of HTML [W3C.REC-html52-20171214] and specifies how an automated software agent can change the application state by navigating between resources following links and performing operations on resources submitting forms.

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

Terms defined in this document appear in *_cursive_* where they are introduced (rendered in plain text as the new term surrounded by underscores).

2. Data and Interaction Model

The Constrained RESTful Application Language (CoRAL) is designed for building Web-based applications [W3C.REC-webarch-20041215] in which automated software agents navigate between resources by following links and perform operations on resources by submitting forms.

2.1. Browsing Context

Borrowing from HTML 5 [W3C.REC-html52-20171214], each such agent maintains a *_browsing context_* in which the representations of Web resources are processed. (In HTML, the browsing context typically corresponds to a tab or window in a Web browser.)

At any time, one representation in a browsing context is designated the *_active_* representation.

2.2. Documents

A resource representation in one of the CoRAL serialization formats is called a CoRAL `_document_`. The URI that was used to retrieve such a document is called the document's `_retrieval context_`. This URI is also considered the base URI for relative URI references in the document.

A CoRAL document consists of a list of zero or more statements that can express links or (in a composition of statements) forms. CoRAL serialization formats may contain additional elements for efficiency or convenience, such as an embedded base URI that takes precedence over the document's base URI, or to concisely represent compound statements (e.g., to express forms).

2.3. Data model

The `_basic CoRAL information model_` is similar to the Resource Description Framework (RDF) [W3C.REC-rdfl1-concepts-20140225] information model: Data is expressed as an (unordered) set of triples (also called statements), consisting of a subject, a predicate and an object. The predicate is always a URI, the subject is a URI or a blank node, and the object is either a URI, a blank node or a literal. All URIs here are limited to the syntax-based normalized form of [RFC3986] Section 6.2.2.

Blank nodes are unnamed entities. Literals are CBOR objects.

These triples form a directed multigraph with the subject and object being source and destination, and the predicate a description on the edge. That graph is equivalent to the data.

To form a set and a graph, we define an equivalence relation: URIs are only equal to URIs and if they are identical byte-wise. A blank node is only equal to itself. A literal is equal to a different literal if its value is equal to the other literal's value in the CBOR generic data model.

Triples are equivalent to each other if their subject, predicate and object are pair-wise equivalent.

The `_CoRAL structured information model_` is a sequence of "passings" of the basic model's edges, starting at a node identifying the document (its retrieval context, typically URI from which it was obtained) where

- * each edge is passed at least one time in total,

- * each edge is passed at most one time after each passing that ends in its start point (with the obvious exception that edges from the retrieval context can be passed once from the start), and
- * between a passing of an edge from A to B and a later passing from B to C, passings can only be along edges that can be reached from B along the graph, until B is the end of a different passing.

For better understanding, think of the structured information model as a sort of tree spanning from the retrieval context, with the oddity that when a node is reached along two different edges (which a normal tree doesn't do), it is up to the builder of the tree whether to describe anything children of the entered node on one parent or on the other parent, on both, or to describe some children at the first and others at a later occasion.

Exceeding the RDF-like model, this represents CoRAL's focus on the discovery of possible future application states over the description of a graph of resources.

2.3.1. Observations

The structured form of a data set is in general not unique: If a node has more than one child, their sequence can be varied. If a node has more than one parent, its children may be expressed on any non-empty set of its parents to obtain a structured data set that expresses the same data set.

In general, arbitrary basic data can not be expressed in a structured data set, because

- * There may not be a tree that covers the directed graph, or the tree's root may not be the retrieval context.
- * There may be multiple edges into a blank node.

In particular, the precise data from one structured information document can only be expressed with the same retrieval context. However, statements can be added to make a data set that is expressible elsewhere (this document defines the carries-information-about relation type leading to the <http://www.iana.org/assignments/relation/carries-information-about> predicate being usable here), and subsets of the data can be taken and expressed.

Forms are not special in the information model, but are merely statements around a blank node. They can be special in serialization formats (which have more efficient notations for them), and are used by the interaction model for special operations.

The structured information model contains more information than the basic information model. [TBD put this into a different context because it's not an observation any more:] Which precise structure is picked is to suit the processing application, typically by profiling the information and its serialization. It is recommended that the information encoded in the structure (including the order) be derived from data available in the general data set, even though the statements that guide the structure are not necessarily encoded in the subset of data that is being structured.

Serializations like the one in Section 3 have even more choices than the structured information model: They can choose to use or not use packed CBOR to compress parts, can spell out URIs in full or use relative references, or can exercise freedoms of the CBOR encoding. Variation there is not to have an influence on the interpretation of a CoRAL document.

2.3.2. Possible variations

- * Each URI is tagged with whether it is intended to be dereferenced or used as an identifier.

2.3.3. Examples

This subsection illustrates the information model and serialization based on an example from [RFC6690]:

```
</sensors>;ct=40;title="Sensor Index",  
</sensors/temp>;rt="temperature-c";if="sensor",  
</sensors/light>;rt="light-lux";if="sensor",  
<http://www.example.com/sensors/t123>;anchor="/sensors/temp";rel="describedby",  
</t>;anchor="/sensors/temp";rel="alternate"
```

Figure 1: Original example at `coap://.../.well-known/core`

After an extraction described in Appendix C.2, this list represents the content of the basic information model representing the above. For the basic model, the table is to be considered unsorted in the first step.

	Subject	Predicate	Object
	coap://.../	rel:hosts	coap://.../sens
ors	coap://.../sensors	linkformat:ct	40
	coap://.../sensors	linkformat:title	"Sensor Index"
	coap://.../	http://www.iana.org/assignments/relation/	coap://.../sens
ors/temp	hosts		
	coap://.../sensors/	linkformat:rt	rt:temperature-
c	temp		
	coap://.../sensors/	linkformat:if	if:sensor
	temp		
	coap://.../sensors/	rel:describedby	http://www.exam
ple.com/sensors/	temp		t123
	coap://.../sensors/	rel:alternate	coap://.../t
	temp		
	coap://.../	http://www.iana.org/assignments/relation/	coap://.../sens
ors/light	hosts		
	coap://.../sensors/	linkformat:rt	rt:light-lux
	light		

+-----+-----+-----+		
coap://.../sensors/	linkformat:if	if:sensor
light		
+-----+-----+-----+		
-----+		

Table 1: Basic (and, through the sequence, Structured) Information Model extracted from there (using CURIes: rel = <http://www.iana.org/assignments/relation/>, linkformat is TBD in the conversion, if, rt is TBD with IANA).

During extraction, some information on item ordering was preserved into the structured data. Note that while the CoRAL structured data preserves some sequence aspects of the Link-Format file (like the order of attributes), others (like the relative order of links from different contexts) are deemed irrelevant and not preserved.

For serialization, the use of the packing described with the conversion results in a binary CBOR file with this CBOR diagnostic notation:

```
[
  [2, simple(10) / item 10 for rel:hosts /, cri"/sensors", [
    [2, 6(2) / item 20 for linkformat:ct /, 40],
    [2, simple(15) / item 15 for linkformat:title /, "Sensor Index"]
  ]],
  [2, simple(10) / item 10 for rel:hosts /, cri"/sensors/temp", [
    [2, 6(1) / item 18 for linkformat:if /, 6(200) / cri"http:TBD...temperature-c
" /],
    [2, 6(-2) / item 19 for linkformat:rt /, 6(250) / cri"http:TBD...sensor" /],
    [2, simple(12) / item 12 for rel:describedby /, cri"http://www.example.com/se
nsors/t123"],
    [2, simple(11) / item 11 for rel:alternate /, cri"/t"]
  ]],
  [2, 10 / item10 for rel:hosts /, cri"/sensors/light", [
    [2, 6(1) / item 18 for linkformat:if /, 6(-201)],
    [2, 6(-2) / item 19 for linkformat:rt /, 6(250)]
  ]]
]
```

Figure 2: Serialized CoRAL file in diagnostic notation.

[TBD: Numbers are made up]

Note that the "temperature-c" interface and "sensor" resource type get code points in the link-format dictionary because they are of reg-name style and thus would be registered as CoRE Parameters, and be included in the packing as well.

2.3.3.1. Literal example

To illustrate non-trivial literals, a link example of [RFC8288] is converted.

(Note that even the conversion scheme hinted at above for [RFC6690] link format makes no claims at being applicable to general purpose web links like the below; this is merely done to demonstrate how literals can be handled. The example even so happens well illustrate that point: General link attributes may only be valid on the target when the link is followed in that direction ("letztes Kapitel" means last chapter), whereas convertible link-format documents use titles that apply to the described resource independent of which link is currently being followed.)

```
Link: </TheBook/chapter2>;
      rel="previous"; title*=UTF-8'de'letztes%20Kapitel,
```

Figure 3: Original link about a book chapter from RFC8288

The model this would be converted to is:

Subject	Predicate	Object
http://.../	rel:previous	http://.../TheBook/ chapter2
http://.../TheBook/ chapter2	linkformat:title	"letztes Kapitel" with language tag "de"

Table 2: Information model extracted from above

In CBOR serialization, this produces:

```
[
  [2, 6(...) / rel:previous /, cri"/TheBook/chapter2", [
    [2, simple(15) / item 15 for linkformat:title /, 38(["de", "letztes Kapitel"]
  )]
  ]]
]
```

Figure 4: Serialization of the RFC8288-based example

2.4. Serialization Format

The primary serialization format is a compact, binary encoding of links and forms in Concise Binary Object Representation (CBOR) [RFC8949]. This format is intended for environments with constraints on power, memory, and processing resources [RFC7228] and shares many similarities with the message format of CoAP: In place of verbose strings, small numeric identifiers are used to encode link relation types and operation types. Uniform Resource Identifiers (URIs) [RFC3986] are expressed as Constrained Resource Identifier (CRI) references [I-D.ietf-core-href] and thus pre-parsed for easy use with CoAP. As a result, link serializations in CoRAL are often much more compact and easier to process than equivalent serializations in CoRE Link Format [RFC6690].

For easy representation of CoRAL documents in text, CBOR diagnostic notation is used. Along with indentation and comments, the notation introduced in [I-D.bormann-cbor-edn-literals] is used to represent CRIs. This format is not expected to be sent over the network.

[To be discussed: For even better readability, the RDF Turtle [W3C.REC-turtle-20140225] format can be used when only the basic information model content is to be conveyed. When used like this, the conversion according to the RDF appendix is implied.]

2.5. Links

Any statement "links" a resource with a second resource or literal, and is thus also referred to as a link.

In [RFC8288] terminology, a CoRAL link's subject is the `_link context_`, the predicate is the `_link relation type_`, and the object is the `_link target_`.

However, a link in CoRAL does not have target attributes. Instead, a link may have a list of zero or more nested elements. These enable both the description of resource metadata and the chaining of links, which is done in [RFC8288] by setting the anchor of one link to the target of another.

A link can be viewed as a statement of the form "{link context} has a {link relation type} resource at {link target}" where the link target may be further described by nested elements.

A link relation type identifies the semantics of a link. In HTML and in [RFC8288], link relation types are typically denoted by an IANA-registered name, such as `stylesheet` or `type`. In CoRAL, all link relation types are, in contrast, denoted by a Universal Resource Identifier (URI) [RFC3986], such as `<http://www.iana.org/assignments/relation/stylesheet>` or `<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>`. This allows for the decentralized creation of new link relation types without the risk of collisions when they come from different organizations or domains of knowledge. URIs can also lead to documentation, schema, and other information about a link relation type. In CoRAL documents, these URIs are only used as identity tokens, though, and are compared with Simple String Comparison as specified in Section 6.2.1 of [RFC3986].

If the link target is a URI and the URI scheme indicates a Web transfer protocol like HTTP or CoAP, an agent can dereference the URI and navigate the browsing context to its target resource; this is called `_following the link_`. An anonymous resource is a resource that is identified by neither a URI nor a literal representation. The agent can still follow the link, but can not dereference it and is limited in its next steps by the outgoing links that are expressed in the current document.

A link can occur as a top-level element in a document or as a nested element within a link. When a link occurs as a top-level element, the link context implicitly is the document's retrieval context. When a link occurs nested within a link, the link context of the nested link is the link target of the enclosing link.

There are no restrictions on the cardinality of links; there can be multiple links to and from a particular target, and multiple links of the same or different types between a given link context and target. However, the nesting nature of the data model constrains the description of resource relations to a tree: Relations between linked resources can only be described by further nesting links.

2.6. Forms

A `_form_` provides instructions to an agent for performing an operation on a resource on the Web. A form has a `_form context_`, an `_operation type_`, a `_request method_`, and a `_submission target_`. Additionally, a form may be accompanied by a list of zero or more `_form fields_`.

In the basic information model, the form is identified with an anonymous node. The form context and operation type are the subject and predicate of an incoming link, respectively; request method and submission target of an outgoing link. Form fields are additional links from that form.

A form can be viewed as an instruction of the form "To perform an {operation type} operation on {form context}, make a {request method} request to {submission target}" where the request may be further described by form fields.

An operation type identifies the semantics of the operation. Operation types are denoted (like link relation types) by a URI.

Form contexts and submission targets are both denoted by a URI. The form context is the resource on which the operation is ultimately performed. To perform the operation, an agent needs to construct a request with the specified method as the request method and the specified submission target as the request URI. Usually, the submission target is the same resource as the form context, but may be a different resource. Constructing and sending the request is called `_submitting the form_`.

A form can occur as a top-level element in a document or as a nested element within a link. When a form occurs as a top-level element, the form context implicitly is the document's retrieval context. When a form occurs nested within a link, the form context is the link target of the enclosing link.

2.7. Form Fields

Form fields can be used to provide more detailed instructions to agents for constructing the request when submitting a form. For example, a form field could instruct an agent to include a certain payload or header field in the request. A payload could, for instance, be described by form fields providing acceptable media types, a reference to schema information, or a number of individual data items that the agents needs to supply. Form fields can be specific to the Web transfer protocol that is used for submitting the form.

A form field is a pair of a `_form field type_` and a `_form field value_`. Additionally, a form field may have a list of zero or more nested elements that further describe the form field value.

A form field type identifies the semantics of the form field. Form field types are predicates and thus URIs. Form field values are URIs, blank nodes or literals.

2.8. Navigation

An agent begins the interaction with an application by performing a GET request on an `_entry point URI_`. The entry point URI is the only URI that the agent is expected to know beforehand. From then on, the agent is expected to make all requests by following links and submitting forms that are provided in the responses resulting from the requests. The entry point URI could be obtained through some discovery process or manual configuration.

If dereferencing the entry point URI yields a CoRAL document (or any other representation that implements the CoRAL data and interaction model), the agent makes this document the active representation in the browsing context and proceeds as follows:

1. The first step for the agent is to decide what to do next, i.e., which type of link to follow or form to submit, based on the link relation types and operation types it understands.

An agent may follow a link without understanding the link relation type, e.g., for the sake of pre-fetching or building a search index. However, an agent **MUST NOT** submit a form without understanding the operation type.

2. The agent then finds the link(s) or form(s) with the respective type in the active representation. This may yield one or more candidates, from which the agent will have to select the most appropriate one. The set of candidates can be empty, for example, when an application state transition is not supported or not allowed.
3. The agent selects one of the candidates based on the metadata associated them (in the form of form fields and nested elements) and their order of appearance in the document. Examples for relevant metadata could include the indication of a media type for the target resource representation, the URI scheme of a target resource, or the request method of an operation.
4. The agent obtains the `_request URI_` from the link target or submission target. Fragment identifiers are not part of the request URI and MUST be separated from the rest of the URI prior to the next step.
5. The agent constructs a new request with the request URI. If the agent is following a link, then the request method MUST be GET. If the agent is submitting a form, then the request method MUST be the one supplied by the form.

The agent SHOULD set HTTP header fields and CoAP request options according to the metadata (e.g., set the HTTP Accept header field or the CoAP Accept option when a media type for the target resource is provided). Depending on the operation type of a form, the agent may also have to include a request payload that matches the specifications of some form fields.

6. The agent sends the request and receives the response.
7. If a fragment identifier was separated from the request URI, the agent selects the fragment indicated by the fragment identifier within the received representation according to the semantics of its media type.
8. The agent updates the browsing context by making the (selected fragment of the) received representation the active representation.
9. Finally, the agent processes the representation according to the semantics of its media type. If the representation is a CoRAL document (or any other representation that implements the CoRAL data and interaction model), the agent again has the choice of what to do next. Go to step 1.

2.9. History Traversal

A browsing context has a `_session history_`, which lists the resource representations that the agent has processed, is processing, or will process.

A session history consists of session history entries. The number of session history entries may be limited and dependent on the agent. An agent with severe constraints on memory size might only have enough memory for the most recent entry.

An entry in the session history consists of a resource representation and the representation's retrieval context. New entries are added to the session history as the agent navigates from resource to resource, discarding entries that are no longer used.

An agent can decide to navigate a browsing context (in addition to following links and submitting forms) by `_traversing the session history_`. For example, when an agent receives a response with a representation that does not contain any further links or forms, it can navigate back to a resource representation it has visited earlier and make that the active representation.

Traversing the history SHOULD take advantage of caches to avoid new requests. An agent may reissue a safe request (e.g., a GET) when it does not have a fresh representation in its cache. An agent MUST NOT reissue an unsafe request (e.g., a PUT or POST) unless it actually intends to perform that operation again.

2.10. Designing interactions in an Open World

CoRAL can be used to build both open world systems ("if something is not said, it may or may not be true") and closed world systems ("if something is not said, it is not true").

In constrained environments (and the web in general), partial representations are often used for efficiency. For example, a device can query another for particular statements using a yet to be defined FETCH version of CoRAL. It is expected that some tools (e.g., server or agent libraries) require the application to be tolerant of unprocessed statements. Furthermore, it can be easier to evolve applications and their packing dictionaries if loss of statements leads to graceful degradation.

Therefore, it is convenient to build applications on open world assumptions. Such applications can only use statements that add possibilities, and none that limit interactions. Any limitations need to be encoded in statements the agent necessarily has to perform an action in the first place, and can then be relaxed in additional statements.

For example, an application built with open-world assumptions can not create a form that allows feeding gremlins, and in an additional statement (e.g., a form field) forbid after midnight. Instead, the application needs to describe a limited-feeding form, which can only be used if any of the attached conditions is met; the condition "before midnight" can then be expressed in an additional statement.

3. Binary Format

This section defines the encoding of documents in the CoRAL binary format.

A document in the binary format is encoded in Concise Binary Object Representation (CBOR) [RFC8949].

The CBOR structure of a document is presented in the Concise Data Definition Language (CDDL) [RFC8610]. All CDDL rules not defined in this document are defined in Appendix D of [RFC8610].

The media type of documents in the binary format is application/coral+cbor.

3.1. Data Structure

The data structure of a document in the binary format is made up of three kinds of elements: links, forms (as short hands for the statements they are constructed of), and (as an extension to the CoRAL data model) directives. Directives provide a way to encode URI references with a common base more efficiently.

3.1.1. Documents

A document in the binary format is encoded as a CBOR array that contains zero or more elements. An element is either a link, a form, or a directive.

```
document = [*element]
element = link / form / directive
```

The elements are processed in the order they appear in the document. Document processors need to maintain an `_environment_` while iterating an array of elements. The environment consists of two variables: the `_current context_` and the `_current base_`. The current context and the current base are both initially set to the document's retrieval context.

3.1.2. Directives

Directives provide the ability to manipulate the environment while processing elements.

There is a single type of directives available: the Base directive.

`directive = base-directive`

It is an error if a document processor encounters any other type of directive.

3.1.2.1. Base Directives

A Base directive is encoded as a CBOR array that contains the unsigned integer 1 and a base URI.

`base-directive = [1, baseURI]`

The base URI is denoted by a Constrained Resource Identifier (CRI) reference [I-D.ietf-core-href]. The CRI reference MUST be resolved against the current context (not the current base).

`baseURI = CRI-Reference`

`CRI-Reference = <Defined in Section XX of RFC XXXX>`

The directive is processed by resolving the CRI reference against the current context and assigning the result to the current base.

3.1.3. URIs

URIs in links and forms are encoded as CRI references.

`URI = CRI-Reference`

A CRI reference is processed by resolving it to a URI as specified in Section 5.2 of [I-D.ietf-core-href] using the current base.

3.1.4. Links

A link is encoded as a CBOR array that contains the unsigned integer 2, the link relation type, the link target, and, optionally, an array of zero or more nested elements.

```
link = [2, relation-type, link-target, ?[*element]]
```

The link relation type is a URI.

```
relation-type = URI
```

The link target is either a URI, a literal value, or null.

```
link-target = URI / literal / null  
literal = bool / int / float / time / bytes / text
```

The nested elements, if any, MUST be processed in a fresh environment. The current context is set to the link target of the enclosing link. The current base is initially set to the link target, if the link target is a URI; otherwise, it is set to the current base of the current environment.

3.1.5. Forms

A form is encoded as a CBOR array that contains the unsigned integer 3, the operation type, the submission target, and, optionally, an array of zero or more form fields.

```
form = [3, operation-type, submission-target, ?[*form-field]]
```

The operation type is a URI.

```
operation-type = URI
```

The submission target is a URI.

```
submission-target = URI
```

The request method is either implied by the operation type or encoded as a form field. If both are given, the form field takes precedence over the operation type. Either way, the method MUST be applicable to the Web transfer protocol identified by the scheme of the submission target.

The form fields, if any, MUST be processed in a fresh environment. The current context is set to an unspecified URI that represents the enclosing form. The current base is initially set to the submission target of the enclosing form.

3.1.6. Form Fields

A form field is encoded as a CBOR sequence that consists of a form field type, a form field value, and, optionally, an array of zero or more nested elements.

form-field = (form-field-type, form-field-value, ?[*element])

The form field type is a URI.

form-field-type = URI

The form field value is either a URI, a literal value, or null.

form-field-value = URI / literal / null

The nested elements, if any, MUST be processed in a fresh environment. The current context is set to the form field value of the enclosing form field. The current base is initially set to the form field value, if the form field value is a URI; otherwise, it is set to the current base of the current environment.

3.2. Dictionary Compression

A document in the binary format MAY reference values from an external dictionary using Packed CBOR [I-D.ietf-cbor-packed]. This helps to reduce representation size and processing cost.

Dictionary references can be used subject to [yet to be defined] profiling.

Implementers should note that Packed CBOR is not designed to be uncompressed, but to be used in a compressed form. In particular, constrained devices may operate without even knowing what a given dictionary entry expands to (as long as they know its meaning) .

3.2.1. Media Type Parameter

The application/coral+cbor media type for documents in the binary format is defined to have a dictionary parameter that specifies the dictionary in use. The dictionary is identified by a URI. For example, a CoRAL document that uses the dictionary identified by the URI `<http://example.com/dictionary>` would have the following content type:

```
application/coral+cbor;dictionary="http://example.com/dictionary"
```

The URI serves only as an identifier; it does not necessarily have to be dereferencable (or even use a dereferencable URI scheme). It is permissible, though, to use a dereferencable URI and to serve a representation that provides information about the dictionary in a machine- or human-readable way. (The representation format and security considerations of such a representation are outside the scope of this document.)

For simplicity, a CoRAL document can reference values only from one dictionary; the value of the dictionary parameter **MUST** be a single URI.

The dictionary parameter is **OPTIONAL**. If it is absent, the default dictionary specified in Appendix B of this document is assumed.

Once a dictionary has made an assignment, the assignment **MUST NOT** be changed or removed. A dictionary, however, may contain additional information about an assignment, which may change over time.

In CoAP, media types (including specific values for their parameters, plus an optional content coding) are encoded as an unsigned integer called the "content format" of a representation. For use with CoAP, each new CoRAL dictionary therefore needs to have a new content format registered in the CoAP Content Formats Registry [CORE-PARAMETERS].

3.3. Export Interface

The definition of documents, links, and forms in the CoRAL binary format can be reused in other CBOR-based protocols. Specifications using CDDL should reference the following rules for this purpose:

```
CoRAL-Document = document
CoRAL-Link = link
CoRAL-Form = form
```

For each embedded document, link, and form, the CBOR-based protocol needs to specify the document retrieval context, link context, and form context, respectively.

4. Document Semantics

4.1. Submitting Documents

By default, a CoRAL document is a representation that captures the current state of a resource. The meaning of a CoRAL document changes when it is submitted in a request. Depending on the request method, the CoRAL document can capture the intended state of a resource (PUT) or be subject to application-specific processing (POST).

4.1.1. PUT Requests

A PUT request with a CoRAL document enclosed in the request payload requests that the state of the target resource be created or replaced with the state described by the CoRAL document. A successful PUT of a CoRAL document generally means that a subsequent GET on that same target resource would result in an equivalent document being sent in a success response.

An origin server SHOULD verify that a submitted CoRAL document is consistent with any constraints the server has for the target resource. When a document is inconsistent with the target resource, the origin server SHOULD either make it consistent (e.g., by removing inconsistent elements) or respond with an appropriate error message containing sufficient information to explain why the document is unsuitable.

The retrieval context and the base URI of a CoRAL document in a PUT are the request URI of the request.

4.1.2. POST Requests

A POST request with a CoRAL document enclosed in the request payload requests that the target resource process the CoRAL document according to the resource's own specific semantics.

The retrieval context of a CoRAL document in a POST is defined by the target resource's processing semantics; it may be an unspecified URI. The base URI of the document is the request URI of the request.

