Thing-to-Thing Research Group
Internet-Draft
Intended status: Experimental
Expires: 11 July 2026

M. Tiloca
RISE AB

7 January 2026

Distribution of Software Updates with End-to-End Secure Group Communication and Block-