4.2. Returning Documents

In a response, the meaning of a CoRAL document changes depending on the request method and the response status code. For example, a CoRAL document in a successful response to a GET represents the current state of the target resource, whereas a CoRAL document in a successful response to a POST might represent either the processing result or the new resource state. A CoRAL document in an error response represents the error condition, usually describing the error state and what next steps are suggested for resolving it.

4.2.1. Success Responses

Success responses have a response status code that indicates that the client's request was successfully received, understood, and accepted (2xx in HTTP, 2.xx in CoAP). When the representation in a success response does not describe the state of the target resource, it describes result of processing the request. For example, when a request has been fulfilled and has resulted in one or more new resources being created, a CoRAL document in the response can link to and describe the resource(s) created.

The retrieval context and the base URI of a CoRAL document representing the current state of a resource are the request URI of the request.

The retrieval context of a CoRAL document representing a processing result is an unspecified URI that refers to the processing result itself. The base URI of the document is the request URI of the request.

4.2.2. Redirection Responses

Redirection responses have a response status code that indicates that further action needs to be taken by the agent (3xx in HTTP). A redirection response, for example, might indicate that the target resource is available at a different URI or the server offers a choice of multiple matching resources, each with its own specific URI.

In the latter case, the representation in the response might contain a list of resource metadata and URI references (i.e., links) from which the agent can choose the most preferred one.

The retrieval context of a CoRAL document representing such multiple choices in a redirection response is an unspecified URI that refers to the redirection itself. The base URI of the document is the request URI of the request.

4.2.3. Error Responses

Error response have a response status code that indicates that either the request cannot be fulfilled or the server failed to fulfill an apparently valid request (4xx or 5xx in HTTP, 4.xx or 5.xx in CoAP). A representation in an error response describes the error condition.

The retrieval context of a CoRAL document representing such an error condition is an unspecified URI that refers to the error condition itself. The base URI of the document is the request URI of the request.

5. Usage Considerations

This section discusses some considerations in creating CoRAL-based applications and vocabularies.

5.1. Specifying CoRAL-based Applications

CoRAL-based applications naturally implement the Web architecture [W3C.REC-webarch-20041215] and thus are centered around orthogonal specifications for identification, interaction, and representation:

- * Resources are identified by URIs or represented by literal values.
- * Interactions are based on the hypermedia interaction model of the Web and the methods provided by the Web transfer protocol. The semantics of possible interactions are identified by link relation types and operation types.
- * Representations are CoRAL documents encoded in the binary format defined in Section 3. Depending on the application, additional representation formats may be used.

5.1.1. Application Interfaces

Specifications for CoRAL-based applications need to list the specific components used in the application interface and their identifiers. This should include the following items:

- * The Web transfer protocols supported.
- * The representation formats used, identified by their Internet media types, including the CoRAL serialization formats.
- * The link relation types used.

- * The operation types used. Additionally, for each operation type, the permissible request methods.
- * The form field types used. Additionally, for each form field type, the permissible form field values.

5.1.2. Resource Identifiers

URIs are a cornerstone of Web-based applications. They enable the uniform identification of resources and are used every time a client interacts with a server or a resource representation needs to refer to another resource.

URIs often include structured application data in the path and query components, such as paths in a filesystem or keys in a database. It is a common practice in HTTP-based application programming interfaces (APIs) to make this part of the application specification, i.e., to prescribe fixed URI templates that are hard-coded in implementations. However, there are a number of problems with this practice [RFC8820].

In CoRAL-based applications, resource names are therefore not part of the application specification --- they are an implementation detail. The specification of a CoRAL-based application MUST NOT mandate any particular form of resource name structure.

[RFC8820] describes the problematic practice of fixed URI structures in more detail and provides some acceptable alternatives.

5.1.3. Implementation Limits

This document places no restrictions on the number of elements in a CoRAL document or the depth of nested elements. Applications using CoRAL (in particular those running in constrained environments) may limit these numbers and define specific implementation limits that an implementation must support at least to be interoperable.

Applications may also mandate the following and other restrictions:

- * Use of only either HTTP or CoAP as the supported Web transfer protocol.
- * Use of only dictionary references in the binary format for certain vocabulary.
- * Use of URI references and CRI references only up to a specific length.

5.2. Minting Vocabulary

New link relation types, operation types, and form field types can be minted by defining a URI that uniquely identifies the item. Although the URI may point to a resource that contains a definition of the semantics, clients SHOULD NOT automatically access that resource to avoid overburdening its server. The URI SHOULD be under the control of the person or party defining it, or be delegated to them.

To avoid interoperability problems, it is RECOMMENDED that only URIs are minted that are normalized according to Section 6.2 of [RFC3986]. This is easily achieved when the URIs are defined in CRI form (in which they also become part of the dictionary), as this avoids many common non-normalized forms of URIs by construction.

Non-normalized forms that are still to be avoided include:

- * Uppercase characters in scheme names and domain names
- * Explicitly stated HTTP default port (e.g., <http://example.com/> is preferable over <http://example.com:80/>)
- * Punycode-encoding of Internationalized Domain Names in URIs
- * URIs that are not in Unicode Normalization Form C

URIs that identify vocabulary do not need to be registered. The inclusion of domain names in URIs allows for the decentralized creation of new URIs without the risk of collisions.

However, URIs can be relatively verbose and impose a high overhead on a representation. This can be a problem in constrained environments [RFC7228]. Therefore, CoRAL alternatively allows the use of packed references that abbreviate CBOR data items from a dictionary, as specified in Section 3.2. These impose a much smaller overhead but instead need to be assigned by an authority to avoid collisions.

5.3. Expressing Registered Link Relation Types

Link relation types registered in the Link Relations Registry [LINK-RELATIONS], such as collection [RFC6573] or icon [W3C.REC-html52-20171214], can be used in CoRAL by appending the registered name to the URI <http://www.iana.org/assignments/relation/>:

```
#using iana = <http://www.iana.org/assignments/relation/>
```

```
iana:collection </items>
iana:icon        </favicon.png>
```

The convention of appending the relation type name to the prefix `<http://www.iana.org/assignments/relation/>` to form URIs is adopted from the Atom Syndication Format [RFC4287]; see also Appendix A.2 of [RFC8288].

Note that registered relation type names are required to be lowercase ASCII letters (see Section 3.3 of [RFC8288]).

5.4. Expressing Simple RDF Statements

In RDF [W3C.REC-rdf11-concepts-20140225], a statement says that some relationship, indicated by a predicate, holds between two resources. Existing RDF vocabularies can therefore be a good source for link relation types that describe resource metadata. For example, a CoRAL document could use the FOAF vocabulary [FOAF] to describe the person or software that made it:

```
#using rdf = <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
#using foaf = <http://xmlns.com/foaf/0.1/>
```

```
foaf:maker null {
  rdf:type      <http://xmlns.com/foaf/0.1/Person>
  foaf:familyName "Hartke"
  foaf:givenName  "Klaus"
  foaf:mbox       <mailto:klaus.hartke@ericsson.com>
}
```

5.5. Expressing Natural Language Texts

Text strings can be associated with a Language Tag [RFC5646] and a base text direction (right-to-left or left-to-right) by using CBOR tag 38.

```
#using base = <http://coreapps.org/base#>
#using iana = <http://www.iana.org/assignments/relation/>

iana:terms-of-service </tos> {
  base:title 38(["de", "Nutzungsbedingungen"])
  base:title 38(["en-US", "Terms of use"])
  base:title 38(["az", "ltr", "stifad rtlri"])
}
```

[Maturity note: Whether direction will actually be expressed in an updated tag 38, how precisely that is done, or whether a new tag will be allocated for text with direction is currently still under discussion.]

5.6. Embedding Representations in CoRAL

When a document links to many Web resources and an agent needs a representation of each of them, it can be inefficient to retrieve each representations individually. To minimize round-trips, documents can embed representations of resources.

A representation can be embedded in a document by including a link of type `<http://coreapps.org/base#representation>`:

```
#using base = <http://coreapps.org/base#>
#using http = <http://coreapps.org/http#>
#using iana = <http://www.iana.org/assignments/relation/>

iana:icon </favicon.gif> {
  base:representation
    b64'R0lGODlhAQABAAAAACH5BAEKAAEALAAAAABAAEAAAIAOw==' {
    http:type "image/gif"
  }
}
```

An embedded representation SHOULD have a nested link of type `<http://coreapps.org/http#type>` or `<http://coreapps.org/coap#type>` that indicates the content type of the representation.

The link relation types `<http://coreapps.org/base#representation>`, `<http://coreapps.org/http#type>`, and `<http://coreapps.org/coap#type>` are defined in Appendix A.

6. Security Considerations

CoRAL document processors need to be fully prepared for all types of hostile input that may be designed to corrupt, overrun, or achieve control of the agent processing the document. For example, hostile input may be constructed to overrun buffers, allocate very big data structures, or exhaust the stack depth by setting up deeply nested elements. Processors need to have appropriate resource management to mitigate these attacks.

CoRAL serialization formats intentionally do not feature the equivalent of XML entity references so as to preclude the entire class of attacks relating to them, such as exponential XML entity expansion ("billion laughs") [CAPEC-197] and malicious XML entity linking [CAPEC-201].

Implementers of the CoRAL binary format need to consider the security aspects of decoding CBOR. See Section 10 of [RFC8949] for security considerations relating to CBOR. In particular, different number encodings for the same numeric value are not equivalent in CoRAL (e.g., a floating-point value of 0.0 is not the same as the integer 0).

CoRAL makes extensive use of resource identifiers. See Section 7 of [RFC3986] for security considerations relating to URIs. See Section 7 of [I-D.ietf-core-href] for security considerations relating to CRIs.

The security of applications using CoRAL can depend on the proper preparation and comparison of internationalized strings. For example, such strings can be used to make authentication and authorization decisions, and the security of an application could be compromised if an entity providing a given string is connected to the wrong account or online resource based on different interpretations of the string. See [RFC6943] for security considerations relating to identifiers in URIs and other strings.

CoRAL is intended to be used in conjunction with a Web transfer protocol like HTTP or CoAP. See Section 9 of [RFC7230], Section 9 of [RFC7231], etc., for security considerations relating to HTTP. See Section 11 of [RFC7252] for security considerations relating to CoAP.

CoRAL does not define any specific mechanisms for protecting the confidentiality and integrity of CoRAL documents. It relies on security mechanisms on the application layer or transport layer for this, such as Transport Layer Security (TLS) [RFC8446].

CoRAL documents and the structure of a web of resources revealed from automatically following links can disclose personal information and other sensitive information. Implementations need to prevent the unintentional disclosure of such information. See Section 9 of [RFC7231] for additional considerations.

Applications using CoRAL ought to consider the attack vectors opened by automatically following, trusting, or otherwise using links and forms in CoRAL documents. See Section 5 of [RFC8288] for related considerations.

In particular, when a CoRAL document is the representation of a resource, the server that is authoritative for that resource may not necessarily be authoritative for nested elements in the document. In this case, unless an application defines specific rules, any link or form where the link/form context and the document's retrieval context do not share the same Web Origin [RFC6454] should be discarded ("same-origin policy").

7. IANA Considerations

7.1. Media Type "application/coral+cbor"

This document registers the media type application/coral+cbor according to the procedures of [RFC6838].

Type name:

application

Subtype name:

coral+cbor

Required parameters:

N/A

Optional parameters:

dictionary - See Section 3.2 of [I-D.ietf-core-coral].

Encoding considerations:

binary - See Section 3 of [I-D.ietf-core-coral].

Security considerations:

See Section 6 of [I-D.ietf-core-coral].

Interoperability considerations:

N/A

Published specification:

[I-D.ietf-core-coral]

Applications that use this media type:

See Section 1 of [I-D.ietf-core-coral].

Fragment identifier considerations:

As specified for application/cbor.

Additional information:

Deprecated alias names for this type: N/A

Magic number(s): N/A

File extension(s): .coral.cbor
Macintosh file type code(s): N/A

Person & email address to contact for further information:
See the Author's Address section of [I-D.ietf-core-coral].

Intended usage:
COMMON

Restrictions on usage:
N/A

Author:
See the Author's Address section of [I-D.ietf-core-coral].

Change controller:
IESG

Provisional registration?
No

7.2. CoAP Content Formats

This document registers CoAP content formats for the content types application/coral+cbor and text/coral according to the procedures of [RFC7252].

* Content Type: application/coral+cbor
Content Coding: identity
ID: TBD3
Reference: [I-D.ietf-core-coral]

[[NOTE TO RFC EDITOR: Please replace all occurrences of TBD3 in this document with the code points assigned by IANA.]]

[[NOTE TO IMPLEMENTERS: Experimental implementations may use content format ID 65087 for application/coral+cbor until IANA has assigned code points.]]

8. References

8.1. Normative References

- [I-D.bormann-cbor-edn-literals]
Bormann, C., "Application-Oriented Literals in CBOR Extended Diagnostic Notation", Work in Progress, Internet-Draft, draft-bormann-cbor-edn-literals-00, 6 October 2021, <<https://datatracker.ietf.org/doc/html/draft-bormann-cbor-edn-literals-00>>.
- [I-D.ietf-cbor-packed]
Bormann, C., "Packed CBOR", Work in Progress, Internet-Draft, draft-ietf-cbor-packed-04, 13 February 2022, <<https://datatracker.ietf.org/doc/html/draft-ietf-cbor-packed-04>>.
- [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>>.
- [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>>.
- [RFC3339] Klyne, G. and C. Newman, "Date and Time on the Internet: Timestamps", RFC 3339, DOI 10.17487/RFC3339, July 2002, <<https://www.rfc-editor.org/rfc/rfc3339>>.
- [RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, DOI 10.17487/RFC3629, November 2003, <<https://www.rfc-editor.org/rfc/rfc3629>>.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <<https://www.rfc-editor.org/rfc/rfc3986>>.
- [RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <<https://www.rfc-editor.org/rfc/rfc4648>>.
- [RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <<https://www.rfc-editor.org/rfc/rfc5234>>.

- [RFC5646] Phillips, A., Ed. and M. Davis, Ed., "Tags for Identifying Languages", BCP 47, RFC 5646, DOI 10.17487/RFC5646, September 2009, <<https://www.rfc-editor.org/rfc/rfc5646>>.
- [RFC6454] Barth, A., "The Web Origin Concept", RFC 6454, DOI 10.17487/RFC6454, December 2011, <<https://www.rfc-editor.org/rfc/rfc6454>>.
- [RFC6657] Melnikov, A. and J. Reschke, "Update to MIME regarding "charset" Parameter Handling in Textual Media Types", RFC 6657, DOI 10.17487/RFC6657, July 2012, <<https://www.rfc-editor.org/rfc/rfc6657>>.
- [RFC6838] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, DOI 10.17487/RFC6838, January 2013, <<https://www.rfc-editor.org/rfc/rfc6838>>.
- [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>>.
- [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>>.
- [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>>.
- [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/>>.

8.2. Informative References

- [CAPEC-197] MITRE, "CAPEC-197: XML Entity Expansion", September 2019, <<https://capec.mitre.org/data/definitions/197.html>>.
- [CAPEC-201] MITRE, "CAPEC-201: XML Entity Linking", September 2019, <<https://capec.mitre.org/data/definitions/201.html>>.

- [CORE-PARAMETERS]
IANA, "Constrained RESTful Environments (CoRE) Parameters",
<<https://www.iana.org/assignments/core-parameters>>.
- [FOAF] Brickley, D. and L. Miller, "FOAF Vocabulary Specification 0.99", January 2014,
<<http://xmlns.com/foaf/spec/20140114.html>>.
- [HAL] Kelly, M., "JSON Hypertext Application Language", Work in Progress, Internet-Draft, draft-kelly-json-hal-08, 12 May 2016, <<https://datatracker.ietf.org/doc/html/draft-kelly-json-hal-08>>.
- [HTTP-METHODS]
IANA, "Hypertext Transfer Protocol (HTTP) Method Registry",
<<https://www.iana.org/assignments/http-methods>>.
- [I-D.ietf-httpapi-linkset]
Wilde, E. and H. V. D. Sompel, "Linkset: Media Types and a Link Relation Type for Link Sets", Work in Progress, Internet-Draft, draft-ietf-httpapi-linkset-08, 8 February 2022, <<https://datatracker.ietf.org/doc/html/draft-ietf-httpapi-linkset-08>>.
- [LINK-RELATIONS]
IANA, "Link Relations",
<<https://www.iana.org/assignments/link-relations>>.
- [MEDIA-TYPES]
IANA, "Media Types",
<<https://www.iana.org/assignments/media-types>>.
- [RFC4287] Nottingham, M., Ed. and R. Sayre, Ed., "The Atom Syndication Format", RFC 4287, DOI 10.17487/RFC4287, December 2005, <<https://www.rfc-editor.org/rfc/rfc4287>>.
- [RFC5789] Dusseault, L. and J. Snell, "PATCH Method for HTTP", RFC 5789, DOI 10.17487/RFC5789, March 2010, <<https://www.rfc-editor.org/rfc/rfc5789>>.
- [RFC6573] Amundsen, M., "The Item and Collection Link Relations", RFC 6573, DOI 10.17487/RFC6573, April 2012, <<https://www.rfc-editor.org/rfc/rfc6573>>.

- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <<https://www.rfc-editor.org/rfc/rfc6690>>.
- [RFC6943] Thaler, D., Ed., "Issues in Identifier Comparison for Security Purposes", RFC 6943, DOI 10.17487/RFC6943, May 2013, <<https://www.rfc-editor.org/rfc/rfc6943>>.
- [RFC7089] Van de Sompel, H., Nelson, M., and R. Sanderson, "HTTP Framework for Time-Based Access to Resource States -- Memento", RFC 7089, DOI 10.17487/RFC7089, December 2013, <<https://www.rfc-editor.org/rfc/rfc7089>>.
- [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>>.
- [RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <<https://www.rfc-editor.org/rfc/rfc7230>>.
- [RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <<https://www.rfc-editor.org/rfc/rfc7231>>.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<https://www.rfc-editor.org/rfc/rfc7252>>.
- [RFC8132] van der Stok, P., Bormann, C., and A. Sehgal, "PATCH and FETCH Methods for the Constrained Application Protocol (CoAP)", RFC 8132, DOI 10.17487/RFC8132, April 2017, <<https://www.rfc-editor.org/rfc/rfc8132>>.
- [RFC8288] Nottingham, M., "Web Linking", RFC 8288, DOI 10.17487/RFC8288, October 2017, <<https://www.rfc-editor.org/rfc/rfc8288>>.
- [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/rfc/rfc8446>>.

- [RFC8820] Nottingham, M., "URI Design and Ownership", BCP 190, RFC 8820, DOI 10.17487/RFC8820, June 2020, <<https://www.rfc-editor.org/rfc/rfc8820>>.
- [UAX31] The Unicode Consortium, "Unicode Standard Annex #31: Unicode Identifier and Pattern Syntax", Revision 33, March 2020, <<https://www.unicode.org/reports/tr31/tr31-33.html>>.
- [UTR36] The Unicode Consortium, "Unicode Technical Report #36: Unicode Security Considerations", Revision 15, September 2014, <<https://www.unicode.org/reports/tr36/tr36-15.html>>.
- [UTS39] The Unicode Consortium, "Unicode Technical Standard #39: Unicode Security Mechanisms", Revision 22, February 2020, <<https://www.unicode.org/reports/tr39/tr39-22.html>>.
- [W3C.REC-html52-20171214]
Faulkner, S., Eicholz, A., Leithead, T., Danilo, A., and S. Moon, "HTML 5.2", World Wide Web Consortium Recommendation REC-html52-20171214, 14 December 2017, <<https://www.w3.org/TR/2017/REC-html52-20171214>>.
- [W3C.REC-rdf-schema-20140225]
Brickley, D. and R. Guha, "RDF Schema 1.1", World Wide Web Consortium Recommendation REC-rdf-schema-20140225, 25 February 2014, <<https://www.w3.org/TR/2014/REC-rdf-schema-20140225>>.
- [W3C.REC-rdf11-concepts-20140225]
Cyganiak, R., Wood, D., and M. Lanthaler, "RDF 1.1 Concepts and Abstract Syntax", World Wide Web Consortium Recommendation REC-rdf11-concepts-20140225, 25 February 2014, <<https://www.w3.org/TR/2014/REC-rdf11-concepts-20140225>>.
- [W3C.REC-turtle-20140225]
Prud'hommeaux, E. and G. Carothers, "RDF 1.1 Turtle", World Wide Web Consortium Recommendation REC-turtle-20140225, 25 February 2014, <<https://www.w3.org/TR/2014/REC-turtle-20140225>>.
- [W3C.REC-webarch-20041215]
Jacobs, I. and N. Walsh, "Architecture of the World Wide Web, Volume One", World Wide Web Consortium Recommendation REC-webarch-20041215, 15 December 2004, <<https://www.w3.org/TR/2004/REC-webarch-20041215>>.

Appendix A. Core Vocabulary

This section defines the core vocabulary for CoRAL: a set of link relation types, operation types, and form field types.

A.1. Base

Link Relation Types:

`<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>`

Indicates that the link's context is an instance of the class specified as the link's target, as defined by RDF Schema [W3C.REC-rdf-schema-20140225].

`<http://coreapps.org/base#title>`

Indicates that the link target is a human-readable label (e.g., a menu entry).

The link target MUST be a literal. The text string SHOULD be wrapped in a tag indicating language and, if necessary, direction if applicable.

`<http://coreapps.org/base#representation>`

Indicates that the link target is a representation of the link context.

The link target MUST be a byte string.

The representation may be a full, partial, or inconsistent version of the representation served from the URI of the resource.

A link with this link relation type can occur as a top-level element in a document or as a nested element within a link. When it occurs as a top-level element, it provides an alternate representation of the document's retrieval context. When it occurs nested within a link, it provides a representation of link target of the enclosing link.

Operation Types:

`<http://coreapps.org/base#update>`

Indicates that the state of the form's context can be replaced with the state described by a representation submitted to the server.

This operation type defaults to the PUT method [RFC7231] [RFC7252] for both HTTP and CoAP. Typical overrides by a form field include the PATCH method [RFC5789] [RFC8132] for HTTP and CoAP and the iPATCH method [RFC8132] for CoAP.

<http://coreapps.org/base#search>

Indicates that the form's context can be searched by submitting a search query.

This operation type defaults to the POST method [RFC7231] for HTTP and the FETCH method [RFC8132] for CoAP. Typical overrides by a form field include the POST method [RFC7252] for CoAP.

A.2. Collections

Link Relation Types:

<http://www.iana.org/assignments/relation/item>

Indicates that the link's context is a collection and that the link's target is a member of that collection, as defined in Section 2.1 of [RFC6573].

<http://www.iana.org/assignments/relation/collection>

Indicates that the link's target is a collection and that the link's context is a member of that collection, as defined in Section 2.2 of [RFC6573].

Operation Types:

<http://coreapps.org/collections#create>

Indicates that the form's context is a collection and that a new item can be created in that collection with the state defined by a representation submitted to the server.

This operation type defaults to the POST method [RFC7231] [RFC7252] for both HTTP and CoAP.

<http://coreapps.org/collections#delete>

Indicates that the form's context is a member of a collection and that the form's context can be removed from that collection.

This operation type defaults to the DELETE method [RFC7231] [RFC7252] for both HTTP and CoAP.

A.3. HTTP

Form Field Types:

<http://coreapps.org/http#method>

Specifies the HTTP method for the request.

The form field value MUST be a text string in the format defined in Section 4.1 of [RFC7231]. The possible set of values is maintained in the HTTP Methods Registry [HTTP-METHODS].

A form field of this type MUST NOT occur more than once in a form. If absent, it defaults to the request method implied by the form's operation type.

<http://coreapps.org/http#accept>

Specifies an acceptable HTTP content type for the request payload. There may be multiple form fields of this type. If a form does not include a form field of this type, the server accepts any or no request payload, depending on the operation type.

The form field value MUST be a text string in the format defined in Section 3.1.1.1 of [RFC7231]. The possible set of media types and their parameters is maintained in the Media Types Registry [MEDIA-TYPES].

Link Relation Types:

<http://coreapps.org/http#type>

Specifies the HTTP content type of the link context.

The link target MUST be a text string in the format defined in Section 3.1.1.1 of [RFC7231]. The possible set of media types and their parameters is maintained in the Media Types Registry [MEDIA-TYPES].

A.4. CoAP

Form Field Types:

<http://coreapps.org/coap#method>

Specifies the CoAP method for the request.

The form field value MUST be an integer identifying a CoAP method (e.g., the integer 2 for the POST method). The possible set of values is maintained in the CoAP Method Codes Registry [CORE-PARAMETERS].

A form field of this type MUST NOT occur more than once in a form. If absent, it defaults to the request method implied by the form's operation type.

<http://coreapps.org/coap#accept>

Specifies an acceptable CoAP content format for the request payload. There may be multiple form fields of this type. If a form does not include a form field of this type, the server accepts any or no request payload, depending on the operation type.

The form field value MUST be an integer identifying a CoAP content format. The possible set of values is maintained in the CoAP Content Formats Registry [CORE-PARAMETERS].

Link Relation Types:

<http://coreapps.org/coap#type>

Specifies the CoAP content format of the link context.

The link target MUST be an integer identifying a CoAP content format (e.g., the integer 42 for the content type application/octet-stream without a content coding). The possible set of values is maintained in the CoAP Content Formats Registry [CORE-PARAMETERS].

Appendix B. Default Dictionary

This section defines a default dictionary that is assumed when the application/coral+cbor media type is used without a dictionary parameter.

Key	Value
0	<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
1	<http://www.iana.org/assignments/relation/item>
2	<http://www.iana.org/assignments/relation/collection>
3	<http://coreapps.org/collections#create>
4	<http://coreapps.org/base#update>
5	<http://coreapps.org/collections#delete>
6	<http://coreapps.org/base#search>
7	<http://coreapps.org/coap#accept>
8	<http://coreapps.org/coap#type>
10	<http://coreapps.org/coap#method>
14	<http://coreapps.org/base#representation>

Table 3: Default Dictionary

Appendix C. Mappings to other formats

While CoRAL has an information model of its own, its data can be converted to different extents with other data formats.

Using these conversions is generally application specific, i.e., this document does not claim equivalence of (say) a given RDF its converted CoRAL document, but applications can choose use these conversions if the limitations described with the conversion are acceptable to them.

C.1. RDF

[TBD: Expand / introduce the common CURIEs used here.]

RDF and the CoRAL Basic Information Model can be interconverted losslessly, as long as some basic restrictions are met:

- * All involved IRIs (on the RDF side) and CRIs (on the CoRAL side) can be converted; that means that round-tripping IRIs through CoRAL converts them to the equivalent URIs.

The precise limitations of what CRIs can not express are described in [I-D.ietf-core-href] and out of scope of this document.

A possible extension to CoRAL that allows tagged URIs in place of CRIs could remove this limitation. (CRIs that can not be expressed as URIs are not valid anyway).

- * A blank node of CoRAL can only have one incoming edge in serialization. RDF documents with multiply connected blank nodes need to undergo skolemization before they can be expressed in CoRAL.
- * CoRAL supports arbitrary literal objects, including CBOR tags. For each object that is used in a literal, a mapping to a datatype (typically XSD) needs to be defined.

When literals are normalized in RDF according to XSD rules, or the literal mappings to RDF datatypes are ambiguous on the CoRAL side, round-tripping CoRAL through RDF can be lossy to the extent of the normalization or ambiguity.

- * As always with expressing arbitrary graphs of the Basic Information Model in serialization, if there is no directed tree spanning the directed graph, statements need to be introduced to reach some topics.

Each statement in RDF is mapped to a statement in CoRAL. Any IRI it contains in RDF is mapped to an equivalent CRI in CoRAL and vice versa. Any blank node of RDF is converted to a blank node (serialized as a null) in CoRAL. (Beware that depending on the context established in Section 4, the retrieval context may be a URI or a blank node). Literals are converted as follows:

- * CBOR text strings are converted to RDF string literals without a language tag.
- * CBOR literals from the following list are converted to their corresponding text representations of the datatype from the following table:

CDDL	XSD datatype
bool	xsd:boolean
integer	xsd:integer
float	xsd:double
decfrac	xsd:decimal
bytes	xsd:base64Binary or xsd:base64hexBinary (?)
tdate	xsd:date
#6.38([lang: tstr, text: tstr])	rdf:langString with lang as language tag
#6.TBD([lang: tstr, dir: tstr, text: tstr])	i18n:{lang}_{dir}

Table 4: Mapping between CDDL types and XSD datatypes

[TBD: Check compatibilities, give type for at least the basic tags.
Directional text might wind up in tag 38,]

- * RDF literals are mapped to any CoRAL literal that yields an equivalent RDF literal in the opposite direction.

C.1.1. Example

The FOAF namespace provides this example:

```
<foaf:Person rdf:about="#danbri" xmlns:foaf="http://xmlns.com/foaf/0.1/">
  <foaf:name>Dan Brickley</foaf:name>
  <foaf:homepage rdf:resource="http://danbri.org/" />
  <foaf:openid rdf:resource="http://danbri.org/" />
  <foaf:img rdf:resource="/images/me.jpg" />
</foaf:Person>
```

Figure 5: Original FOAF file at <http://.../me.xml>

Converted, assuming no particular profiling or dictionary setup (and an ad-hoc table following Section 3.1 of [I-D.ietf-chor-packed]), this could be:

```

51([[cri'http://danbri.org/'], [<-3, "xmlns.com", ["foaf", "0.1"], null>>], [],
[
  [2, cri'http://www.iana.org/assignments/relation/carries-information-about', cr
i'/me.xml#danbri',
    [2, cri'http://www.w3.org/1999/02/22-rdf-syntax-ns#type', 6(<<'Person'>>)],
    [2, 6(<<'name'>>), "Dan Brickley"],
    [2, 6(<<'homepage'>>), 6(0)],
    [2, 6(<<'openid'>>), 6(0)],
    [2, 6(<<'img'>>), cri'/images/me.jpg']
  ]
])

```

Figure 6: Serialized FOAF file at <http://.../me.coral>

The TBD:talks-about statement is introduced to bridge the gap between the basic and the necessarily structured information model. [TBD: Introduce that somewhere else more generally.]

In this packing, an invalid CRI (with trailing null leaving room for a fragment identifier to be added through packing) is added into the prefixes list. It is not sure whether this particular trick will ever be permitted by any of the profilings, or whether this is better done with base URIs. The mechanism is used because right now it works with the specifications involved without the need for further text, and is likely to be replaced by better mechanisms in later revisions of this document.

C.2. CoRE Link Format

Generic information in Web Links as described in [RFC8288] can not be converted to CoRAL in any practical way: Attributes are not managed, and it is not clear from the syntax whether an attribute is making a statement about the link or its target. (See Section 2.3.3.1 for an example).

Applications that use links with the attribute semantics common in the CoRE ecosystem (typically used with [RFC6690] Link Format) can use this conversion. It defines terms for common properties used for discovering resources, and describes a way to compatibly extend the mapping.

The same mechanism (but probably with a different mapping between names and attributes, and different rules about the necessity of packing entries) can be defined for any data model that builds on [RFC8288] semantics, e.g., the links sent in headers or payloads about [RFC7089] mementos, or applications building on [I-D.ietf-httpapi-linkset].

In several points the mapping describes URIs to necessarily have an entry in the packing table; this refers to the profiling described further down. Parts of a Link Format document that would need an entry but do not have one can not be converted; these are ignored in the conversion unless the converter is configured to be strict and fail the complete conversion in that case.

This mapping from Link Format to CoRAL is performed as follows: * For each relation in a link, a statement is created mapping the link context to the subject, the link target to the object and the relation to the predicate.

If the relation is of ext-rel-type, it is used as a URI as is. Otherwise it is a registered value, prefixed with <http://www.iana.org/assignments/relation/> and necessarily packed using table TBD. (This is equivalent to the RPP mechanism for attribute values).

- * Each target attribute is converted to one or more statements by the mechanism indicated for the attribute name in the following table. Statements produced from a link have the target as its subject, the attribute name without any trailing asterisk (prefixed with <https://TBD/> [to be picked together with IANA as it'll be a registry]) as its predicate, and the object(s) depending on the mechanism.

Attributes are necessarily listed in this table.

TN	Name	Mechanism
TBD	hreflang	[do we need that?]
TBD	media	[do we need that?]
16	title	string
TBD	type	[do we need that?]
0	rt	WSSP; RPP http://www.iana.org/TBDr/
1	if	WSSP; RPP http://www.iana.org/TBDi/
2	sz	int
3	ct	WSSP; int

Table 5: Initial entries of the target attribute registry (TN = table number)

Available mechanisms are:

- * SPSP (space split): Link format values are split at space characters (SP in the RFC6690 ABNF), and all values treated using another mechanism.
- * string: The attribute value is stored as a text string literal. If the Link Format attribute is language tagged (i.e. when the attribute name ends with an asterisk and the value is of ext-value shape), the literal is encapsulated in a CBOR language tag (38).
- * int: The target attribute is processed as an ASCII encoded number and expressed as an integer literal. A failing conversion is treated like an unknown registered value: It is ignored unless configured otherwise.
- * RPP (registered-prefix / packed): The input value (often the result of the SPSP mechanism) is parsed according to the relation-type ABNF production. If it is of ext-rel-type, it is expressed as that URI. If it is prefixed with the string indicated with the mechanism, and necessarily compressed through table TBD.

All currently registered link attributes are used in the CoRE ecosystem as indicating a property of the target that is independent of the link being followed. If this conversion is to be extended to

cover attributes that pertain to the full link being followed (typically along with one or more link relations), the relevant relations are not expressed as a single statement, but as a form, i.e. as two statements linking the context to a blank node and the blank node to the target; the attributes are attached to the blank node. The precise mechanism out of scope for this document, and left to those who first register such an attribute.

Some structure can be carried over from Link Format to the structured model: The sequences of links gets reused, and the set and sequence of attributes in a particular occurrence of a link get applied to the statement produced from the link (or all the statements, if the link has multiple link relations). Statements whose subject is not the document itself are attached to the retrieval context using the necessarily packed `http://www.iana.org/assignments/relation/carries-information-about` property. Statements about URLs mentioned elsewhere in the document can be expressed there instead.

Link relations of the reg-name form, link attributes, and attribute values from the RPP mechanism MUST be serialized using packed CBOR as initialized in table TBD. No other packing is used. A consumer MAY ignore any items compressed through the dictionary for which it does not know the expanded version: These necessarily represent statements that involve terms the consumer does not understand.

[As an alternative, packing attributes together with their URIs is considered: Rather than `[2, 6(/ attr:rt /), 6(/ rt:core.rd /)]` we could have `6(rt-core)` right away; unregistered values would stay `[2, 6(/ attr:rt /), value]` or maybe `254([value])` using prefix packing.]

Appendix D. Change Log

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

Changes from -04 to -05:

- * Literals can no longer have properties. The only use case was annotating languages and directions, and that can be done in CBOR.
- * Added section about open and close world modelling.
- * Information model merged with the previous data model and interaction section.

Changes from -03 to -04:

- * Formalize information model, as basic and structured model.

- * Remove textual representation, using CBOR diagnostig notation instead.
- * Use Packed CBOR instead of custom dictionaries.
- * Give explicit conversions from Link Format and with RDF.
- * Remove references to IRIs (outside RDF) as CRIs are closer to URIs.
- * Remove requirement for deterministic encoding.
- * Many editorial changes.
- * Update references.
- * Change of authorship.

Changes from -02 to -03:

- * Changed the binary format to express relation types, operation types and form field types using [I-D.ietf-core-href] (#2).
- * Clarified the current context and current base for nested elements and form fields (#53).
- * Minor editorial improvements (#27).

Changes from -01 to -02:

- * Added nested elements to form fields.
- * Replaced the special construct for embedded representations with links.
- * Changed the textual format to allow simple/qualified names wherever IRI references are allowed.
- * Introduced predefined names in the textual format (#39).
- * Minor editorial improvements and bug fixes (#16 #28 #31 #37 #39).

Changes from -00 to -01:

- * Added a section on the semantics of CoRAL documents in responses.
- * Minor editorial improvements.

Acknowledgements

The concept and original version of CoRAL (as well as CRIs) was developed by Klaus Hartke. It was heavily inspired by Mike Kelly's JSON Hypertext Application Language [HAL].

The recommendations for minting URIs have been adopted from RDF 1.1 Concepts and Abstract Syntax [W3C.REC-rdf11-concepts-20140225] to ease the interoperability between RDF predicates and link relation types.

Thanks to Carsten Bormann, Jaime Jiménez, Jim Schaad, Sebastian Käbisch, Ari Keränen, Michael Koster, Matthias Kovatsch and Niklas Widell for helpful comments and discussions that have shaped the document.

Authors' Addresses

Christian Amsüss
Email: christian@amsuess.com

Thomas Fossati
ARM
Email: thomas.fossati@arm.com

CoRE Working Group
Internet-Draft
Obsoletes: 7390 (if approved)
Updates: 7252, 7641 (if approved)
Intended status: Standards Track
Expires: 8 September 2022

E. Dijk
IoTconsultancy.nl
C. Wang
InterDigital
M. Tiloca
RISE AB
7 March 2022

Group Communication for the Constrained Application Protocol (CoAP)
draft-ietf-core-groupcomm-bis-06

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 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

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

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

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

This Internet-Draft will expire on 8 September 2022.

Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	4
1.1. Scope	5
1.2. Terminology	5
1.3. Changes to Other Documents	6
2. Group Definition and Group Configuration	7
2.1. Group Definition	7
2.1.1. CoAP Group	7
2.1.2. Application Group	7
2.1.3. Security Group	8
2.1.4. Relations Between Group Types	8
2.2. Group Configuration	11
2.2.1. Group Naming	11
2.2.2. Group Creation and Membership	17
2.2.3. Group Discovery	18
2.2.4. Group Maintenance	25
3. CoAP Usage in Group Communication	26
3.1. Request/Response Model	26
3.1.1. General	26
3.1.2. Response Suppression	27
3.1.3. Repeating a Request	27
3.1.4. Request/Response Matching and Distinguishing Responses	28
3.1.5. Token Reuse	28
3.1.6. Client Handling of Multiple Responses With Same Token	29

3.2.	Caching	30
3.2.1.	Freshness Model	31
3.2.2.	Validation Model	31
3.3.	URI Path Selection	32
3.4.	Port Selection for UDP Transport	33
3.5.	Proxy Operation	33
3.5.1.	Forward-Proxies	33
3.5.2.	Reverse-Proxies	35
3.6.	Congestion Control	37
3.7.	Observing Resources	38
3.8.	Block-Wise Transfer	40
3.9.	Transport Protocols	41
3.9.1.	UDP/IPv6 Multicast Transport	41
3.9.2.	UDP/IPv4 Multicast Transport	42
3.9.3.	6LoWPAN	42
3.9.4.	Other Transports	42
3.10.	Interworking with Other Protocols	43
3.10.1.	MLD/MLDv2/IGMP/IGMPv3	43
3.10.2.	RPL	43
3.10.3.	MPL	44
4.	Unsecured Group Communication	45
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	51
6.2.3.	Countering Attacks	52
6.3.	Risk of Amplification	54
6.4.	Replay of Non-Confirmable Messages	55
6.5.	Use of CoAP No-Response Option	56
6.6.	6LoWPAN and MPL	56
6.7.	Wi-Fi	57
6.8.	Monitoring	57
6.8.1.	General Monitoring	57
6.8.2.	Pervasive Monitoring	58
7.	IANA Considerations	58
8.	References	58
8.1.	Normative References	58
8.2.	Informative References	61
Appendix A.	Use Cases	64
A.1.	Discovery	64
A.1.1.	Distributed Device Discovery	64
A.1.2.	Distributed Service Discovery	65
A.1.3.	Directory Discovery	65

A.2. Operational Phase	65
A.2.1. Actuator Group Control	65
A.2.2. Device Group Status Request	66
A.2.3. Network-wide Query	66
A.2.4. Network-wide / Group Notification	66
A.3. Software Update	66
Appendix B. Examples of Message Exchanges	67
Appendix C. Document Updates	73
C.1. Version -05 to -06	73
C.2. Version -04 to -05	74
C.3. Version -03 to -04	74
C.4. Version -02 to -03	74
C.5. Version -01 to -02	75
C.6. Version -00 to -01	75
Acknowledgments	76
Authors' Addresses	76

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 supporting group communication include the framework "Eclipse Californium" 2.0.x [Californium] from the Eclipse Foundation and the "Implementation of CoAP Server & Client in Go" [Go-OCF] from the Open Connectivity Foundation (OCF).

Both unsecured and secured CoAP group communication 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) [I-D.ietf-cose-rfc8152bis-struct][I-D.ietf-cose-rfc8152bis-algs].

This documents 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 left for future work. 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 left for future work. 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 contains an optional 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 defines 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 "NoSec" mode (see Section 4 and Section 6.1) and highlights the risk of amplification attacks (see Section 6.3).

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).

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. Group membership(s) of an endpoint or node 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 this CoAP group: it is a member only if it also has a CoAP endpoint listening on the group's associated IP multicast address and UDP port.

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.

A special security group named "NoSec" identifies group communication without any security at the transport layer nor at the CoAP layer. Further details on identifying a security group are provided in Section 2.2.1.3.

2.1.4. Relations Between Group Types

Using the above group type definitions, a CoAP group communication message sent by an endpoint can be represented as 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 used server port number, 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 same 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 a same application group. That is, 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. It has to 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 UML class diagram notation. The class attributes in square brackets are optionally defined.

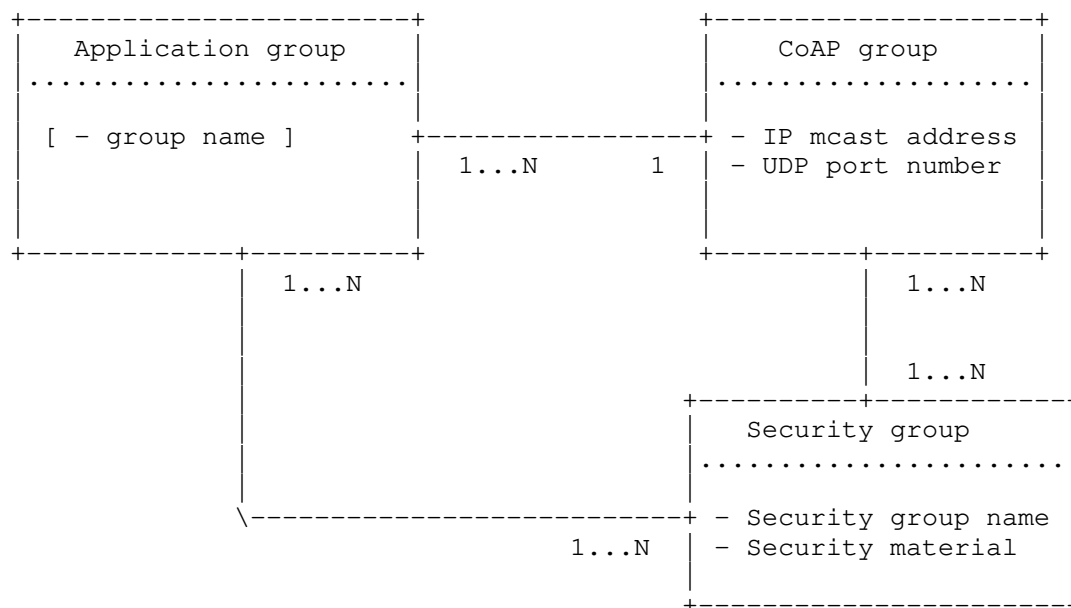


Figure 1: Relations Among Different Group Types

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). There are three application groups (1, 2, 3) and two security groups (1, 2). Security Group 1 is used by both Application Group 1 and 2. Three clients (Cli1, Cli2, Cli3) are configured with security material for Security Group 1. Two clients (Cli2, Cli4) are configured with security material for Security Group 2. All the shown application groups use the same CoAP group (not shown in the figure), i.e., one specific multicast IP address and UDP port number on which all the shown resources are hosted for each server.

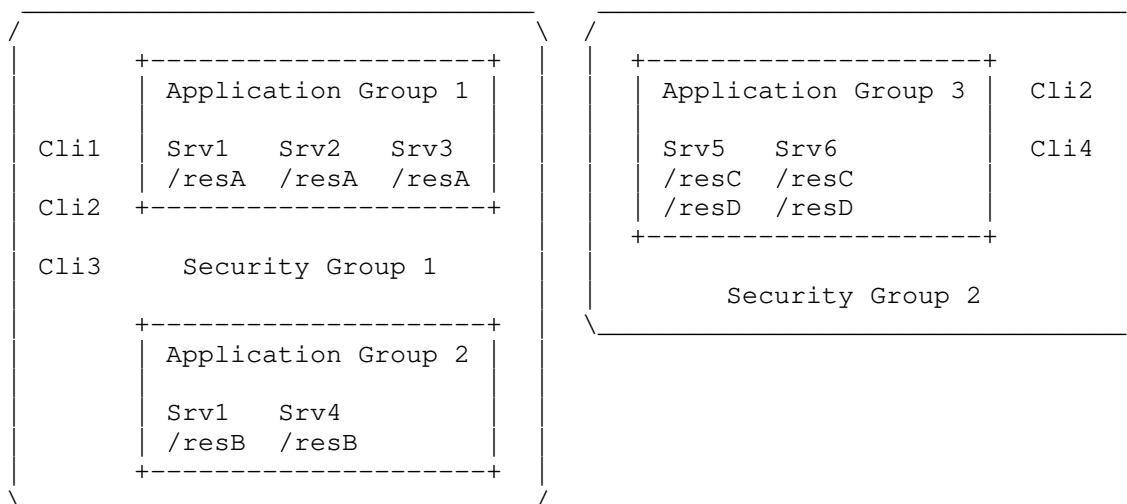


Figure 2: Deployment Example of Different Group Types

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 group are named as specified below, separately for CoAP groups, application groups and security groups.

2.2.1.1. CoAP Groups

A CoAP group is identified and named by the authority component in the group URI (see Section 2.1.1), which includes the host subcomponent (possibly an IP multicast address literal) and an optional UDP port number. Note that the two authority components <HOST> and <HOST>:5683 both identify the same CoAP group, whose members listen to the CoAP default port number 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 string. The decision of whether and how exactly an application group name is encoded and transported is application specific.

The following defines a number of possible methods to use. The shown examples consider a CoAP group identified by the group hostname `grp.example.org`. 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: 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).

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 `GROUPNAME` should be as short as possible.

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/GROUPNAME/res1` and `/base/gp/GROUPNAME/res1/res2` conform to this practice. Just like application group names, the path segment used as delimiter should be as short as possible in constrained networks.

A full-fledged example is provided in Figure 3. Another example of a compact CoAP request is provided in Figure 4, in which an IPv6 literal address and the default CoAP port number 5683 are used in the authority component. Also the resource structure is different in this example.

Application group name: gp1

Group URI: coap://grp.example.org:5685/gp/gp1/light?foo=bar

CoAP group request

Header: GET (T=NON, Code=0.01, MID=0x7d41)

Uri-Host: grp.example.org

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

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

- * URI query component - This method can use the following formats. In either case, when using this method in constrained networks, an application group name GROUPNAME 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 ?GROUPNAME conforms to this format.

A full-fledged example is provided in Figure 5.

- 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=GROUPNAME and ?par1=v1&gp=GROUPNAME conform to this format.

A full-fledged example is provided in Figure 6.

Application group name: gpl

Group URI: coap://grp.example.org:5685/light?gpl

CoAP group request

Header: GET (T=NON, Code=0.01, MID=0x7d41)

Uri-Host: grp.example.org

Uri-Path: light

Uri-Query: gpl

Figure 5: Example of application group name in URI query (1/2)

Application group name: gpl

Group URI: coap://grp.example.org:5685/light?foo=bar&gp=gpl

CoAP group request

Header: GET (T=NON, Code=0.01, MID=0x7d41)

Uri-Host: grp.example.org

Uri-Path: light

Uri-Query: foo=bar

Uri-Query: gp=gpl

Figure 6: Example of application group name in URI query (2/2)

- * 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).

A full-fledged example is provided in Figure 7.

Application group name: grp.example.org:5685

Group URI: coap://grp.example.org:5685/light?foo=bar

CoAP group request

Header: GET (T=NON, Code=0.01, MID=0x7d41)

Uri-Host: grp.example.org

Uri-Path: light

Uri-Query: foo=bar

Figure 7: Example of application group name in URI authority

- * 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 any two CoAP groups cannot differ only by the port subcomponent of the URI authority component.

A full-fledged example is provided in Figure 8.

Application group name: grp.example.org

Group URI: coap://grp.example.org:5685/light?foo=bar

CoAP group request

Header: GET (T=NON, Code=0.01, MID=0x7d41)

Uri-Host: grp.example.org

Uri-Path: light

Uri-Query: foo=bar

Figure 8: Example of application group name in URI host

- * 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-fledged example is provided in Figure 9.

Application group name: grp1, as inferable from port number 5685

Group URI: coap://grp.example.org:5685/light?foo=bar

CoAP group request

Header: GET(T=NON, Code=0.01, MID=0x7d41)

Uri-Host: grp.example.org

Uri-Path: light

Uri-Query: foo=bar

Figure 9: Example of application group name in URI port

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 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-fledged example is provided in Figure 10.

Application group name: grp1

Group URI: coap://grp.example.org:5685/light?foo=bar

CoAP group request

Header: GET (T=NON, Code=0.01, MID=0x7d41)

Uri-Host: grp.example.org

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

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 [I-D.ietf-core-resource-directory] provides an example of 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. At the same time, the CoAP group is ff35:30:2001:db8:f1::8000:1 and the "NoSec" security group is used.

2.2.1.3. Security Groups

A security group is identified by a stable and invariant string used as group name. This is generally not related with other kinds of group identifiers that may be specific of the used security solution.

The "NoSec" security group name MUST be only used to represent the case of group communication without any security. This typically results in CoAP messages that do not include any security group name, identifier, or security-related data structures.

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 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 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 configuration of group membership for unsecured group communication, based on JSON-formatted configuration resources. 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 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 specified group configuration.

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" security group, any CoAP endpoint may become a group member at any time: there is no configuring entity that needs to provide security material for this group, as there is no security material for it. This means that group creation and membership cannot be tightly controlled for the "NoSec" group.

The configuration of groups and membership may be performed at different moments in the life-cycle of a device; for example 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 [I-D.ietf-core-resource-directory].

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 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 shown 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 provides examples 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>". Third, a CoAP group is used and is identified by the URI authority `grp.example.org:5685`.

- * 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-fledged example is provided in Figure 11.

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.org:5685/.well-known/core?rt=g.*
```

```
// Response from server S1, as member of:
//   - The CoAP group "grp.example.org: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.org: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

- * 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/gpl"` to specify the application group with name `"gpl"`.

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.

A full-fledged example is provided in Figure 12. Note that the servers need to respond now with an absolute URI and not a relative URI like in the previous example, because the group resource `"gpl"` is hosted at the authority indicated by the CoAP group (`grp.example.org:5685`) and not by the authority that is serving the Link Format document (which is `[ff03::fd]:5683`).

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.

```
// Request to realm-local members of the application group "gpl"
Req: GET coap://[ff03::fd]/.well-known/core?href=/gp/gpl
```

```
// CoAP response from server S1, as member of:
// - The CoAP group "grp.example.org:5685"
// - The application group "gpl"
Res: 2.05 Content
Content-Format: 40
Payload:
<coap://grp.example.org:5685/gp/gpl>;rt=g.light
```

```
// CoAP response from server S2, as member of:
// - The CoAP group "grp.example.org:5685"
// - The application groups "gpl"
Res: 2.05 Content
Content-Format: 40
Payload:
<coap://grp.example.org:5685/gp/gpl>;rt=g.light
```

Figure 12: Discovery of members of an application group, together with the associated CoAP group

- * 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.

A full-fledged example is provided in Figure 13.

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.


```

// 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.org:5685"
//   - The application group "gp1" of type "g.temp"
Res: 2.05 Content
Content-Format: 40
Payload:
<coap://grp.example.org:5685/gp/gp1>;rt=g.temp

// Response from server S2, as member of:
//   - The CoAP group "grp.example.org:5685"
//   - The application groups "gp1" and "gp2" of type "g.temp"
Res: 2.05 Content
Content-Format: 40
Payload:
<coap://grp.example.org:5685/gp/gp1>;rt=g.temp,
<coap://grp.example.org:5685/gp/gp2>;rt=g.temp

```

Figure 13: Discovery of members of application groups of a specified type, and of the associated CoAP group

- * 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, 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. In this example the scope is realm-local to address all servers in the current 6LoWPAN wireless mesh network of the client.

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. The CoAP group per application group is obtained as the authority of each returned link.

A full-fledged example is provided in Figure 14.

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.

```
// 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.org:5685" and "grp2.example.org"
//   - The application groups "gp1" and "gp5"
Res: 2.05 Content
Content-Format: 40
Payload:
<coap://grp.example.org:5685/gp/gp1>;rt=g.light,
<coap://grp2.example.org/gp/gp5>;rt=g.lock

// Response from server S2, as member of:
//   - The CoAP group "grp.example.org:5685"
//   - The application groups "gp1" and "gp2"
Res: 2.05 Content
Content-Format: 40
Payload:
<coap://grp.example.org:5685/gp/gp1>;rt=g.light,
<coap://grp.example.org:5685/gp/gp2>;rt=g.light

// Response from server S3, as member of:
//   - The CoAP group "grp2.example.org"
//   - The application group "gp5"
Res: 2.05 Content
Content-Format: 40
Payload:
<coap://grp2.example.org/gp/gp5>;rt=g.lock
```

Figure 14: Discovery of the resources and members of any application group, and of the associated CoAP group

Note that all the above examples are 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 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.

For example, some applications may use the "rt" attribute on a parent resource to denote support for a particular REST API to access child resources, as detailed in the example in Figure 15. In this example 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, 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.org:5685" and "grp2.example.org"
//   - The application groups "gp1" and "gp5"
Res: 2.05 Content
Content-Format: 40
Payload:
<coap://grp.example.org:5685/gp/gp1>;rt=oic.d.light;gpt=light,
<coap://grp2.example.org/gp/gp5>;rt=oic.d.smartlock;gpt=lock

// Response from server S2, as member of:
//   - The CoAP group "grp.example.org:5685"
//   - The application groups "gp1" and "gp2"
Res: 2.05 Content
Content-Format: 40
Payload:
<coap://grp.example.org:5685/gp/gp1>;rt=oic.d.light;gpt=light,
<coap://grp.example.org:5685/gp/gp2>;rt=oic.d.light;gpt=light

// Response from server S3, as member of:
//   - The CoAP group "grp2.example.org"
//   - The application group "gp5"
Res: 2.05 Content
Content-Format: 40
Payload:
<coap://grp2.example.org/gp/gp5>;rt=oic.d.smartlock;gpt=lock
```

Figure 15: Example of using a custom 'gpt' link attribute to denote group type

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., the "NoSec" security group, 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. 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 be a mixture of success (e.g., 2.05 Content) and failure (e.g., 4.04 Not Found) codes, depending on the individual server processing results.

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 for a server to support this option only on selected resources where it is useful in the application context. If the option is supported on a resource, it MUST override the default response suppression of that resource.

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) group members have been reached with the request. This is useful in case a number of group members did not respond to the initial request 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 group members, 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:

$$\text{MIN_TOKEN_REUSE_TIME} = (\text{NON_LIFETIME} + \text{MAX_LATENCY} + \text{MAX_SERVER_RESPONSE_DELAY})$$

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.

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.tiloca-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 group members may have joined the group since the last group request to the same group/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 server group members; and, for each group member, the client's cache currently stores a fresh response.

How the client in the case above determines the current CoAP server group members is out of scope for this document. It may be, for example, via a group manager server, or by observing group join requests, or observing IGMP/MLD multicast group join messages, etc.

For caching at proxies, the freshness model defined in [I-D.tiloca-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.tiloca-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 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 of identical ETag values returned by two or more servers, the client, by default, SHOULD NOT include an ETag Option in a group request containing that ETag value.

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 can be done, 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 group members. However there are valid use cases where a group request is known to be successful only for a subset of the CoAP group, for example only members of a specific application group, while those group members for which the request is unsuccessful (for example because they are outside the application group) either ignore the group request or respond with an error status code.

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" multicast 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. 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. This request specifies the group URI where the request has to be forwarded to, 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.
- * The proxy does not necessarily know how many members there are in the CoAP group or how many group members will actually respond. Also, the proxy does not know for how long to collect responses before it stops forwarding them to the client. A CoAP client that is not using a Proxy might face the same problems in collecting responses to a group request. However, the client itself would typically have application-specific rules or knowledge on how to handle this situation, while an application-agnostic CoAP Proxy would typically not have this knowledge. For example, a CoAP client could monitor incoming responses and use this information to decide how long to continue collecting responses - which is something a proxy cannot do.

A forward-proxying method using this approach and addressing the issues raised above is defined in [I-D.tiloca-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 still identifies the CoAP servers by other means such as application-specific information in the response payload.

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 a group of servers 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].

Furthermore, the reverse-proxy can actually stand in for (and thus prevent to directly reach) only the whole set of servers in the group, or also for each of those individual servers (e.g., if acting as firewall).

For a reverse-proxy that sends a request to a group of servers, 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.tiloca-core-groupcomm-proxy] or other means.
- * A reverse-proxy MAY have preconfigured time duration(s) that are used for the collecting of server responses and forwarding these back to the client. These duration(s) may be set as global configuration or 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.tiloca-core-groupcomm-proxy] is not necessarily needed.
- * 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 group of servers, 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 signalling protocol of [I-D.tiloca-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 paragraph of Section 3.5.1 which applies also to the case of reverse-proxies.

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. 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 3.9.3) 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, that allows a CoAP client to retrieve a representation of a resource and automatically keep this representation up-to-date over a longer period of time. The client gets notified when the representation has changed. [RFC7641] does not mention whether the Observe Option can be combined with CoAP (multicast) 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], it is also possible to use 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 group members did not respond to the initial request. The client MAY additionally use the same Message ID in the repeated request to avoid that group members 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 group members 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 group member, that was expected to respond to the initial group request, did not respond to the initial request. In this case, the client MUST use a Message ID that differs from 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 liveness 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 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 cancelled 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 a group of servers through a CoAP-to-CoAP proxy, the limitations stated in Section 3.5 apply. The method defined in [I-D.tiloca-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.

The client has to use unicast for any further request, separately addressing each different server, in order to retrieve more blocks of the resource from that server, if any. 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.

[I-D.ietf-core-new-block] specifies an 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 group. An IPv6 CoAP server SHOULD support the "All CoAP Nodes" groups with at least link-local (2), admin-local (4) and site-local (5) scopes. An IPv6 CoAP server on a 6LoWPAN node (see Section 3.9.3) 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/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 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].

3.9.3. 6LoWPAN

In 6LoWPAN [RFC4944] [RFC6282] networks, IPv6 packets (up to 1280 bytes) may be fragmented into smaller IEEE 802.15.4 MAC frames (up to 127 bytes), if the packet size requires this. Every 6LoWPAN IPv6 router that receives a multi-fragment packet reassembles the packet and refragments it upon transmission. Since the loss of a single fragment implies the loss of the entire IPv6 packet, the performance in terms of packet loss and throughput of multi-fragment multicast IPv6 packets is typically far worse than the performance of single-fragment IPv6 multicast packets. For this reason, a CoAP request sent over multicast in 6LoWPAN networks SHOULD be sized in such a way that it fits in a single IEEE 802.15.4 MAC frame, if possible.

On 6LoWPAN networks, multicast groups can be defined with realm-local scope [RFC7346]. Such a realm-local group is restricted to the local 6LoWPAN network/subnet. In other words, a multicast request to that 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 group (see Section 3.9.1) in order to discover only CoAP servers on the local 6LoWPAN network.

3.9.4. 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.

Because it supports unicast only, [RFC8323] (CoAP over TCP, TLS, and WebSockets) is not in scope as a transport for CoAP group communication.

3.10. Interworking with Other Protocols

3.10.1. MLD/MLDv2/IGMP/IGMPv3

CoAP nodes that are IP hosts (i.e., not IP routers) are generally unaware of the specific IP multicast routing/forwarding protocol being used in their network. When such a host needs to join a specific (CoAP) multicast group, it requires a way to signal to IP multicast routers which IP multicast address(es) it needs to listen to.

The MLDv2 protocol [RFC3810] is the standard IPv6 method to achieve this; therefore, this method SHOULD be used by members of a CoAP group to subscribe to its multicast IPv6 address, on IPv6 networks that support it. 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 that implement 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 IGMPv3 protocol [RFC3376] is the standard IPv4 method to signal multicast group subscriptions. This SHOULD be used by members of a CoAP group to subscribe to its multicast IPv4 address on IPv4 networks.

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

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 3 (Storing mode with multicast support).

In this mode, RPL DAO can be used by a 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 group members.

The same DAO mechanism can be used to convey CoAP group membership information to an edge router (e.g., 6LBR), 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 then learns which IP multicast traffic it needs to pass through from the backbone network into the LLN subnet, and which traffic not. 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.

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 group members.

However, a CoAP multicast request to a 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, which works only in case all CoAP servers joining a group are in link-local communication range of an ingress MPL Forwarder. This is typically not the case on mesh networks.
- * 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 conceptual "NoSec" security group as defined in Section 2.1 is used for unsecured group communication.

It is NOT RECOMMENDED to use CoAP group communication in NoSec mode, even in case of non-sensitive and non-critical applications.

Before possibly and exceptionally using the NoSec mode, 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).

5. Secured Group Communication using Group OSCORE

This section defines 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 [I-D.ietf-cose-rfc8152bis-struct] [I-D.ietf-cose-rfc8152bis-algs] 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, 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 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 group members' authentication credentials including the corresponding public keys; manages, renews and provides security material in the group; and handles the join process of new group members.

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

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.ietf-core-oscore-discovery] and based on the CoRE Resource Directory [I-D.ietf-core-resource-directory].

If security is required, CoAP group communication as described in this specification MUST use Group OSCORE. In particular, a CoAP group as defined in Section 2.1 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 configured to receive CoAP group messages sent to the particular group and -- in case of IP multicast transport -- 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 group member cannot access 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 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, 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.tiloca-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.mattsson-t2trg-amplification-attacks], the NoSec mode is susceptible to source IP address spoofing, hence amplification attacks are especially feasible to perform and greatly effective in their impact, since a single request can result in multiple responses from multiple servers (see Section 6.3).

Therefore, even in case of non-sensitive and non-critical applications, it is generally NOT RECOMMENDED to use CoAP group communication in NoSec mode, also in order to prevent an easy proliferation of high-volume amplification attacks as further discussed in Section 6.3.

Exceptionally, early discovery of devices and resources is a typical use case where NoSec mode is applied. In such a situation, the querying devices do not have yet configured any mutual security relations at the time they perform the discovery. Also, high-volume and harmful amplifications can be prevented through appropriate and conservative configurations, since only a few CoAP servers are expected to be configured for responding to the group requests sent for discovery (see Section 6.3).

CoAP group communication in NoSec mode SHOULD NOT be deployed for sensitive and mission-critical applications (e.g., health monitoring systems and alarm monitoring systems).

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

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

For sensitive and mission-critical applications, CoAP group communication MUST be protected by using Group OSCORE as specified in [I-D.ietf-core-oscore-groupcomm]. The same security considerations from Section 11 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].

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

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

In such a case, the Group Manager may cautiously consider to rekey the 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 paralyzing communications in the group, when a slow rekeying scheme is used and frequently invoked.

At the same, the security implications of delaying the rekeying process have to be carefully considered and understood, before enforcing such 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 group membership change. 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 group is rekeyed, most recently left nodes would preserve access to group communications protected with the retained 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 countersigned 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 by a specific and identified member of the OSCORE group.

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 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].

In fact, upon receiving a group request protected with Group OSCORE in group mode, a server is able to verify whether the request is not a replay and originates from the alleged sender in the OSCORE group, by verifying the signature included in the request using the public key of that sender (see Section 8.2 of [I-D.ietf-core-oscore-groupcomm]). Furthermore, as also discussed in Section 8 of [I-D.ietf-core-oscore-groupcomm], it is recommended that servers failing to decrypt and verify an incoming message do not send back any error message.

This 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 assumed

or 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, especially if this differs from the one used in previous group requests from the same CoAP client. 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-fledged response.

- * Group OSCORE limits the impact of attacks based on IP spoofing also 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 legitimate group members.

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 be originated by the alleged, identifiable sender 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 group members 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 group.

- * Group OSCORE prevents to make any single group member a target for subverting security in the whole OSCORE 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 OSCORE 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 can not be authenticated in some way;
- * SHOULD NOT provide large amplification factors through its responses to a non-authenticated group request, and can possibly rely on CoAP block-wise transfers [RFC7959] to reduce the amount of amplification.

Amplifications attacks using CoAP are further discussed in [I-D.mattsson-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, even in case of non-sensitive and non-critical applications, it is generally NOT RECOMMENDED to use CoAP group communication in NoSec mode, in order to prevent an easy proliferation of high-volume amplification attacks.

Exceptions should be carefully limited to use cases and accesses to a group resource that 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 [I-D.ietf-core-resource-directory], 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 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 every 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 described in Appendix E of [I-D.ietf-core-oscore-groupcomm] and based on the Echo Option for CoAP defined in [RFC9175], 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 much responses from servers to a group request as possible, by using the value '0' - Interested in all responses. This even overrides 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, a multicast IPv6 packet may be fragmented prior to transmission. A 6LoWPAN Router that forwards a fragmented packet may have a relatively high impact on the occupation of the wireless channel and may locally experience high memory load due to packet buffering. For example, the MPL [RFC7731] protocol requires an MPL Forwarder to store the packet for a longer duration, to allow multiple forwarding transmissions to neighboring Forwarders. 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 at a future point in time it needs to receive again all the packet fragments (this time, possibly

from another neighboring MPL Forwarder).

For these reasons, a fragmented IPv6 multicast packet is a possible attack vector in a Denial of Service (DoS) amplification attack. 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 criterium 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, simultaneously turn on all the lights in a room. This intrinsically exposes the group to some unique monitoring risks that devices not in a group are not as vulnerable to. For example, assume an attacker is 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 group some network topology information may be deduced).

6.8.2. Pervasive Monitoring

A key additional threat consideration for group communication is pervasive monitoring [RFC7258]. CoAP group communication solutions that are built on top of IP multicast need to pay particular heed to these dangers. This is because IP multicast is easier to intercept compared to IP unicast. Also, CoAP traffic is typically used for the Internet of Things. This means that CoAP (multicast) group communication may be used for the control and monitoring of critical infrastructure (e.g., lights, alarms, HVAC, electrical grid, etc.) that may be prime targets for attack.

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 example, the source node (controller) sends out CoAP group communication messages which easily identifies it as a controller. CoAP multicast traffic is inherently more vulnerable compared to unicast, as the same packet may be replicated over many more links, leading to a higher probability of packet capture by a pervasive monitoring system.

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 this fulfills the application needs.

Even if all CoAP multicast traffic is encrypted/protected, 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 J. Park, "Group OSCORE - Secure Group Communication
for CoAP", Work in Progress, Internet-Draft, draft-ietf-

core-oscore-groupcomm-14, 7 March 2022,
<<https://www.ietf.org/archive/id/draft-ietf-core-oscore-groupcomm-14.txt>>.

[I-D.ietf-cose-rfc8152bis-algs]

Schaad, J., "CBOR Object Signing and Encryption (COSE): Initial Algorithms", Work in Progress, Internet-Draft, draft-ietf-cose-rfc8152bis-algs-12, 24 September 2020, <<https://www.ietf.org/archive/id/draft-ietf-cose-rfc8152bis-algs-12.txt>>.

[I-D.ietf-cose-rfc8152bis-struct]

Schaad, J., "CBOR Object Signing and Encryption (COSE): Structures and Process", Work in Progress, Internet-Draft, draft-ietf-cose-rfc8152bis-struct-15, 1 February 2021, <<https://www.ietf.org/archive/id/draft-ietf-cose-rfc8152bis-struct-15.txt>>.

[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/info/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/info/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/info/rfc3376>>.

[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/info/rfc3810>>.

[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/info/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/info/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/info/rfc6282>>.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <<https://www.rfc-editor.org/info/rfc6690>>.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<https://www.rfc-editor.org/info/rfc7252>>.
- [RFC7641] Hartke, K., "Observing Resources in the Constrained Application Protocol (CoAP)", RFC 7641, DOI 10.17487/RFC7641, September 2015, <<https://www.rfc-editor.org/info/rfc7641>>.
- [RFC7959] Bormann, C. and Z. Shelby, Ed., "Block-Wise Transfers in the Constrained Application Protocol (CoAP)", RFC 7959, DOI 10.17487/RFC7959, August 2016, <<https://www.rfc-editor.org/info/rfc7959>>.
- [RFC8075] Castellani, A., Loreto, S., Rahman, A., Fossati, T., and E. Dijk, "Guidelines for Mapping Implementations: HTTP to the Constrained Application Protocol (CoAP)", RFC 8075, DOI 10.17487/RFC8075, February 2017, <<https://www.rfc-editor.org/info/rfc8075>>.
- [RFC8132] van der Stok, P., Bormann, C., and A. Sehgal, "PATCH and FETCH Methods for the Constrained Application Protocol (CoAP)", RFC 8132, DOI 10.17487/RFC8132, April 2017, <<https://www.rfc-editor.org/info/rfc8132>>.
- [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>>.

- [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/info/rfc9175>>.

8.2. Informative References

- [Californium]
Eclipse Foundation, "Eclipse Californium", March 2019, <<https://github.com/eclipse/californium/tree/2.0.x/californium-core/src/main/java/org/eclipse/californium/core>>.
- [Go-OCF] Open Connectivity Foundation (OCF), "Implementation of CoAP Server & Client in Go", March 2019, <<https://github.com/go-ocf/go-coap>>.
- [I-D.ietf-ace-key-groupcomm-oscore]
Tiloca, M., Park, J., and F. Palombini, "Key Management for OSCORE Groups in ACE", Work in Progress, Internet-Draft, draft-ietf-ace-key-groupcomm-oscore-13, 7 March 2022, <<https://www.ietf.org/archive/id/draft-ietf-ace-key-groupcomm-oscore-13.txt>>.
- [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-46, 8 November 2021, <<https://www.ietf.org/archive/id/draft-ietf-ace-oauth-authz-46.txt>>.
- [I-D.ietf-ace-oscore-gm-admin]
Tiloca, M., Höglund, R., Stok, P. V. D., and F. Palombini, "Admin Interface for the OSCORE Group Manager", Work in Progress, Internet-Draft, draft-ietf-ace-oscore-gm-admin-05, 7 March 2022, <<https://www.ietf.org/archive/id/draft-ietf-ace-oscore-gm-admin-05.txt>>.
- [I-D.ietf-core-coap-pubsub]
Koster, M., Keranen, A., and J. Jimenez, "Publish-Subscribe Broker for the Constrained Application Protocol (CoAP)", Work in Progress, Internet-Draft, draft-ietf-core-coap-pubsub-09, 30 September 2019, <<https://www.ietf.org/archive/id/draft-ietf-core-coap-pubsub-09.txt>>.

[I-D.ietf-core-new-block]

Boucadair, M. and J. Shallow, "Constrained Application Protocol (CoAP) Block-Wise Transfer Options Supporting Robust Transmission", Work in Progress, Internet-Draft, draft-ietf-core-new-block-14, 26 May 2021, <<https://www.ietf.org/archive/id/draft-ietf-core-new-block-14.txt>>.

[I-D.ietf-core-resource-directory]

Amsüss, C., Shelby, Z., Kostner, 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://www.ietf.org/archive/id/draft-ietf-core-resource-directory-28.txt>>.

[I-D.mattsson-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-mattsson-t2trg-amplification-attacks-00, 11 February 2022, <<https://www.ietf.org/archive/id/draft-mattsson-t2trg-amplification-attacks-00.txt>>.

[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-06, 7 March 2022, <<https://www.ietf.org/archive/id/draft-tiloca-core-groupcomm-proxy-06.txt>>.

[I-D.tiloca-core-oscore-discovery]

Tiloca, M., Amsuess, C., and P. V. D. Stok, "Discovery of OSCORE Groups with the CoRE Resource Directory", Work in Progress, Internet-Draft, draft-tiloca-core-oscore-discovery-11, 7 March 2022, <<https://www.ietf.org/archive/id/draft-tiloca-core-oscore-discovery-11.txt>>.

[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/info/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/info/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/info/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/info/rfc7258>>.
- [RFC7346] Droms, R., "IPv6 Multicast Address Scopes", RFC 7346, DOI 10.17487/RFC7346, August 2014, <<https://www.rfc-editor.org/info/rfc7346>>.
- [RFC7390] Rahman, A., Ed. and E. Dijk, Ed., "Group Communication for the Constrained Application Protocol (CoAP)", RFC 7390, DOI 10.17487/RFC7390, October 2014, <<https://www.rfc-editor.org/info/rfc7390>>.
- [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/info/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/info/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/info/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/info/rfc8710>>.

Appendix A. Use Cases

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

A.1. Discovery

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

A.1.1. Distributed Device Discovery

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

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

A.1.2. Distributed Service Discovery

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

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

A.1.3. Directory Discovery

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

A.2. Operational Phase

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

A.2.1. Actuator Group Control

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

A.2.2. Device Group Status Request

To properly monitor the status of systems, there may be a need for ad-hoc, unplanned status updates. Group communication can be used to quickly send out a request to a (potentially large) number of devices for specific information. Each device then responds back with the requested data. Those devices that did not respond to the request can optionally be polled again via reliable unicast communication to complete the dataset. The device group may be defined e.g., as "all temperature sensors on floor 3", or "all lights in wing B". For example, it could be a status request for device temperature, most recent sensor event detected, firmware version, network load, and/or battery level.

A.2.3. Network-wide Query

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

A.2.4. Network-wide / Group Notification

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

A.3. Software Update

Group communication can be useful to efficiently distribute new software (firmware, image, application, etc.) to a group of multiple devices. In this case, the group is defined in terms of device type: all devices in the target group are known to be capable of installing and running the new software. The software is distributed as a series of smaller blocks that are collected by all devices and stored in memory. All devices in the target group are usually responsible

for integrity verification of the received software; which can be done per-block or for the entire software image once all blocks have been received. Due to the inherent unreliability of CoAP multicast, there needs to be a backup mechanism (e.g., implemented using CoAP unicast) by which a device can individually request missing blocks of a whole software image/entity. Prior to a multicast software update, the group of recipients can be separately notified that there is new software available and coming, using the above network-wide or group notification.

Appendix B. 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.

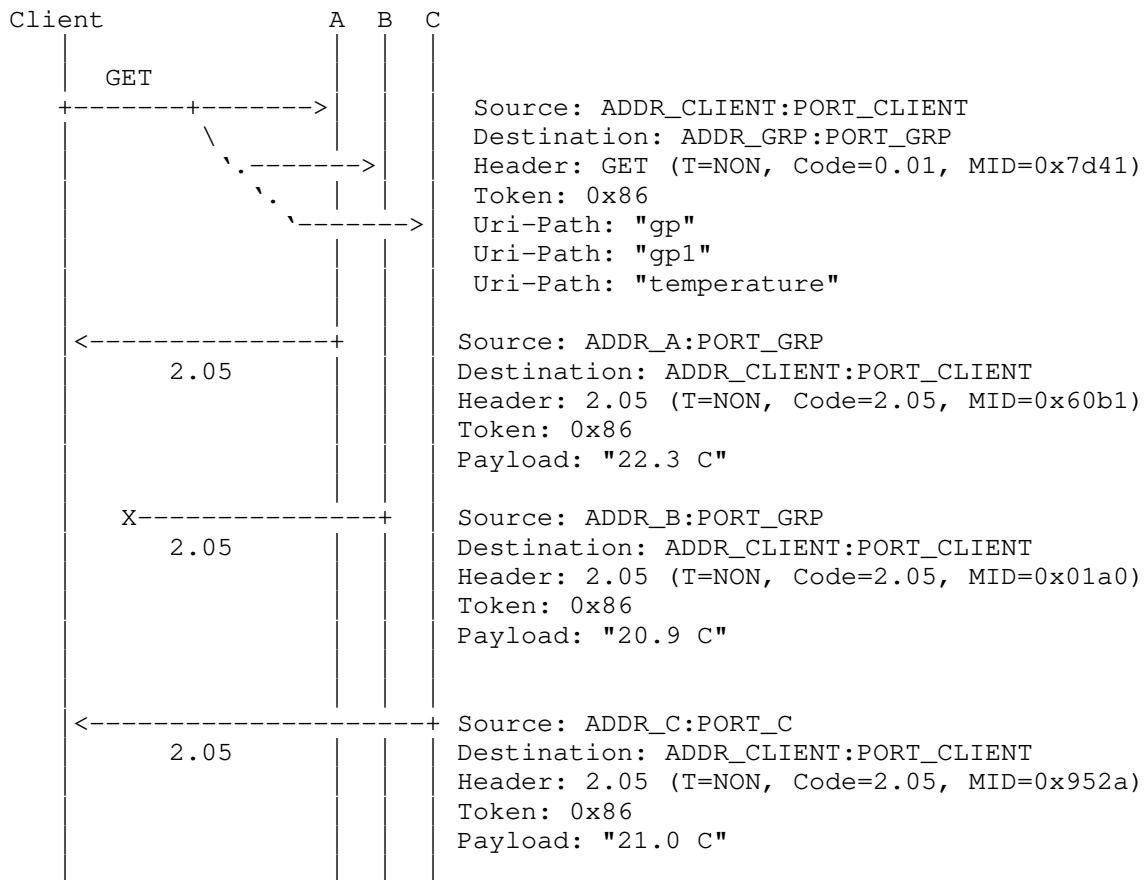
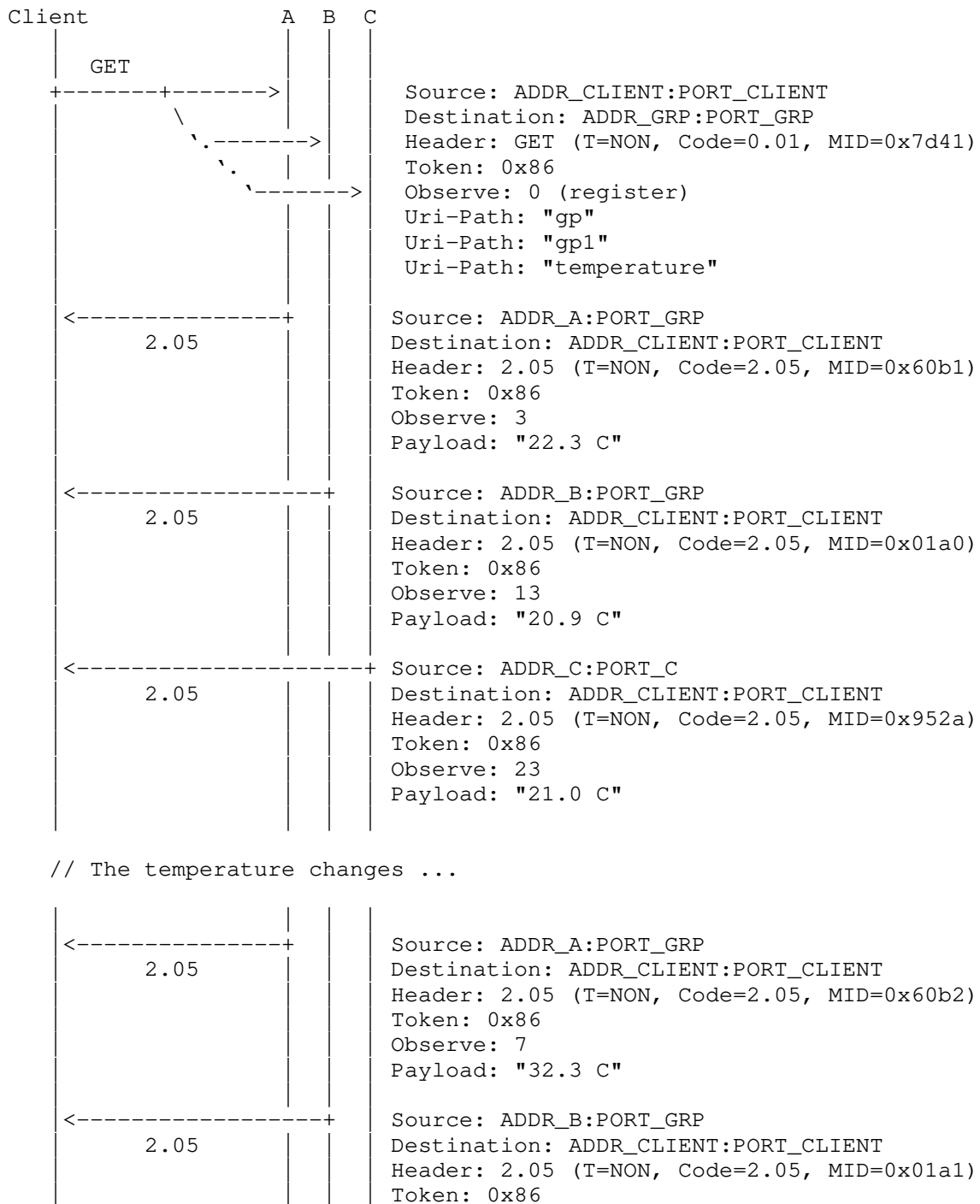


Figure 16: Example of Non-confirmable group request, followed by Non-confirmable Responses

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 payload the new representation of the "temperature" resource.



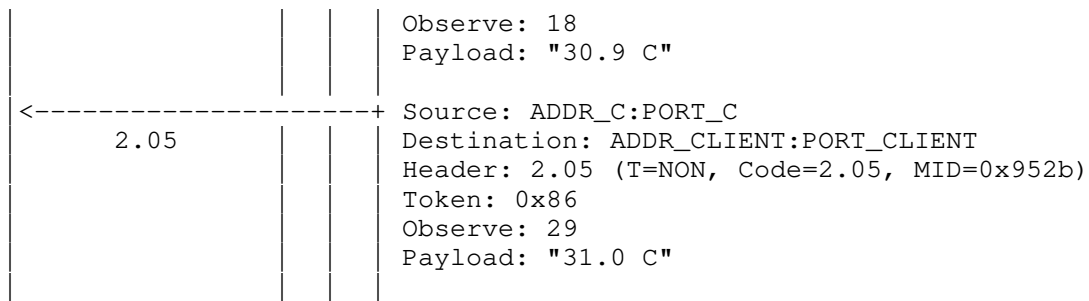
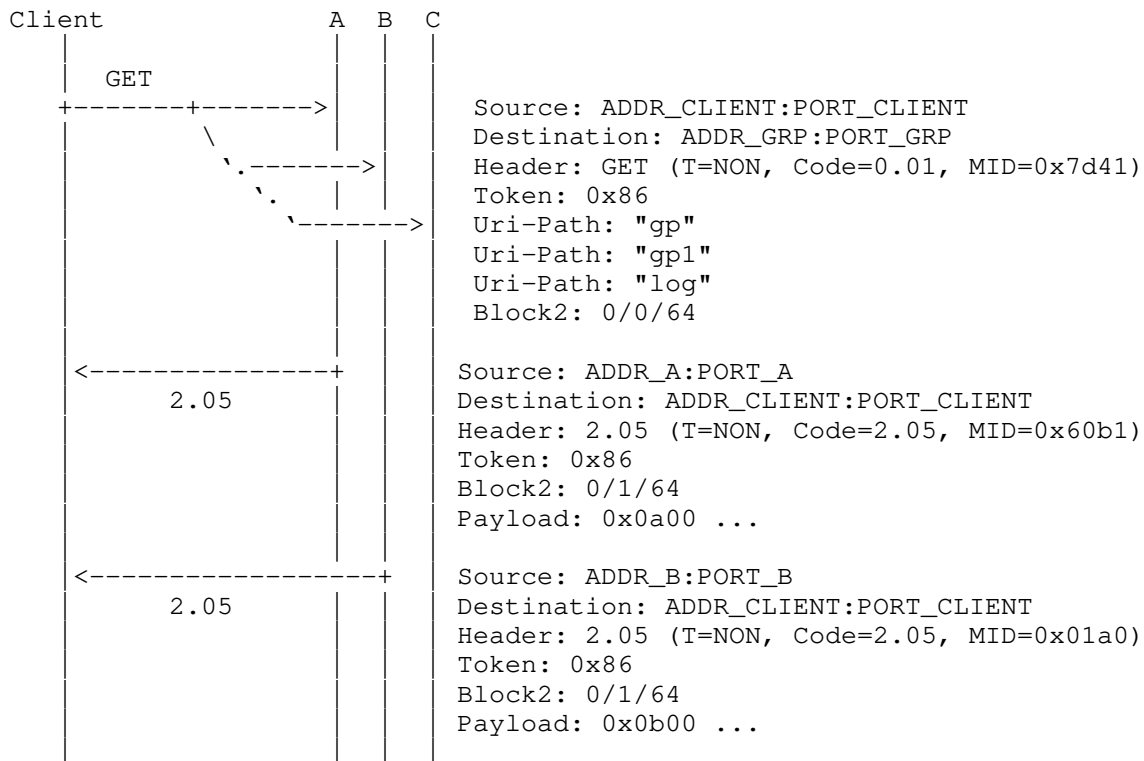
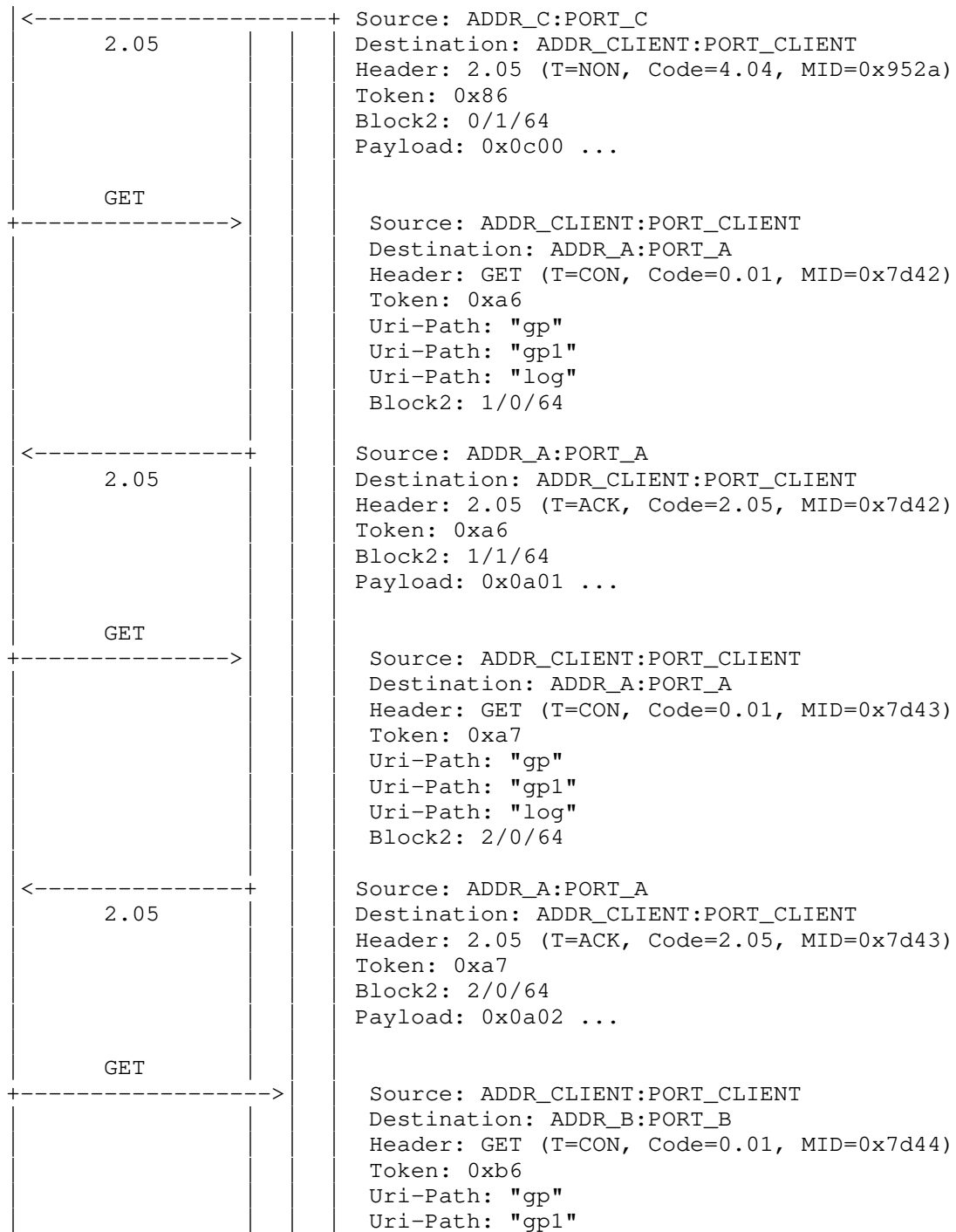
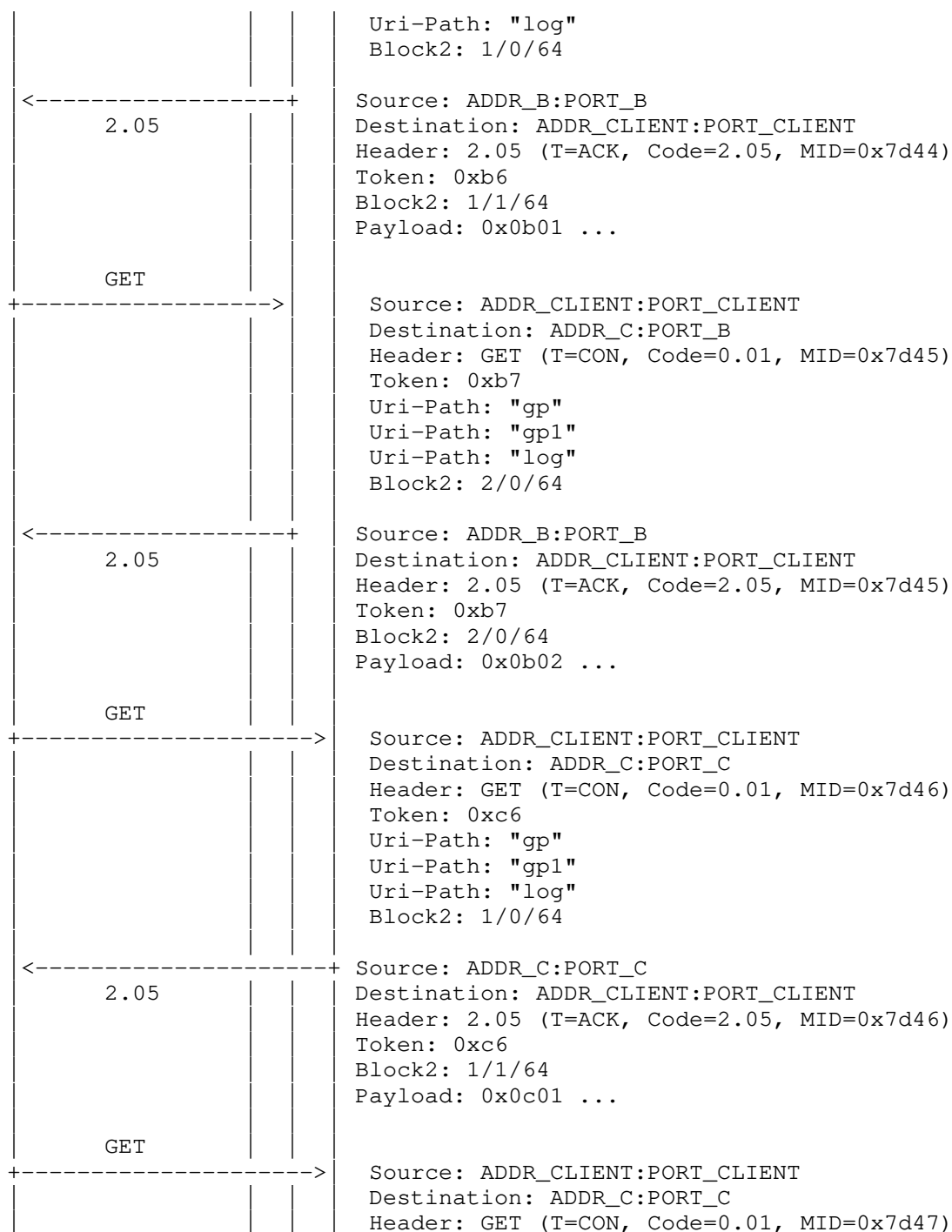


Figure 17: Example of Non-confirmable Observe group request, followed by Non- confirmable Responses as Observe notifications

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.







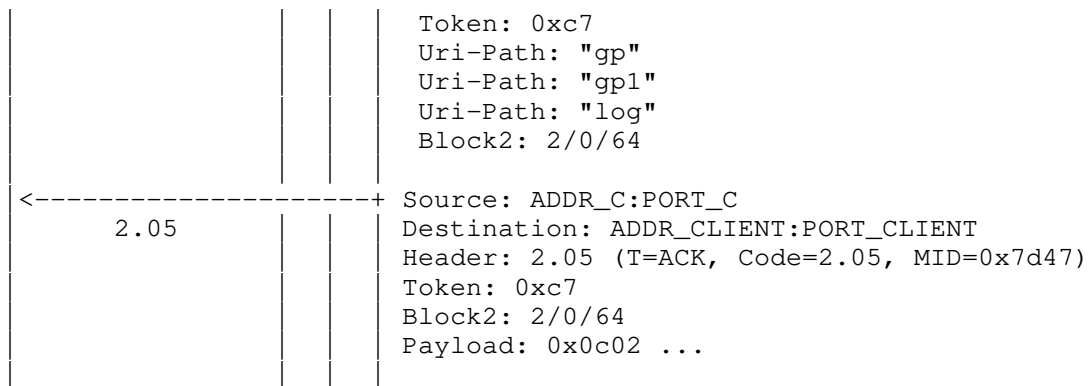


Figure 18: Example of Non-confirmable group request starting a blockwise transfer, followed by Non-confirmable Responses with the first block. The transfer continues over confirmable unicast exchanges

Appendix C. Document Updates

RFC EDITOR: PLEASE REMOVE THIS SECTION.

C.1. 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.

C.2. 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.

C.3. 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.

C.4. 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.

C.5. 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.

C.6. Version -00 to -01

- * Clarifications on group memberships for the different group types.
- * Simplified description of Token reuse, 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 Amsuess, Carsten Bormann, Thomas Fossati, Rikard Hoeglund, Jaime Jimenez, John Mattsson and Jim Schaad for their comments and feedback.

The work on this document has been partly supported by VINNOVA and the Celtic-Next project CRITISEC; and by the H2020 project SIFIS-Home (Grant agreement 952652).

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
Intended status: Standards Track
Expires: 8 September 2022

C. Bormann, Ed.
Universität Bremen TZI
H. Birkholz
Fraunhofer SIT
7 March 2022

Constrained Resource Identifiers
draft-ietf-core-href-10

Abstract

The Constrained Resource Identifier (CRI) is a complement to the Uniform Resource Identifier (URI) that serializes the URI components in Concise Binary Object Representation (CBOR) instead of a sequence of characters. This simplifies parsing, comparison and reference resolution in environments with severe limitations on processing power, code size, and memory size.

The present revision -10 of this draft contains an experimental addition that allows representing user information (<https://alice@chains.example>) in the URI authority component. This feature lacks test vectors and implementation experience at the time of writing and requires discussion.

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/>.

Source for this draft and an issue tracker can be found at <https://github.com/core-wg/href>.

Status of This Memo

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

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

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

This Internet-Draft will expire on 8 September 2022.

Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	3
1.1. Notational Conventions	4
2. Constraints	4
2.1. Constraints not expressed by the data model	6
3. Creation and Normalization	7
4. Comparison	8
5. CRI References	8
5.1. CBOR Serialization	9
5.1.1. The discard Section	11
5.1.2. Visualization	11
5.1.3. Examples	11
5.1.4. Specific Terminology	12
5.2. Ingesting and encoding a CRI Reference	12
5.3. Reference Resolution	13
6. Relationship between CRIs, URIs and IRIs	14
6.1. Converting CRIs to URIs	15
7. Extended CRI: Accommodating Percent Encoding (PET)	17
8. Implementation Status	18
9. Security Considerations	19
10. IANA Considerations	19
11. References	19
11.1. Normative References	19
11.2. Informative References	20
Appendix A. The Small Print	21
Appendix B. Change Log	22

Acknowledgements	25
Contributors	25
Authors' Addresses	25

1. Introduction

The Uniform Resource Identifier (URI) [RFC3986] 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 RFC 3986.

This document defines the `_Constrained Resource Identifier (CRI)_` by constraining URIs to a simplified subset and serializing their components in Concise Binary Object Representation (CBOR) [RFC8949] 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 RFC 3986 (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) [RFC7230], 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 [RFC2119] [RFC8174] 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 (rendered in plain text as the new term surrounded by underscores).

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 [RFC3986], which constrains schemes 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 [RFC3986] are modeled by two different values used in place of an absent authority:

- * the path can begin with a root ("/", as when the authority is present), or
- * the path can be rootless.

(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 [RFC3986]) or "sub-delims" (Section 2.2 of [RFC3986]); any other character needs to be percent-encoded (Section 7). 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 [RFC3986], 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 [RFC3986]), 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 [RFC3986], not on the scheme-specific Scheme-Based Normalization level (Section 6.2.3 of [RFC3986])).

(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 A.)

- 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 A.)

- 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 A.

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. (An example of this is [0, ["p"]] which appends a slash and the path segment "p" to its base).

(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 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 [RFC3986], 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.

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 [RFC8820]). 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 (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 serialization of CRIs in Concise Binary Object Representation (CBOR) [RFC8949]. To reduce representation size, CRIs are not serialized directly. Instead, CRIs are indirectly referenced through `_CRI references_`. These take advantage of hierarchical locality and provide a very compact encoding. The CBOR serialization 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 unspecified; 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 Serialization

A CRI or CRI reference is encoded as a CBOR array [RFC8949], with the structure as described in the Concise Data Definition Language (CDDL) [RFC8610] as follows:

```
// RFC Ed.: throughout this section, please replace RFC-XXXX with the
// RFC number of this specification and remove this note.
```

```
; not expressed in this CDDL spec: trailing nulls to be left off
```

```
RFC-XXXX-Definitions = [CRI, CRI-Reference]
```

```
CRI = [
    scheme,
    authority / no-authority,
    local-part
]
```

```
CRI-Reference = [
    ((scheme / null, authority / no-authority)
    // discard),                ; relative reference
    local-part
]
```

```

local-part = (
  path / null,
  query / null,
  fragment / null
)

scheme      = scheme-name / scheme-id
scheme-name = text .regexp "[a-z][a-z0-9+.-]*"
scheme-id   = (COAP / COAPS / HTTP / HTTPS / URN / DID /
               other-scheme)
               .within nint
COAP = -1 COAPS = -2 HTTP = -3 HTTPS = -4 URN = -5 DID = -6
other-scheme = nint .feature "scheme-id-extension"

no-authority = NOAUTH-NOSLASH / NOAUTH-LEADINGSLASH
NOAUTH-LEADINGSLASH = null .feature "no-authority"
NOAUTH-NOSLASH = 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 serialization

This CDDL specification is simplified for exposition and needs to be augmented by the following rule for interchange of CRIs and CRI references: Trailing null values MUST be removed, and two leading null values (scheme and authority both not given) are represented by using the discard alternative instead.

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.

5.1.1. 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 [RFC3986] calls "relative references" (i.e., references that need to undergo a resolution process to obtain a URI) correspond to the CRI form that starts with discard. The exception are relative references with an authority (called a "network-path reference" in Section 4.2 of [RFC3986]), which discard the entire path of the base CRI. These CRI references never carry a discard section: the value of discard defaults to true.

5.1.2. Visualization

The structure of a CRI reference is visualized using the somewhat limited means of a railroad diagram:

```
cri-reference:
>
```

```
>
```

```
>
```

```
>
```

```
  scheme authority path query fragment
```

```
    discard
```

This visualization does not go into the details of the elements.

5.1.3. Examples

```
[ -1,          / scheme -- equivalent to "coap" /
  [h'C6336401', / host /
    61616],    / port /
  [".well-known", / path /
    "core"]
]

[true,          / discard /
  [".well-known", / path /
    "core"],
  ["rt=temperature-c"]] / query /

[ -6,          / scheme -- equivalent to "did" /
  true,        / authority = NOAUTH-NOSLASH /
  ["web:alice:bob"] / path /
]
```

5.1.4. 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 number 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 number), 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 "").

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 number, 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 fragment in the buffer if query is non-null in the CRI reference (and therefore has been copied to the buffer).

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) [RFC3986] 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.

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 [RFC3986].

scheme

If the CRI reference contains a scheme section, the scheme component of the URI reference consists of the value of that section. 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 [RFC3986]) or "sub-delims" (Section 2.2 of [RFC3986]) 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 a dot ("."), or not in the set of unreserved characters (Section 2.3 of [RFC3986]) or "sub-delims" (Section 2.2 of [RFC3986]) MUST be percent-encoded.

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 [RFC3986]). Any zone-id is appended to the string, separated by "%25" as defined in Section 2 of [RFC6874], or as specified in a superseding zone-id specification document [I-D.carpenter-6man-rfc6874bis]; 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 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 [RFC3986]), 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 [RFC3986]), 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 [RFC3986]), 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. Extended CRI: Accommodating Percent Encoding (PET)

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.

Basic CRI does not support URI components that require percent-encoding (Section 2.1 of [RFC3986]) 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`

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"}]]
```

8. Implementation Status

With the exception of the authority=true fix, host-names split into labels, and Section 7, CRIs are implemented in <https://gitlab.com/chrysn/micrurus>. A go-lang implementation of version -10 of this document is found at: <https://github.com/thomas-fossati/href>

9. 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 [RFC8949] for additional security considerations relating to CBOR.

The security considerations discussed in Section 7 of [RFC3986] and Section 8 of [RFC3987] for URIs and IRIs also apply to CRIs.

10. IANA Considerations

This document has no IANA actions.

11. References

11.1. Normative References

- [I-D.carpenter-6man-rfc6874bis]
Carpenter, B., 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://www.ietf.org/archive/id/draft-carpenter-6man-rfc6874bis-03.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>>.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform
Resource Identifier (URI): Generic Syntax", STD 66,
RFC 3986, DOI 10.17487/RFC3986, January 2005,
<<https://www.rfc-editor.org/info/rfc3986>>.
- [RFC3987] Duerst, M. and M. Suignard, "Internationalized Resource
Identifiers (IRIs)", RFC 3987, DOI 10.17487/RFC3987,
January 2005, <<https://www.rfc-editor.org/info/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/info/rfc6874>>.

- [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>>.
- [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/info/rfc8610>>.
- [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>>.
- [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/>>.

11.2. Informative References

- [RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, DOI 10.17487/RFC7228, May 2014, <<https://www.rfc-editor.org/info/rfc7228>>.
- [RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <<https://www.rfc-editor.org/info/rfc7230>>.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<https://www.rfc-editor.org/info/rfc7252>>.
- [RFC8141] Saint-Andre, P. and J. Klensin, "Uniform Resource Names (URNs)", RFC 8141, DOI 10.17487/RFC8141, April 2017, <<https://www.rfc-editor.org/info/rfc8141>>.
- [RFC8288] Nottingham, M., "Web Linking", RFC 8288, DOI 10.17487/RFC8288, October 2017, <<https://www.rfc-editor.org/info/rfc8288>>.
- [RFC8820] Nottingham, M., "URI Design and Ownership", BCP 190, RFC 8820, DOI 10.17487/RFC8820, June 2020, <<https://www.rfc-editor.org/info/rfc8820>>.

[W3C.REC-html52-20171214]

Faulkner, S., Eicholz, A., Leithead, T., Danilo, A., and
S. Moon, "HTML 5.2", World Wide Web Consortium
Recommendation REC-html52-20171214, 14 December 2017,
<<https://www.w3.org/TR/2017/REC-html52-20171214>>.

Appendix A. 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 [RFC3986], `s://x` is distinct from `s://x/` -- i.e., a URI with an empty path is different from one with a lone empty path segment. However, in HTTP, 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.
// TBD: add similar text for HTTP, if that can be made. Section 6.2.3 of [RFC3986] even 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) 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 B. Change Log

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

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 [RFC3986]). 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 A, 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

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: 8 September 2022

M. Tiloca
RISE AB
G. Selander
F. Palombini
J. Mattsson
Ericsson AB
J. Park
Universitaet Duisburg-Essen
7 March 2022

Group OSCORE - Secure Group Communication for CoAP
draft-ietf-core-oscore-groupcomm-14

Abstract

This document defines 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 approach 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.

Status of This Memo

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

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

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

This Internet-Draft will expire on 8 September 2022.

Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	4
1.1. Terminology	6
2. Security Context	8
2.1. Common Context	11
2.1.1. AEAD Algorithm	11
2.1.2. ID Context	11
2.1.3. Group Manager Authentication Credential	12
2.1.4. Signature Encryption Algorithm	12
2.1.5. Signature Algorithm	12
2.1.6. Group Encryption Key	12
2.1.7. Pairwise Key Agreement Algorithm	13
2.2. Sender Context and Recipient Context	13
2.3. Authentication Credentials	14
2.4. Pairwise Keys	16
2.4.1. Derivation of Pairwise Keys	16
2.4.2. ECDH with Montgomery Coordinates	18
2.4.3. Usage of Sequence Numbers	19
2.4.4. Security Context for Pairwise Mode	20
2.5. Update of Security Context	20
2.5.1. Loss of Mutable Security Context	21
2.5.2. Exhaustion of Sender Sequence Number	22
2.5.3. Retrieving New Security Context Parameters	23
3. The Group Manager	25
3.1. Support for Additional Entities	26
3.2. Management of Group Keying Material	27
3.2.1. Recycling of Identifiers	30
3.3. Responsibilities of the Group Manager	32
4. The COSE Object	34
4.1. Countersignature	34
4.1.1. Keystream Derivation	35
4.1.2. Clarifications on Using a Countersignature	36
4.2. The 'kid' and 'kid context' parameters	36
4.3. external_aad	36
5. OSCORE Header Compression	39
5.1. Examples of Compressed COSE Objects	40
5.1.1. Examples in Group Mode	40
5.1.2. Examples in Pairwise Mode	41

6.	Message Binding, Sequence Numbers, Freshness and Replay Protection	42
6.1.	Supporting Observe	42
6.2.	Update of Replay Window	42
6.3.	Message Freshness	43
7.	Message Reception	43
8.	Message Processing in Group Mode	44
8.1.	Protecting the Request	46
8.1.1.	Supporting Observe	46
8.2.	Verifying the Request	47
8.2.1.	Supporting Observe	49
8.3.	Protecting the Response	49
8.3.1.	Supporting Observe	50
8.4.	Verifying the Response	51
8.4.1.	Supporting Observe	53
8.5.	External Signature Checkers	54
9.	Message Processing in Pairwise Mode	55
9.1.	Pre-Conditions	56
9.2.	Main Differences from OSCORE	56
9.3.	Protecting the Request	57
9.4.	Verifying the Request	57
9.5.	Protecting the Response	57
9.6.	Verifying the Response	58
10.	Challenge-Response Synchronization	59
11.	Implementation Compliance	62
12.	Security Considerations	63
12.1.	Security of the Group Mode	64
12.2.	Security of the Pairwise Mode	66
12.3.	Uniqueness of (key, nonce)	67
12.4.	Management of Group Keying Material	67
12.5.	Update of Security Context and Key Rotation	68
12.5.1.	Late Update on the Sender	68
12.5.2.	Late Update on the Recipient	69
12.6.	Collision of Group Identifiers	69
12.7.	Cross-group Message Injection	70
12.7.1.	Attack Description	70
12.7.2.	Attack Prevention in Group Mode	71
12.8.	Prevention of Group Cloning Attack	72
12.9.	Group OSCORE for Unicast Requests	73
12.10.	End-to-end Protection	74
12.11.	Master Secret	74
12.12.	Replay Protection	75
12.13.	Message Freshness	75
12.14.	Client Aliveness	75
12.15.	Cryptographic Considerations	76
12.16.	Message Segmentation	77
12.17.	Privacy Considerations	78
13.	IANA Considerations	79

13.1. OSCORE Flag Bits Registry	79
14. References	79
14.1. Normative References	79
14.2. Informative References	81
Appendix A. Assumptions and Security Objectives	84
A.1. Assumptions	85
A.2. Security Objectives	86
Appendix B. List of Use Cases	87
Appendix C. Example of Group Identifier Format	90
Appendix D. Set-up of New Endpoints	91
Appendix E. Document Updates	91
E.1. Version -13 to -14	91
E.2. Version -12 to -13	92
E.3. Version -11 to -12	92
E.4. Version -10 to -11	93
E.5. Version -09 to -10	94
E.6. Version -08 to -09	95
E.7. Version -07 to -08	95
E.8. Version -06 to -07	97
E.9. Version -05 to -06	97
E.10. Version -04 to -05	98
E.11. Version -03 to -04	98
E.12. Version -02 to -03	99
E.13. Version -01 to -02	100
E.14. Version -00 to -01	101
Acknowledgments	102
Authors' Addresses	102

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) [I-D.ietf-cose-rfc8152bis-struct] [I-D.ietf-cose-rfc8152bis-algs] 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 in a COSE object, which replaces the authenticated and encrypted fields in the protected message.

This document defines Group OSCORE, a security protocol for Group communication for CoAP [I-D.ietf-core-groupcomm-bis], providing the same end-to-end security properties as OSCORE in the case where CoAP requests have multiple recipients. In particular, the described approach 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. 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 [RFC6347][I-D.ietf-tls-dtls13], 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 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 with the group mode, and responses with the 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 is to use 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 is to deploy Group OSCORE, with the endpoints being part of a group, and use the pairwise mode. This solution assumes a trusted third party called Group Manager (see Section 3). However, it 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-lwig-security-protocol-comparison].

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]; CBOR [RFC8949]; COSE [I-D.ietf-cose-rfc8152bis-struct][I-D.ietf-cose-rfc8152bis-algs] and related countersignatures [I-D.ietf-cose-countersign].

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: set of 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 (CCSs) [RFC8392], X.509 certificates [RFC7925] and C509 certificates [I-D.ietf-cose-cbor-encoded-cert]. Further details about authentication credentials are provided in Section 2.3.
- * 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: entity responsible for a group. Each endpoint in a group communicates securely with the respective Group Manager, which is neither required to be an actual group member nor to take part in the group communication. The full list of responsibilities of the Group Manager is provided in Section 3.3.

- * **Silent server:** member of a group that never sends protected responses in reply to requests. For CoAP group communications, requests are normally sent without necessarily expecting a response. A silent server may send unprotected responses, as error responses reporting an OSCORE error. Note that an endpoint can implement both a silent server and a client, i.e., the two roles are independent. An endpoint acting only as a silent server performs only Group OSCORE processing on incoming requests. Silent servers maintain less keying material and in particular do not have a Sender Context for the group. Since silent servers do not have a Sender ID, they cannot support the pairwise mode.
- * **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.
- * **Group request:** CoAP request message sent by a client in the group to all servers in that 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.

2. Security Context

As per the terminology in Section 1.1, this document refers to a group as a set of endpoints sharing keying material and security parameters for executing the Group OSCORE protocol. Each endpoint of a group is aware of whether the group uses the group mode, or the pairwise mode, or both. Then, an endpoint can use any mode it supports if also used in the group.

All members of a group maintain a Security Context as defined in Section 3 of [RFC8613] and extended 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.

The default setting for how to manage information about the group, including the Security Context, is described in terms of a Group Manager (see Section 3). In particular, the Group Manager indicates whether the group uses the group mode, the pairwise mode, or both of them, as part of the group data provided to candidate group members when joining the group.

The remainder of this section provides further details about the Security Context of Group OSCORE. In particular, each endpoint which is member of a group maintains a Security Context as defined in Section 3 of [RFC8613], extended as follows (see Figure 1).

- * One Common Context, shared by all the endpoints in the group. Several new parameters are included in the Common Context.

If a Group Manager is used for maintaining the group, the Common Context is extended with the authentication credential of the Group Manager, including the Group Manager's public key. When processing messages, the authentication credential of the Group Manager is included in the external additional authenticated data (see Section 4.3).

If the group uses the group mode, the Common context is extended with the following new parameters.

- Signature Encryption Algorithm and Signature Algorithm. These relate to the encryption/decryption operations and to the computation/verification of countersignatures, respectively, when a message is protected with the group mode (see Section 8).
- Group Encryption Key, used to perform encryption/decryption of countersignatures, when a message is protected with the group mode (see Section 8).

If the group uses the pairwise mode, the Common Context is extended with a Pairwise Key Agreement Algorithm used for agreement on a static-static Diffie-Hellman shared secret, from which pairwise keys are derived (see Section 2.4.1) to protect messages with the pairwise mode (see Section 9).

- * One Sender Context, extended with the endpoint's private key and authentication credential including the endpoint's public key.

The private key is used to sign messages protected with the group mode, or for deriving pairwise keys in pairwise mode (see Section 2.4). The authentication credential is used for deriving pairwise keys in pairwise mode, and is included in the external additional authenticated data when processing outgoing messages (see Section 9).

If the endpoint supports the pairwise mode, the Sender Context is also extended with the Pairwise Sender Keys associated with the other endpoints (see Section 2.4).

The Sender Context is omitted if the endpoint is configured exclusively as silent server.

- * 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. The Recipient Context is extended with the authentication credential of the other endpoint, including that endpoint's public key.

The public key is used to verify the signature of messages protected with the group mode from the other endpoint and for deriving the pairwise keys in pairwise mode (see Section 2.4). The authentication credential is used for deriving pairwise keys in pairwise mode, and is included in the external additional authenticated data when processing incoming messages from the other endpoint (see Section 9).

If the endpoint supports the pairwise mode, then the Recipient Context is also extended with the Pairwise Recipient Key associated with the other endpoint (see Section 2.4).

Context Component	New Information Elements
Common Context	Group Manager Authentication Credential * Signature Encryption Algorithm * Signature Algorithm * Group 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 optional elements labeled with * (with ^) are present only if the group uses the group mode (the pairwise mode).

2.1. Common Context

The Common Context may be acquired from the Group Manager (see Section 3). The following sections define how the Common Context is extended, compared to [RFC8613].

2.1.1. AEAD Algorithm

AEAD Algorithm identifies the COSE AEAD algorithm to use for encryption, when messages are protected using the pairwise mode (see Section 9). This algorithm MUST provide integrity protection. This parameter is immutable once the Common Context is established, and it is not relevant if the group uses only the group mode.

2.1.2. ID Context

The ID Context parameter (see Sections 3.1 and 3.3 of [RFC8613]) in the Common Context 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 12.6).

2.1.3. Group Manager Authentication Credential

Group Manager Authentication Credential specifies the authentication credential of the Group Manager, including the Group Manager's public key. This is included in the external additional authenticated data when processing messages (see Section 4.3).

Each group member MUST obtain the authentication credential of the Group Manager with a valid proof-of-possession of the corresponding private key, for instance from the Group Manager itself when joining the group. Further details on the provisioning of the Group Manager's authentication credential to the group members are out of the scope of this document.

2.1.4. Signature Encryption Algorithm

Signature Encryption Algorithm identifies the algorithm to use for encryption, when messages are protected using the group mode (see Section 8). This algorithm MAY provide integrity protection. This parameter is immutable once the Common Context is established.

This algorithm is not used to encrypt the countersignature in messages protected using the group mode, for which the method defined in Section 4.1 is used.

2.1.5. Signature Algorithm

Signature Algorithm identifies the digital signature algorithm used to compute a countersignature on the COSE object (see Sections 3.2 and 3.3 of [I-D.ietf-cose-countersign]), when messages are protected using the group mode (see Section 8). This parameter is immutable once the Common Context is established.

2.1.6. Group Encryption Key

Group Encryption Key specifies the encryption key for deriving a keystream to encrypt/decrypt a countersignature, when a message is protected with the group mode (see Section 8).

The Group 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 takes the value of Signature Encryption Algorithm from the Common Context (see Section 2.1.5).

- * The 'type' element of the 'info' array is "Group Encryption Key". 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 Signature Encryption Algorithm from the Common Context (see Section 2.1.5), in bytes.

2.1.7. Pairwise Key Agreement Algorithm

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 2.4.1) to protect messages with the pairwise mode (see Section 9). This parameter is immutable once the Common Context is established.

2.2. Sender Context and Recipient Context

OSCORE specifies the derivation of Sender Context and Recipient Context, specifically of Sender/Recipient Keys and Common IV, from a set of input parameters (see Section 3.2 of [RFC8613]).

The derivation of Sender/Recipient Keys and Common IV defined in OSCORE applies also to Group OSCORE, with the following extensions compared to Section 3.2.1 of [RFC8613].

- * If the group uses (also) the group mode, the 'alg_aead' element of the 'info' array takes the value of Signature Encryption Algorithm from the Common Context (see Section 2.1.5).
- * If the group uses only the pairwise mode, the 'alg_aead' element of the 'info' array takes the value of AEAD Algorithm from the Common Context (see Section 2.1.1).

The Sender ID SHALL be unique for each endpoint in a group with a certain tuple (Master Secret, Master Salt, Group Identifier), see Section 3.3 of [RFC8613].

For Group OSCORE, the Sender Context and Recipient Context additionally contain asymmetric keys, as described previously in Section 2. The private key of the sender and the authentication credential including the corresponding public key can, for example, be generated by the endpoint or provisioned during manufacturing.

With the exception of the authentication credential of the sender endpoint and the possibly associated pairwise keys, a receiver endpoint can derive a complete Security Context from a received Group

OSCORE message and the Common Context. 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 not feasible 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 admits a maximum amount of Recipient Contexts for a same Security Context, e.g., due to memory limitations. After reaching that limit, the creation of a new Recipient Context results in an overflow. When this happens, the endpoint has to delete a current Recipient Context to install the new one. It is up to the application to define policies for selecting the current Recipient Context to delete. If the new Recipient Context has been installed after the endpoint has experienced the overflow above, then the Recipient Context is initialized with an invalid Replay Window, and accordingly requires the endpoint to take appropriate actions (see Section 2.5.1.2).

2.3. Authentication Credentials

In a group, the following MUST hold for the authentication credential of each endpoint as well as for the authentication credential of the Group Manager.

- * All authentication credentials MUST be encoded according to the same format used in the group. The used format MUST provide the public key as well as the comprehensive set of information related to the public key algorithm, including, e.g., the used elliptic curve (when applicable).
- * All authentication credentials and the public key specified therein MUST be for the public key algorithm used in the group and aligned with the possible associated parameters used in the group, e.g., the used elliptic curve (when applicable).

If the group uses (also) the group mode, the public key algorithm is the Signature Algorithm used in the group. If the group uses only the pairwise mode, the public key algorithm is the Pairwise Key Agreement Algorithm used in the group.

If the authentication credentials are X.509 certificates [RFC7925] 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 (CCSs) [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 2.4.1) and are included in the external additional authenticated data when processing messages (see Section 4.3). 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 CCSs 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.

2.4. 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).

2.4.1. 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 12.15. For X25519 and X448, the procedure is described in Section 5 of [RFC7748] using public keys mapped to Montgomery coordinates, see Section 2.4.2.
- * 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 2.3.

- * 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 2.3.
- * 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 and X448 functions defined in Section 5 of [RFC7748]. For further details, see Section 2.4.2. ECC asymmetric keys in Montgomery or Weirstrass form are used directly in the key agreement algorithm without coordinate mapping.

After establishing a partially or completely new Security Context (see Section 2.5 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.

2.4.2. ECDH with Montgomery Coordinates

2.4.2.1. 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 $u = (1 + y) / (1 - y) \pmod{p}$ 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 (= 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, 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].

2.4.2.2. 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 $u = y^2 * (d * y^2 - 1) / (y^2 - 1) \pmod{p}$ 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 $-1 \pmod{p}$.

The private signing key byte strings (= 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, 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].

2.4.3. 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 group and pairwise communication modes, 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 2.5.2).

2.4.4. Security Context for Pairwise Mode

If the pairwise mode is supported, the Security Context additionally includes 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 2.4.1) 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 2.5.3). 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.

2.5. 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 a 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 2.5.3).

On the other hand, the mutable parts of the Security Context are updated by the endpoint when executing the security protocol, but may nevertheless become outdated, e.g., due to loss of the mutable Security Context (see Section 2.5.1) or exhaustion of Sender Sequence Numbers (see Section 2.5.2).

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 and MUST accordingly take the actions defined in Section 2.5.1.

2.5.1. Loss of Mutable Security Context

An endpoint may lose its mutable Security Context, e.g., due to a reboot (see Section 2.5.1.1) or to an overflow of Recipient Contexts (see Section 2.5.1.2).

In such a case, the endpoint needs to prevent the re-use of a nonce with the same AEAD key, and to handle incoming replayed messages.

2.5.1.1. Reboot and Total Loss

In case a loss of the Sender Context and/or of the Recipient Contexts is detected (e.g., following a reboot), the endpoint **MUST NOT** protect further messages using this Security Context to avoid reusing an AEAD nonce with the same AEAD key.

In particular, before resuming its operations in the group, the endpoint **MUST** retrieve new Security Context parameters from the Group Manager (see Section 2.5.3) and use them to derive a new Sender Context (see Section 2.2). Since this includes a newly derived Sender Key, a server will not reuse the same pair (key, nonce), even when using the Partial IV of (old re-injected) requests to build the AEAD nonce for protecting the corresponding responses.

From then on, the endpoint **MUST** use the latest installed Sender Context to protect outgoing messages. Also, newly created Recipient Contexts will have a Replay Window which is initialized as valid.

If not able to establish an updated Sender Context, e.g., because of lack of connectivity with the Group Manager, 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.

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 a 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.

2.5.1.2. Overflow of Recipient Contexts

After reaching the maximum amount of Recipient Contexts, an endpoint will experience an overflow when installing a new Recipient Context, as it requires to first delete an existing one (see Section 2.2).

Every time this happens, the Replay Window of the new Recipient Context is initialized as not valid. Therefore, the endpoint MUST take the following actions, before accepting request messages from the client associated with the new Recipient Context.

If it is not configured as silent server, the endpoint MUST either:

- * Retrieve new Security Context parameters from the Group Manager and derive a new Sender Context, as defined in Section 2.5.1.1; or
- * When receiving a first request to process with the new Recipient Context, use the approach specified in Section 10 and based on the Echo Option for CoAP [RFC9175], if supported. 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. If the endpoint supports the CoAP Echo Option, it is RECOMMENDED to take this approach.

If it is configured exclusively as silent server, the endpoint MUST wait for the next group rekeying to occur, in order to derive a new Security Context and re-initialize the Replay Window of each Recipient Contexts as valid.

2.5.2. Exhaustion of Sender Sequence Number

An endpoint can eventually exhaust the Sender Sequence Number, which is incremented for each new outgoing message including a Partial IV. This is the case for group requests, Observe notifications [RFC7641] and, optionally, any other response.

Implementations MUST be able to detect an exhaustion of Sender Sequence Number, after the endpoint has consumed the largest usable value. If an implementation's integers support wrapping addition, the implementation MUST treat Sender Sequence Number as exhausted when a wrap-around is detected.

Upon exhausting the Sender Sequence Numbers, 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 2.5.3), 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.

2.5.3. 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 2.5.3.1); and ii) establishment of a new Security Context for the group (see Section 2.5.3.2). The update of the Replay Window in each of the Recipient Contexts is discussed in Section 6.2.

As group membership changes, or as group members get new Sender IDs (see Section 2.5.3.1) 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.

2.5.3.1. 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 exhaustion of Sender Sequence Numbers (see Section 2.5.2). An endpoint may request to re-join the group, e.g., because of losing its mutable Security Context (see Section 2.5.1), 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).

2.5.3.2. 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 2.5.3.1), 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 install 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.
- For each ongoing observation where it is an observer client and that it wants to keep active, it resets to 0 the Notification Number of each associated server (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 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 12.5.1).

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 12.5.

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 and 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 3.3.

A possible Group Manager to use 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 [I-D.ietf-ace-oauth-authz].

The Group Manager assigns an integer Key Generation Number to each of its groups, identifying the current version of the keying material used in that group. The first Key Generation Number assigned to every group MUST be 0. Separately for each group, the value of the

Key Generation Number increases strictly monotonically, each time the Group Manager distributes new keying material to that group (see Section 3.2). That is, if the current Key Generation Number for a group is X , then $X+1$ will denote the keying material distributed and used in that group immediately after the current one.

The Group Manager assigns unique Group Identifiers (Gids) to the groups under its control. Also, for each group, the Group Manager assigns unique Sender IDs (and thus Recipient IDs) to the respective group members. 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 also with the current Gid to use in the group.

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 and to external entities with selected roles (see Section 3.1). Upon endpoints' joining, the Group Manager collects such authentication credentials and MUST verify proof-of-possession of the respective private key.

An endpoint acquires group data such as the Gid and OSCORE input parameters including its own Sender ID from the Group Manager, and provides information about its authentication credential to the Group Manager, for example upon joining the group.

Furthermore, when joining the group or later on as a group member, an endpoint can 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. Together with the requested authentication credentials, the Group Manager MUST provide the Sender ID of the associated group members and the current Key Generation Number in the group. 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.

3.1. Support for Additional Entities

The Group Manager MAY serve additional entities acting as signature checkers, e.g., intermediary gateways. 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 solely verify countersignatures of messages protected in group mode (see Section 8.5).

In order to verify countersignatures of messages in a group, a signature checker needs to retrieve the following information about that group from the Group Manager.

- * The current ID Context (Gid) used in the group.
- * The authentication credentials of the group members and the authentication credential 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 that the signature checker stores and uses 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 that the signature checker stores and uses is just the end-entity certificate or end-entity untagged CWT.

- * The current Group Encryption Key (see Section 2.1.6).
- * The identifiers of the algorithms used in the group (see Section 2), i.e.: i) Signature Encryption Algorithm and Signature Algorithm; and ii) AEAD Algorithm and Pairwise Key Agreement Algorithm, if the group uses also the pairwise mode.

A signature checker MUST be authorized before it can retrieve such information. To this end, the same method mentioned above based on the ACE framework [I-D.ietf-ace-oauth-authz] can be used.

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-oscure]. 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 when 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.

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 2.5.3.1).
 - * 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 group rekeying or more consecutive instances. 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.

- * 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).

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 MUST rekey the group in such a way that the new keying material is not provided to those evicted elder members.

This ensures that an Observe notification [RFC7641] can never successfully match against the Observe requests of two different observations. In fact, the excluded elder members would eventually re-join the group, thus terminating any of their ongoing (long-lasting) observations (see Section 6.1). Therefore, it is ensured by construction that no observer client can have two different ongoing observations such that the two respective Observe requests were protected using the same Partial IV, Gid and Sender ID.

- Until a further following group rekeying, the Group Manager MUST store the list of those latest-evicted elder members. If any of those endpoints re-joins the group before a further following group rekeying occurs, the Group Manager MUST NOT rekey the group upon their re-joining. When one of those endpoints re-joins the group, the Group Manager can rely, e.g., on the ongoing secure communication association to recognize the endpoint as included in the stored list.

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 Gid, Master Secret and 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

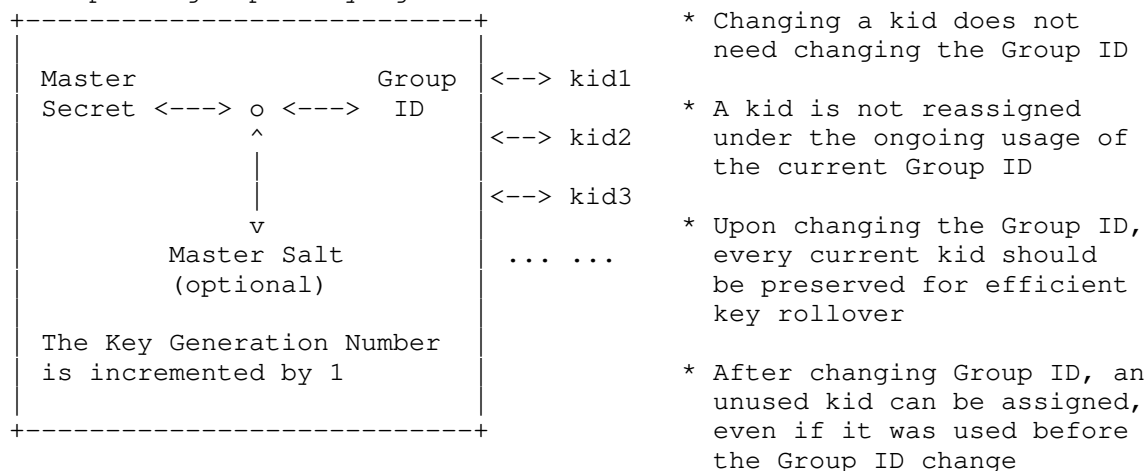


Figure 2: Relations among keying material components.

3.3. 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 these functionalities.

4. The COSE Object

Building on Section 5 of [RFC8613], this section defines how to use COSE [I-D.ietf-cose-rfc8152bis-struct] to wrap and protect data in the original message. OSCORE uses the untagged COSE_Encrypt0 structure with an Authenticated Encryption with Associated Data (AEAD) algorithm. Unless otherwise specified, the following modifications 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:

- * COSE_CounterSignature0: its value is set to the encrypted countersignature of the COSE object, namely ENC_SIGNATURE. That is:

- The countersignature of the COSE object, namely SIGNATURE, is computed by the sender as described in Sections 3.2 and 3.3 of [I-D.ietf-cose-countersign], 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 4.3 of this document, and the ciphertext of the COSE object as payload.

- The encrypted countersignature, namely ENC_SIGNATURE, is computed as

ENC_SIGNATURE = SIGNATURE XOR KEYSTREAM

where KEYSTREAM is derived as per Section 4.1.1.

4.1.1. Keystream Derivation

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 Group Encryption Key from the Common Context (see Section 2.1.6).
- * 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 2.1.5), in bytes.

4.1.2. Clarifications on Using a Countersignature

Note that 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 [I-D.ietf-cose-countersign], 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 requests 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. 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 authenticated encryption/decryption (see Section 5.3 of [I-D.ietf-cose-rfc8152bis-struct]), as well as in group mode for computing and verifying the countersignature (see Section 4.4 of [I-D.ietf-cose-rfc8152bis-struct]).

In particular, the `external_aad` includes also the Signature Algorithm, the Signature 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_signature_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 / null
]
```

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 group does not use the pairwise mode, regardless whether the endpoint supports the pairwise mode or not. Otherwise, this parameter MUST encode the value of 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_signature_enc'`, which specifies Signature Encryption Algorithm from the Common Context (see Section 2.1.5). This parameter MUST be set to the CBOR simple value "null" (0xf6) if the group does not use the group mode, regardless whether the endpoint supports the group mode or not. Otherwise, this parameter MUST encode the value of Signature Encryption Algorithm as a CBOR integer or text string, consistently with the "Value" field in the "COSE Algorithms" Registry for this AEAD algorithm.
 - `'alg_signature'`, which specifies Signature Algorithm from the Common Context (see Section 2.1.5). This parameter MUST be set to the CBOR simple value "null" (0xf6) if the group does not use the group mode, regardless whether the endpoint supports the group mode or not. Otherwise, this parameter MUST encode the value of Signature Algorithm as a CBOR integer or text string, consistently with the "Value" field in the "COSE Algorithms" Registry for this signature algorithm.
 - `'alg_pairwise_key_agreement'`, which specifies Pairwise Key Agreement Algorithm from the Common Context (see Section 2.1.5). This parameter MUST be set to the CBOR simple value "null" (0xf6) if the group does not use the pairwise mode, regardless whether the endpoint supports the pairwise mode or not. Otherwise, this parameter MUST encode the value of Pairwise Key Agreement Algorithm as a CBOR integer or text string, consistently with the "Value" field in the "COSE Algorithms" Registry for this HKDF 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).

In case Observe [RFC7641] is used, this enables endpoints to safely keep an observation 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 notification cryptographically matches with only one observation request, rather than with multiple ones 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 12.7 when using the group mode, as further explained in Section 12.7.2.

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 2.3).
- * The new element 'gm_cred', containing the Group Manager's authentication credential. If no Group Manager maintains the group, this parameter MUST encode the CBOR simple value "null" (0xf6). Otherwise, 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 2.3). This prevents the attack described in Section 12.8.

5. OSCORE Header Compression

The OSCORE header compression defined in Section 6 of [RFC8613] is used, with the following differences.

- * The payload of the OSCORE message SHALL encode the ciphertext of the COSE_Encrypt0 object. In the group mode, the ciphertext above is concatenated with the value of the COSE_CounterSignature0 of the COSE object, computed as described in Section 4.1.
- * 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 13.1.
- * The Group Flag MUST be set to 1 if the OSCORE message is protected using the group mode (see Section 8).
- * The Group Flag MUST be set to 0 if the 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. Examples of Compressed COSE Objects

This section covers a list of OSCORE Header Compression examples of Group OSCORE used in group mode (see Section 5.1.1) or in pairwise mode (see Section 5.1.2).

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.

5.1.1. Examples in Group Mode

- * Request with ciphertext = 0xaa0155667924dff8a24e4cb35b9, kid = 0x25, Partial IV = 5 and kid context = 0x44616c.

* Before compression (96 bytes):

```
[
  h'',
  { 4:h'25', 6:h'05', 10:h'44616c', 11:h'de9e ... f1' },
  h'aa0155667924dff8a24e4cb35b9'
]
```

* After compression (85 bytes):

Flag byte: 0b00111001 = 0x39 (1 byte)

Option Value: 0x39 05 03 44 61 6c 25 (7 bytes)

Payload: 0xaa0155667924dff8a24e4cb35b9 de9e ... f1
(14 bytes + size of the encrypted countersignature)

- * Response with ciphertext = 0x60b035059d9ef5667c5a0710823b, kid = 0x52 and no Partial IV.

* Before compression (88 bytes):

```
[  
  h'',  
  { 4:h'52', 11:h'cale ... b3' },  
  h'60b035059d9ef5667c5a0710823b'  
]
```

* After compression (80 bytes):

Flag byte: 0b00101000 = 0x28 (1 byte)

Option Value: 0x28 52 (2 bytes)

Payload: 0x60b035059d9ef5667c5a0710823b cale ... b3
(14 bytes + size of the encrypted countersignature)

5.1.2. Examples in Pairwise Mode

* Request with ciphertext = 0xaea0155667924dff8a24e4cb35b9, kid = 0x25, Partial IV = 5 and kid context = 0x44616c.

* Before compression (29 bytes):

```
[  
  h'',  
  { 4:h'25', 6:h'05', 10:h'44616c' },  
  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):

```
[  
  h'',  
  {},  
  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

The requirements and properties described in Section 7 of [RFC8613] also apply to Group OSCORE. In particular, Group OSCORE provides message binding of responses to requests, which enables absolute freshness of responses that are not notifications, relative freshness of requests and notification responses, and replay protection of requests. In addition, the following holds for Group OSCORE.

6.1. Supporting Observe

When Observe [RFC7641] is used, a client maintains for each ongoing observation one Notification Number for each different server. Then, separately for each server, the client uses the associated Notification Number to perform ordering and replay protection of notifications received from that server (see Section 8.4.1).

Group OSCORE allows to preserve an observation active indefinitely, even in case the group is rekeyed, with consequent change of ID Context, or in case the observer client obtains a new Sender ID.

As defined in Section 8 when discussing support for Observe, this is achieved by the client and server(s) storing the 'kid' and 'kid context' used in the original Observe request, throughout the whole duration of the observation.

Upon leaving the group or before re-joining the group, a group member MUST terminate all the ongoing observations that it has started in the group as observer client.

6.2. Update of Replay Window

Sender Sequence Numbers seen by a server as Partial IV values in request messages can spontaneously increase at a fast pace, for example when a client exchanges unicast messages with other servers using the Group OSCORE Security Context. As in OSCORE [RFC8613], a server always needs to accept such increases and accordingly updates the Replay Window in each of its Recipient Contexts.

As discussed in Section 2.5.1, a newly created Recipient Context would have an invalid Replay Window, if its installation has required to delete another Recipient Context. Hence, the server is not able to verify if a request from the client associated with the new Recipient Context is a replay. When this happens, the server **MUST** validate the Replay Window of the new Recipient Context, before accepting messages from the associated client (see Section 2.5.1).

Furthermore, when the Group Manager establishes a new Security Context for the group (see Section 2.5.3.2), every server re-initializes the Replay Window in each of its Recipient Contexts.

6.3. Message Freshness

When receiving a request from a client for the first time, the server is not synchronized with the client's Sender Sequence Number, i.e., it is not able to verify if that request is fresh. This applies to a server that has just joined the group, with respect to already present clients, and recurs as new clients are added as group members.

During its operations in the group, the server may also lose synchronization with a client's Sender Sequence Number. This can happen, for instance, if the server has rebooted or has deleted its previously synchronized version of the Recipient Context for that client (see Section 2.5.1).

If the application requires message freshness, e.g., according to time- or event-based policies, the server has to (re-)synchronize with a client's Sender Sequence Number before delivering request messages from that client to the application. To this end, the server can use the approach in Section 10 based on the Echo Option for CoAP [RFC9175], as a variant of the approach defined in Appendix B.1.2 of [RFC8613] applicable to Group OSCORE.

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/or Pairwise Key Agreement Algorithm in the Common Context. Alternatively implementations can use an additional parameter in the Security Context, to explicitly signal 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 recipient endpoint retrieves a Security Context which is both valid to process the message and also associated with an OSCORE group, but the endpoint does not support the pairwise mode.
- * The Group Flag is set to 1, and the recipient endpoint retrieves a Security Context which is both valid to process the message and also associated with an OSCORE group, but the endpoint does not support the group mode.
- * The Group Flag is set to 1, and the recipient endpoint can not retrieve a Security Context which is both valid to process the message and also associated with an OSCORE group.

As per Section 6.1 of [RFC8613], this holds also when retrieving a Security Context which is valid but not associated with an OSCORE group. Future specifications may define how to process incoming messages protected with a Security Contexts as in [RFC8613], when the Group Flag bit is set to 1.

Otherwise, if a Security Context associated with an OSCORE group and valid to process the message 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 Group Manager indicates that the group uses (also) the group mode, as part of the group data provided to candidate group members when joining the group.

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 2.5.3.2) until the message processing is completed.

The group mode MUST be used to protect group requests intended for multiple recipients or for the whole group. This includes 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]. For encryption and decryption operations, the Signature Encryption Algorithm from the Common Context is used.

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. 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]. However, a recipient MUST stop processing and reject any message which is malformed and does not follow the format specified in Section 4 of this document, or which is not cryptographically validated in a successful way.

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

A client transmits a secure group request 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 Signature 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 OSCORE message includes also the encrypted countersignature (see Section 4.1).

8.1.1. Supporting Observe

If Observe [RFC7641] is supported, the following holds for each newly started observation.

- * If the client intends to keep the observation active beyond a possible change of Sender ID, the client MUST store the value of the 'kid' parameter from the original Observe request, and retain it for the whole duration of the observation. Even in case the client is individually rekeyed and receives a new Sender ID from the Group Manager (see Section 2.5.3.1), the client MUST NOT update the stored value associated with a particular Observe request.
- * If the client intends to keep the observation 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 original Observe request, and retain it for the whole duration of the observation. Upon establishing a new Security Context with a new Gid as ID Context (see Section 2.5.3.2), the client MUST NOT update the stored value associated with a particular Observe 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 observer 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 observe notification protected with the new Security Context (see Section 8.3.1).

8.2. Verifying the Request

Upon receiving a secure group request with the Group Flag set to 1, following the procedure in Section 7, a server proceeds 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 4.1.1.

The server verifies the original countersignature SIGNATURE.

- 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. The server MAY set an Outer Max-Age option with value zero. The diagnostic payload MAY contain a string, which, if present, MUST be "Decryption failed" as if the decryption had failed.
 - When decrypting the COSE object using the Recipient Key, the Signature Encryption Algorithm from the Common Context MUST be used.
- * Additionally, if the used Recipient Context was created upon receiving this group request and the message is not verified successfully, the server MAY delete that Recipient Context. Such a configuration, which is specified by the application, 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 7.2.1 of [I-D.ietf-core-observe-multicast-notifications].

8.2.1. Supporting Observe

If Observe [RFC7641] is supported, the following holds for each newly started observation.

- * The server MUST store the value of the 'kid' parameter from the original Observe request, and retain it for the whole duration of the observation. The server MUST NOT update the stored value of a 'kid' parameter associated with a particular Observe request, even in case the observer client is individually rekeyed and starts using a new Sender ID received from the Group Manager (see Section 2.5.3.1).
- * The server MUST store the value of the 'kid context' parameter from the original Observe request, and retain it for the whole duration of the observation, 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 2.5.3.2), the server MUST NOT update the stored value associated with the ongoing observation.

8.3. Protecting the Response

If a server generates a CoAP message in response to a Group OSCORE request, then the server SHALL follow the description in Section 8.3 of [RFC8613], with the modifications described in this section.

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 step 3, if the server is using a different Security Context for the response compared to what was used to verify the request (see Section 3.2), then 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.
- * 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 Signature Encryption Algorithm from the Common Context MUST be used.

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), then the new ID Context MUST be included in the 'kid context' parameter of the response.

The server can obtain a new Sender ID from the Group Manager, when individually rekeyed (see Section 2.5.3.1) 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 OSCORE message includes also the encrypted countersignature (see Section 4.1).

8.3.1. Supporting Observe

If Observe [RFC7641] is supported, the following holds when protecting notifications for an ongoing observation.

- * The server MUST use the stored value of the 'kid' parameter from the original Observe request (see Section 8.2.1), as value for the 'request_kid' parameter in the external_aad structure (see Section 4.3).
- * The server MUST use the stored value of the 'kid context' parameter from the original Observe request (see Section 8.2.1), as value for the 'request_kid_context' parameter in the external_aad structure (see Section 4.3).

Furthermore, the server may have ongoing observations started by Observe requests protected with an old Security Context. After completing the establishment of a new Security Context, the server MUST protect the following notifications with the Sender Context of the new Security Context.

For each ongoing observation, the server can help the client to synchronize, by including also the 'kid context' parameter in notifications 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 notification sent before the installation of the new Security Context.

8.4. Verifying the Response

Upon receiving a secure response message with the Group Flag set to 1, following the procedure in Section 7, the client proceeds as described in Section 8.4 of [RFC8613], with the following modifications.

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 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 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 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 4.1.1.

The client verifies the original countersignature SIGNATURE.

- If the verification of the countersignature fails, the server SHALL stop processing the response, and SHALL NOT update the Notification Number associated with the server if the response is an Observe notification [RFC7641].
- After a successful verification of the countersignature, the client performs also the following actions if the response is not an Observe notification.
 - o In case the request was protected in pairwise mode and 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 In case the request was protected in pairwise mode and 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 2.4.1).

In either case, if the client determines that the response has come from a different server than the expected one, then the client SHALL discard the response and SHALL NOT deliver it to the application. Otherwise, the client hereafter considers the received 'kid' as the current Recipient ID for the server.

- When decrypting the COSE object using the Recipient Key, the Signature Encryption Algorithm from the Common Context MUST be used.
- * Additionally, if the used Recipient Context was created upon receiving this response and the message is not verified successfully, the client MAY delete that Recipient Context. Such a configuration, which is specified by the application, mitigates attacks that aim at overloading the client's storage.

8.4.1. Supporting Observe

If Observe [RFC7641] is supported, the following holds when verifying notifications for an ongoing observation.

- * The client MUST use the stored value of the 'kid' parameter from the original Observe request (see Section 8.1.1), as value for the 'request_kid' parameter in the external_aad structure (see Section 4.3).
- * The client MUST use the stored value of the 'kid context' parameter from the original Observe request (see Section 8.1.1), as value for the 'request_kid_context' parameter in the external_aad structure (see Section 4.3).

This ensures that the client can correctly verify notifications, even in case it is individually rekeyed and starts using a new Sender ID received from the Group Manager (see Section 2.5.3.1), as well as when it installs a new Security Context with a new ID Context (Gid) following a group rekeying (see Section 3.2).

- * The ordering and the replay protection of notifications received from a server are performed as per Sections 4.1.3.5.2 and 7.4.1 of [RFC8613], by using the Notification Number associated with that server for the observation in question. In addition, the client performs the following actions for each ongoing observation.
 - When receiving the first valid notification from a server, the client MUST store the current kid "kid1" of that server for the observation in question. If the 'kid' field is included in the OSCORE option of the notification, its value specifies "kid1". If the Observe request was protected in pairwise mode (see Section 9.3), the 'kid' field may not be present in the OSCORE option of the notification (see Section 4.2). In this case, the client assumes "kid1" to be the Recipient ID for the server to which the Observe request was intended for.

- When receiving another valid notification 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 MUST cancel or re-register the observation in question.

Note that, if "kid2" is different from "kid1" and the 'kid' field is omitted from the notification - which is possible if the Observe 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 4.1.1), thus subsequently failing to verify the countersignature and discarding the notification.

This ensures that the client remains able to correctly perform the ordering and replay protection of notifications, even in case a server legitimately starts using a new Sender ID, as received from the Group Manager when individually rekeyed (see Section 2.5.3.1) or when re-joining the group.

8.5. External Signature Checkers

When receiving a message protected in group mode, a signature checker (see Section 3.1) proceeds as follows.

- * The signature checker retrieves the encrypted countersignature ENC_SIGNATURE from the message payload, and computes the original countersignature SIGNATURE as

$$\text{SIGNATURE} = \text{ENC_SIGNATURE} \text{ XOR } \text{KEYSTREAM}$$

where KEYSTREAM is derived as per Section 4.1.1.

- * 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 pairwise mode takes advantage of an existing Security Context for the group mode to establish a Security Context shared exclusively with any other member. In order to use the pairwise mode in a group that uses also the group mode, the signature scheme of the group mode MUST support a combined signature and encryption scheme. This can be, for example, signature using ECDSA, and encryption using AES-CCM with a key derived with ECDH. For encryption and decryption operations, the AEAD Algorithm from the Common Context is used (see Section 2.1.1).

The pairwise mode does not support the use of additional entities acting as verifiers of source authentication and integrity of group messages, such as intermediary gateways (see Section 3).

An endpoint implementing only a silent server does not support the pairwise mode.

If the signature algorithm used in the group supports ECDH (e.g., ECDSA, EdDSA), the pairwise mode MUST be supported by endpoints that use the CoAP Echo Option [RFC9175] and/or block-wise transfers [RFC7959], for instance for responses after the first block-wise request, which possibly 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 12.9, which leverages requests sent over unicast to a single group member and protected with the group mode.

Senders cannot use the pairwise mode to protect a message intended for multiple recipients. In fact, the pairwise mode is defined only between two endpoints and the keying material is thus only available to one recipient.

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.

The Group Manager indicates that the group uses (also) the pairwise mode, as part of the group data provided to candidate group members when joining the group.

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 2.4.4).

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.

To make addressing information of individual endpoints available, servers in the group MAY expose a resource to which a client can send a group request targeting a set of servers, identified by their 'kid' values specified in the request payload. The specified set may be empty, hence identifying all the servers in the group. 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 4.3 of this document is used for the encryption process.
- * The Pairwise Sender/Recipient Keys used as Sender/Recipient keys are derived as defined in Section 2.4 of this document.

9.3. Protecting the Request

When using the pairwise mode, the request is protected as defined in Section 8.1 of [RFC8613], with the differences summarized in Section 9.2 of this document. The following difference also applies.

- * If Observe [RFC7641] is supported, what is defined in Section 8.1.1 of this document holds.

9.4. Verifying the Request

Upon receiving a request with the Group Flag set to 0, following the procedure in Section 7, the server MUST process it as defined 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 2.4.1). The new Pairwise Sender/Recipient Keys are deleted if the Recipient Context is deleted as a result of the message not being successfully verified.
- * If Observe [RFC7641] is supported, what is defined in Section 8.2.1 of this document holds.

9.5. Protecting the Response

When using the pairwise mode, a response is protected as defined in Section 8.3 of [RFC8613], with the differences summarized in Section 9.2 of this document. The following differences also apply.

- * If the server is using a different Security Context for the response compared to what was used to verify the request (see Section 3.2), then 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.
- * 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), then the new ID Context MUST be included in the 'kid context' parameter of the response.
- * The server can obtain a new Sender ID from the Group Manager, when individually rekeyed (see Section 2.5.3.1) 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 pairwise mode, even when the request was also protected in pairwise mode.

That is, when responding to a request protected in pairwise mode, the server SHOULD include the 'kid' parameter in a response protected in pairwise mode, if it is replying to that client for the first time since the assignment of its new Sender ID.

- * If Observe [RFC7641] is supported, what is defined in Section 8.3.1 of this document holds.

9.6. Verifying the Response

Upon receiving a response with the Group Flag set to 0, following the procedure in Section 7, the client MUST process it as defined 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 2.4.1).

- * If a new Recipient Context is created for this Recipient ID, new Pairwise Sender/Recipient Keys are also derived (see Section 2.4.1). The new Pairwise Sender/Recipient Keys are deleted if the Recipient Context is deleted as a result of the message not being successfully verified.
- * If Observe [RFC7641] is supported, what is defined in Section 8.4.1 of this document holds. The client can also in this case identify a server to be the same one across a change of Sender ID, by relying on the server's public key. As to the expected server's authentication credential, the same holds as specified above for non-notification responses.

10. Challenge-Response Synchronization

This section describes how a server endpoint can 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 an OSCORE protected 4.01 (Unauthorized) response message, including only the Echo Option and no diagnostic payload. The Echo option value SHOULD NOT be reused; when 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 client sends a request as a unicast message addressed to the same server, echoing the Echo Option value. The client MUST NOT send the request including the Echo Option over multicast.

If the group uses also the group mode and the used Signature Algorithm supports ECDH (e.g., ECDSA, EdDSA), 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 MUST support the pairwise mode.

The client does not necessarily resend the same group request, but can instead send a more recent one, if the application permits it. This allows the client to not retain previously sent group 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 group 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 proceeds as follows. In case the Replay Window in the Recipient Context associated with the client has not been set yet, the server updates

the Replay Window to mark the current Sender Sequence Number from the latest received request as seen (but all newer ones as new), and delivers the message as fresh to the application. Otherwise, the server discards the verification result and treats the message as fresh or 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. A client has to be ready to perform the challenge-response based on the Echo Option if a server starts it.

It is the role of the server application to define under what circumstances Sender Sequence Numbers lose synchronization. This can include experiencing a "large enough" gap $D = (SN2 - SN1)$, between the Sender Sequence Number $SN1$ of the latest accepted group request from a client and the Sender Sequence Number $SN2$ of a group request just received from that client. However, a client may send several unicast requests to different group members as protected with the pairwise mode, which may result in the server experiencing the gap D in a relatively short time. This would induce the server to perform more challenge-response exchanges than actually needed.

In order to ameliorate this, the server may rely on a trade-off between the Sender Sequence Number gap D and a time gap $T = (t2 - t1)$, where $t1$ is the time when the latest group request from a client was accepted and $t2$ is the time when the latest group request from that client has been received, respectively. Then, the server can start a challenge-response when experiencing a time gap T larger than a given, preconfigured threshold. Also, the server can start a challenge-response when experiencing a Sender Sequence Number gap D greater than a different threshold, computed as a monotonically increasing function of the currently experienced time gap T .

The challenge-response approach described in this section provides an assurance of absolute message freshness. However, it can result in an impact on performance which is undesirable or unbearable, especially in large groups where many endpoints at the same time might join as new members or lose synchronization.

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 12.9, in case such requests are protected with the group mode. Instead, protecting those requests with the pairwise mode prevents the attack above. In fact, only the exact 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.

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 [I-D.ietf-cose-rfc8152bis-algs] is mandatory to implement as Signature Encryption Algorithm (see Section 2.1.4).
- * 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 2.1.5).

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 [I-D.ietf-cose-rfc8152bis-algs], this can be addressed by combining both randomness and determinism [I-D.mattsson-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 [I-D.ietf-cose-rfc8152bis-algs] 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 [I-D.ietf-cose-rfc8152bis-algs] is mandatory to implement as Pairwise Key Agreement Algorithm (see Section 2.1.7).
- * 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. Security Considerations

The same threat model discussed for OSCORE in Appendix D.1 of [RFC8613] holds for Group OSCORE. In addition, when using the group mode, source authentication of messages is explicitly ensured by means of countersignatures, as discussed in Section 12.1.

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 12.3 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 observations active beyond a possible change of ID Context (Gid), following a group rekeying (see Section 4.3). 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 12.7 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.

12.1. 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.

- * 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 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 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.

- * 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 4.1). 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.

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 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 an 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 an encryption algorithm that does not provide integrity protection results in a minimal communication overhead, by limiting the message payload to the ciphertext and the encrypted countersignature. Second, it is possible to deploy semi-trusted entities such as signature checkers (see Section 3.1), which can break property 5, but cannot break properties 1, 2 and 3.

12.2. 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 2.4).

The used encryption 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.

12.3. 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 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 group requests and responses including a Partial IV (e.g., Observe notifications). 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., single response to a group 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.

12.4. Management of Group Keying Material

The approach 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 which 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.

12.5. 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.

12.5.1. 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, recent, Security Context for a maximum amount of time defined by the application. By doing so, the recipient can still try to process the received message using the old retained Security Context as a second attempt. This makes particular sense when the recipient is a client, that would hence be able to process incoming responses protected with the old, recent, Security Context used to protect the associated

group request. Instead, a recipient server would better and more simply discard an incoming group request which is not successfully processed with the new Security Context.

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.

12.5.2. 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.

12.6. 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.

12.7. 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 12.7.1).

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 12.7.2, 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 4.3).

12.7.1. 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.

12.7.2. Attack Prevention in Group Mode

When a Group OSCORE message is protected with the 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 4.3).

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 group requests; as well as the Sender ID of the message originator, which is always present in group 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 12.7.1, 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 12.7.1 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.

12.8. 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 4.3).

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.

12.9. 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 MUST protect 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., (s)he does 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 2.4). 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. A client supporting the pairwise mode SHOULD use it to protect requests sent to a single group member over unicast, instead of using the group mode. For an example where this is not fulfilled, see Section 7.2.1 of [I-D.ietf-core-observe-multicast-notifications].

With particular reference to block-wise transfers [RFC7959], Section 3.8 of [I-D.ietf-core-groupcomm-bis] points out that, while an initial request including the CoAP Block2 option can be sent over multicast, any other request in a transfer has to occur over unicast, individually addressing the servers in the group.

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.

12.10. 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 (also) sent over multicast.

12.11. Master Secret

Group OSCORE derives the Security Context using the same construction as 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].

12.12. 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 Sender Sequence Number, the Partial IV also gets exhausted. As discussed in Section 2.5.3, 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 12.3) 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.

As discussed in Section 6.2, a Replay Window may be initialized as not valid, following the loss of mutable Security Context Section 2.5.1. In particular, Section 2.5.1.1 and Section 2.5.1.2 define measures that endpoints need to take in such a situation, before resuming to accept incoming messages from other group members.

12.13. Message Freshness

As discussed in Section 6.3, a server may not be able to assert whether an incoming request is fresh, in case it does not have or has lost synchronization with the client's Sender Sequence Number.

If freshness is relevant for the application, the server may (re-)synchronize with the client's Sender Sequence Number at any time, by using the approach described in Section 10 and based on the CoAP Echo Option [RFC9175], as a variant of the approach defined in Appendix B.1.2 of [RFC8613] applicable to Group OSCORE.

12.14. Client Aliveness

Building on Section 12.5 of [RFC8613], a server may use the CoAP Echo Option [RFC9175] to verify the aliveness of the client that originated a received request, by using the approach described in Section 10 of this document.

12.15. 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 2.5.2, an endpoint that experiences an exhaustion of its own Sender Sequence Numbers **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 nonces with the same Sender Key.

In order to renew its own Sender Context, the endpoint **SHOULD** inform the Group Manager, which can either renew the whole 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 additional entities acting as signature checkers (see Section 3.1), 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 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 [I-D.ietf-cose-countersign]). 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 2.4.1 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 2.4.1, 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 2.4 as a Key Encapsulation Mechanism (KEM)) is demonstrated in [Degabriele] and [Thormarker].

Applications using ECDH (except X25519 and X448) based KEM in Section 2.4 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 2.4.

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 2.4. 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 2.4, 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 DH, Section 5.7.1.2 of [NIST-800-56A] which is referenced in Section 2.4.

HashEdDSA variants of Ed25519 and Ed448 are not used by COSE, see Section 2.2 of [I-D.ietf-cose-rfc8152bis-algs], 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.

12.16. Message Segmentation

The same considerations from Section 12.7 of [RFC8613] hold for Group OSCORE.

12.17. Privacy Considerations

Group OSCORE ensures end-to-end integrity protection and encryption of the message payload and all options that are not used for proxy operations. In particular, options are processed according to the same class U/I/E that they have for OSCORE. Therefore, the same privacy considerations from Section 12.8 of [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 4.1 and Section 4.1.1). 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 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 Sender Sequence Number synchronization with a client may reveal when a server device goes through a reboot. This can be mitigated by the server device storing the precise state of the Replay Window of each known client on a clean shutdown.

Finally, the approach described in Section 12.6 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.

13. IANA Considerations

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

This document has the following actions for IANA.

13.1. OSCORE Flag Bits Registry

IANA is asked to add the following value 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	[This Document]

14. References

14.1. 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-06, 7 March 2022, <<https://www.ietf.org/archive/id/draft-ietf-core-groupcomm-bis-06.txt>>.

[I-D.ietf-cose-countersign]

Schaad, J. and R. Housley, "CBOR Object Signing and Encryption (COSE): Countersignatures", Work in Progress,

Internet-Draft, draft-ietf-cose-countersign-05, 23 June 2021, <<https://www.ietf.org/archive/id/draft-ietf-cose-countersign-05.txt>>.

[I-D.ietf-cose-rfc8152bis-algs]

Schaad, J., "CBOR Object Signing and Encryption (COSE): Initial Algorithms", Work in Progress, Internet-Draft, draft-ietf-cose-rfc8152bis-algs-12, 24 September 2020, <<https://www.ietf.org/archive/id/draft-ietf-cose-rfc8152bis-algs-12.txt>>.

[I-D.ietf-cose-rfc8152bis-struct]

Schaad, J., "CBOR Object Signing and Encryption (COSE): Structures and Process", Work in Progress, Internet-Draft, draft-ietf-cose-rfc8152bis-struct-15, 1 February 2021, <<https://www.ietf.org/archive/id/draft-ietf-cose-rfc8152bis-struct-15.txt>>.

[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/info/rfc2119>>.

[RFC4086] Eastlake 3rd, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, DOI 10.17487/RFC4086, June 2005, <<https://www.rfc-editor.org/info/rfc4086>>.

[RFC6979] Pornin, T., "Deterministic Usage of the Digital Signature Algorithm (DSA) and Elliptic Curve Digital Signature Algorithm (ECDSA)", RFC 6979, DOI 10.17487/RFC6979, August 2013, <<https://www.rfc-editor.org/info/rfc6979>>.

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

- [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>>.
- [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/info/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/info/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/info/rfc8174>>.
- [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/info/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/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>>.
- [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/info/rfc9175>>.

14.2. Informative References

- [Degabriele] Degabriele, J.P., Lehmann, A., Paterson, K.G., Smart, N.P., and M. Streffler, "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-04, 6 March 2022, <<https://www.ietf.org/archive/id/draft-amsuess-core-cachable-oscore-04.txt>>.

[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-15, 23 December 2021, <<https://www.ietf.org/archive/id/draft-ietf-ace-key-groupcomm-15.txt>>.

[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-13, 7 March 2022, <<https://www.ietf.org/archive/id/draft-ietf-ace-key-groupcomm-oscore-13.txt>>.

[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-46, 8 November 2021, <<https://www.ietf.org/archive/id/draft-ietf-ace-oauth-authz-46.txt>>.

[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-03, 7 March 2022, <<https://www.ietf.org/archive/id/draft-ietf-core-observe-multicast-notifications-03.txt>>.

[I-D.ietf-cose-cbor-encoded-cert]

Mattsson, J. P., Selander, G., Raza, S., Höglund, J., and M. Furuheid, "CBOR Encoded X.509 Certificates (C509 Certificates)", Work in Progress, Internet-Draft, draft-ietf-cose-cbor-encoded-cert-03, 10 January 2022, <<https://www.ietf.org/archive/id/draft-ietf-cose-cbor-encoded-cert-03.txt>>.

[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://www.ietf.org/archive/id/draft-ietf-lwig-curve-representations-23.txt>>.

[I-D.ietf-lwig-security-protocol-comparison]
Mattsson, J. P., Palombini, F., and M. Vucinic,
"Comparison of CoAP Security Protocols", Work in Progress,
Internet-Draft, draft-ietf-lwig-security-protocol-
comparison-05, 2 November 2020,
<<https://www.ietf.org/archive/id/draft-ietf-lwig-security-protocol-comparison-05.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/internet-drafts/draft-ietf-tls-dtls13-43.txt>>.

[I-D.mattsson-cfrg-det-sigs-with-noise]
Mattsson, J. P., Thormarker, E., and S. Ruohomaa,
"Deterministic ECDSA and EdDSA Signatures with Additional
Randomness", Work in Progress, Internet-Draft, draft-
mattsson-cfrg-det-sigs-with-noise-04, 15 February 2022,
<<https://www.ietf.org/archive/id/draft-mattsson-cfrg-det-sigs-with-noise-04.txt>>.

[RFC4944] Montenegro, G., Kushalnagar, N., Hui, J., and D. Culler,
"Transmission of IPv6 Packets over IEEE 802.15.4
Networks", RFC 4944, DOI 10.17487/RFC4944, September 2007,
<<https://www.rfc-editor.org/info/rfc4944>>.

[RFC4949] Shirey, R., "Internet Security Glossary, Version 2",
FYI 36, RFC 4949, DOI 10.17487/RFC4949, August 2007,
<<https://www.rfc-editor.org/info/rfc4949>>.

[RFC5869] Krawczyk, H. and P. Eronen, "HMAC-based Extract-and-Expand
Key Derivation Function (HKDF)", RFC 5869,
DOI 10.17487/RFC5869, May 2010,
<<https://www.rfc-editor.org/info/rfc5869>>.

[RFC6282] Hui, J., Ed. and P. Thubert, "Compression Format for IPv6
Datagrams over IEEE 802.15.4-Based Networks", RFC 6282,
DOI 10.17487/RFC6282, September 2011,
<<https://www.rfc-editor.org/info/rfc6282>>.

- [RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <<https://www.rfc-editor.org/info/rfc6347>>.
- [RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, DOI 10.17487/RFC7228, May 2014, <<https://www.rfc-editor.org/info/rfc7228>>.
- [RFC7925] Tschofenig, H., Ed. and T. Fossati, "Transport Layer Security (TLS) / Datagram Transport Layer Security (DTLS) Profiles for the Internet of Things", RFC 7925, DOI 10.17487/RFC7925, July 2016, <<https://www.rfc-editor.org/info/rfc7925>>.
- [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>>.
- [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/info/rfc8392>>.
- [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 approach 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 2 to 100 devices. 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 ciphersuite: all members of a security group must use the same ciphersuite to provide authenticity, integrity and confidentiality of messages in the group. The ciphersuite is specified as part of the Security Context.
- * 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.
- * 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 approach described in this document aims at fulfilling the following security objectives:

- * Data replay protection: group 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. In accordance with OSCORE [RFC8613], this results in providing absolute freshness of responses that are not notifications, as well as relative freshness of group requests and notification responses. 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, a M-to-N communication topology would be required, for instance in case multiple sensors (presence or day-light) are responsible to trigger events to a 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 (BACSSs) 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 controls. Devices receiving parameter and configuration updates are expected to possibly reply back, to provide a feedback about the execution of the update operation (e.g., OK, failure, error) and their current operational status.
- * 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 assumes that enough time elapses before all possible Group Epoch values are used, i.e., before the Group Manager terminates the group or starts reassigning Gid values to the group (see Section 3.2). Thus, the expected highest rate for addition/removal of group members and consequent group rekeying should be taken into account for a proper dimensioning of the Group Epoch size.

As discussed in Section 12.6, 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. Set-up of New Endpoints

An endpoint joins a group by explicitly interacting with the responsible Group Manager. When becoming members of a group, endpoints are not required to know how many and what endpoints are in the same group.

Communications between a joining endpoint and the Group Manager rely on the CoAP protocol and must be secured. Specific details on how to secure communications between joining endpoints and a Group Manager are out of the scope of this document.

The Group Manager must verify that the joining endpoint is authorized to join the group. To this end, the Group Manager can directly authorize the joining endpoint, or expect it to provide authorization evidence previously obtained from a trusted entity. Further details about the authorization of joining endpoints are out of scope.

In case of successful authorization check, the Group Manager generates a Sender ID assigned to the joining endpoint, before proceeding with the rest of the join process. That is, the Group Manager provides the joining endpoint with the keying material and parameters to initialize the Security Context, including its own authentication credential (see Section 2). The actual provisioning of keying material and parameters to the joining endpoint is out of the scope of this document.

As mentioned in Section 3, the Group Manager and the join process can be as specified in [I-D.ietf-ace-key-groupcomm-oscore].

Appendix E. Document Updates

RFC EDITOR: PLEASE REMOVE THIS SECTION.

E.1. 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.

E.2. Version -12 to -13

- * Fixes in the derivation of the Group Encryption Key.
- * Added Mandatory-to-Implement compliance requirements.
- * Changed UCCS to CCS.

E.3. 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.

E.4. 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 with the 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.

E.5. 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.

E.6. 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.

E.7. 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 (Section 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.

E.8. 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.

E.9. 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.

E.10. 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).

E.11. 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).

E.12. 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.

E.13. 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.

E.14. 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

The authors sincerely thank Christian Amsuess, Stefan Beck, Rolf Blom, Carsten Bormann, Esko Dijk, Martin Gunnarsson, Klaus Hartke, Rikard Hoeglund, 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 project SIFIS-Home (Grant agreement 952652); 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

Jiye Park
Universitaet Duisburg-Essen
Schuetzenbahn 70
45127 Essen
Germany
Email: ji-ye.park@uni-due.de

CoRE Working Group
Internet-Draft
Updates: 8613 (if approved)
Intended status: Standards Track
Expires: 8 September 2022

M. Tiloca
R. Höglund
RISE AB
7 March 2022

OSCORE-capable Proxies
draft-tiloca-core-oscore-capable-proxies-02

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

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

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

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

This Internet-Draft will expire on 8 September 2022.

Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.

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

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	6
2.4. Further Use Cases	7
3. Message Processing	8
3.1. General Rules on Protecting Options	8
3.2. Processing an Outgoing Request	9
3.3. Processing an Incoming Request	10
3.4. Processing an Outgoing Response	11
3.5. Processing an Incoming Response	11
4. Example	11
5. Caching of OSCORE-Protected Responses	11
6. Security Considerations	12
7. IANA Considerations	13
8. References	13
8.1. Normative References	13
8.2. Informative References	13
Appendix A. OSCORE-protected Onion Forwarding	15
Acknowledgments	18
Authors' Addresses	18

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 option:** the Proxy-URI Option, the Proxy-Scheme Option, or any of the Uri-* Options.
- * **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 proposed 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 10 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 associations also between each client and the proxy, to especially ensure the integrity of that information provided to the proxy (see Section 13.3 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. Further Use Cases

The approach proposed 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 see Section 4 of [I-D.amsuess-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. However, in order for OSCORE to be an applicable security mechanism for this, it has to be terminated at the proxy. That is, it would be required for the client and the proxy to share a dedicated OSCORE Security Context and to use it for protecting their communication leg.

- * 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 proposed 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 endpoint protects a message using first the OSCORE Security Context shared with the origin server, and then the dedicated OSCORE Security Context shared with each of the different chain hops. Once received at a chain hop, a message 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 also 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.

3.1. General Rules on Protecting Options

When a sender endpoint protects an outgoing message by applying the i -th OSCORE layer in sequence, the following CoAP options are also protected, in addition to those already specified as class I or class E in the document defining them.

- * An OSCORE Option which is present as the result of the j -th OSCORE layer immediately previously applied, i.e., $j = (i-1)$. Such an OSCORE Option is protected like an option of class E.
- * Any option such that both the following conditions hold.

1. The option is intended to be consumed by the other OSCORE endpoint X sharing the OSCORE Security Context used for applying the i-th OSCORE layer.
2. The option does not play a role at the other OSCORE endpoint X for correctly processing the message before having removed the i-th OSCORE layer.

Examples of such options are:

- The proxy-related options Proxy-Uri, Proxy-Scheme and Uri-* defined in [RFC7252].
- Listen-To-Multicast-Notifications defined in [I-D.ietf-core-observe-multicast-notifications].
- Multicast-Timeout, Response-Forwarding and Group-ETag defined in [I-D.tiloca-core-groupcomm-proxy].

On the other hand, when applying the i-th OSCORE layer, an option intended to the endpoint X is not protected if it plays a role for removing the i-th OSCORE layer at that endpoint. Examples of such options are:

- * Clearly, and consistently with [RFC8613], the OSCORE option added to the outgoing message as a result of applying the i-th OSCORE layer.
- * The EDHOC option defined in [I-D.ietf-core-oscore-edhoc], to signal to the endpoint X that part of the message payload has to be extracted and used to complete an ongoing execution of the EDHOC key establishment protocol [I-D.ietf-lake-edhoc], before the i-th OSCORE layer can be removed.

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

The recipient endpoint performs the following actions on the received request REQ, depending on which of the following three conditions apply.

- * A - REQ includes visible proxy-related options.

If the endpoint is not configured to be a 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 consumes the proxy-related options and 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.

- * B - REQ does not include proxy-related options and 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.

- * C - REQ does not include proxy-related options and includes an OSCORE Option.

The endpoint decrypts REQ using the OSCORE Security Context indicated by the OSCORE Option, i.e., $REQ^* = \text{dec}(REQ)$. 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 evaluates which of the three conditions (A, B, C) applies to REQ, thus performing again the algorithm defined in this section.

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. Example

TODO: add example with message exchange.

5. 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, both the origin client and the origin server are required 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 now 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.

If the approach defined in [I-D.amsuess-core-cachable-oscore] is used, 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 alternative A 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 is made; 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.

6. Security Considerations

TODO

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 J. Park, "Group OSCORE - Secure Group Communication for CoAP", Work in Progress, Internet-Draft, draft-ietf-core-oscore-groupcomm-14, 7 March 2022, <<https://www.ietf.org/archive/id/draft-ietf-core-oscore-groupcomm-14.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>>.

8.2. 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-04, 6 March 2022, <<https://www.ietf.org/archive/id/draft-amsuess-core-cachable-oscore-04.txt>>.
- [I-D.amsuess-core-transport-indication]
Amsüss, C., "CoAP Protocol Indication", Work in Progress, Internet-Draft, draft-amsuess-core-transport-indication-03, 3 March 2022, <<https://www.ietf.org/archive/id/draft-amsuess-core-transport-indication-03.txt>>.

[I-D.ietf-core-coap-pubsub]

Koster, M., Keranen, A., and J. Jimenez, "Publish-Subscribe Broker for the Constrained Application Protocol (CoAP)", Work in Progress, Internet-Draft, draft-ietf-core-coap-pubsub-09, 30 September 2019, <<https://www.ietf.org/archive/id/draft-ietf-core-coap-pubsub-09.txt>>.

[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-06, 7 March 2022, <<https://www.ietf.org/archive/id/draft-ietf-core-groupcomm-bis-06.txt>>.

[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-03, 7 March 2022, <<https://www.ietf.org/archive/id/draft-ietf-core-observe-multicast-notifications-03.txt>>.

[I-D.ietf-core-oscore-edhoc]

Palombini, F., Tiloca, M., Hoeglund, R., Hristozov, S., and G. Selander, "Profiling EDHOC for CoAP and OSCORE", Work in Progress, Internet-Draft, draft-ietf-core-oscore-edhoc-03, 7 March 2022, <<https://www.ietf.org/archive/id/draft-ietf-core-oscore-edhoc-03.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.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-06, 7 March 2022, <<https://www.ietf.org/archive/id/draft-tiloca-core-groupcomm-proxy-06.txt>>.

[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-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>.

[RFC7641] Hartke, K., "Observing Resources in the Constrained Application Protocol (CoAP)", RFC 7641, DOI 10.17487/RFC7641, September 2015, <<https://www.rfc-editor.org/info/rfc7641>>.

Appendix A. OSCORE-protected Onion Forwarding

TODO: better elaborate on the listed points below.

- * The client can hide its position in the network from the origin server, while still possibly protecting communications end-to-end with OSCORE.
- * Use the method defined in Section 3 to achieve OSCORE-protected onion forwarding, through a chain of proxies (at least three are expected). Every message generated by or intended to the origin client must traverse the whole chain of proxies until the intended other endpoint (typically, the origin server). The chain of proxies has to be known in advance by the client, i.e., the exact proxies and their order in the chain.
- * The typical case addressed in this document considers an origin client that, at most, shares one OSCORE Security Context with the origin server and one OSCORE Security Context with the first proxy in the chain.

If onion forwarding is used, the origin client shares an OSCORE Security Context with the origin server, and a dedicated OSCORE Security Context with each of the proxies in the chain.

- * The origin client protects a request by applying first the OSCORE layer intended to the origin server, then the OSCORE layer intended to the last proxy in the chain, then the OSCORE layer intended to the second from last proxy in the chain and so on, until it applies the OSCORE layer intended to the first proxy in the chain.

Before protecting a request with the OSCORE layer to be consumed by a certain proxy in the chain, the origin client also adds proxy-related options intended to that proxy, as indications to forward the request to (the next hop towards) the origin server.

Other than the actions above from the client, there should be no difference from the basic approach defined in Section 3. Each proxy in the chain would process and remove one OSCORE layer from the received request and then forward it to (the next hop towards) the origin server.

- * The exact way used by the client to establish OSCORE Security Contexts with the proxies and the origin server is out of scope.

However, if EDHOC is used, it is most convenient for the client to run it with the first proxy in the chain, then with the second proxy in the chain through the first one and so on, and finally with the origin server by traversing the whole chain of proxies.

Then, it is especially convenient to use the optimized workflow defined in [I-D.ietf-core-oscore-edhoc] and based on the EDHOC + OSCORE request. This would basically allow the client to complete the EDHOC execution with an endpoint and start the EDHOC execution with the next endpoint in the chain, by means of a single message sent on the wire.

- * Hop-by-hop security has to also be achieved between each pair of proxies in the chain. To this end, two adjacent proxies would better use TLS over TCP than OSCORE between one another (this should be acceptable for non-constrained proxies). This takes advantage of the TCP packet aggregation policies, and thus:
 - As request forwarding occurs in MTU-size bundles, the length of the origin request can be hidden as well.
 - Requests and responses traversing the proxy chain cannot be correlated, e.g., by externally monitoring the timing of message forwarding (which would jeopardize the client's wish to hide itself from anything but the first proxy in the chain).

- * Cacheability of responses can still happen, as per Section 5 and using the approach defined in [I-D.amsuess-core-cachable-oscore].

The last proxy in the chain would be the only proxy actually seeing the Deterministic Request originated by the client and then caching the corresponding responses from the origin server. It is good that other proxies are not able to do the same, thus preventing what might lead to request-response correlation, again opening for localization of the origin client.

- * Possible optimizations along the proxy chains
 - In particular settings involving additional configuration on the client, some proxy in the chain might be a reverse-proxy. Then, such a proxy can be configured to map on one hand the OSCORE Security Context shared with the origin client (and used to remove a corresponding OSCORE layer from a received request to forward) and, on the other hand, the addressing information of the next hop in the chain where to forward the received request to. This would spare the origin client to add a set of proxy-related options for every single proxy in the chain.
 - It is mentioned above to additionally use TLS over TCP hop-by-hop between every two adjacent proxies in the chain. That said:
 - o The OSCORE protection of the request has certainly to rely on authenticated encryption algorithms (as usual), when applying the OSCORE layer intended to the origin server (the first one applied by the origin client) and the OSCORE layer intended to the first proxy in the chain (the last one applied by the origin client).
 - o For any other OSCORE layer applied by the origin client (i.e., intended for any proxy in the chain but the first one), the OSCORE protection can better rely on an encryption-only algorithm not providing an authentication tag (as admitted in the group mode of Group OSCORE [I-D.ietf-core-oscore-groupcomm] and assuming the registration of such algorithms in COSE).
 - o This would be secure to do, since every pair of adjacent proxies in the chain relies on its TLS connection for the respective hop-by-hop communication anyway. The benefit is that it avoids transmitting several unneeded authentication tags from OSCORE.

Acknowledgments

The authors sincerely thank Christian Amsuess, Peter Blomqvist and Goeran Selander for their comments and feedback.

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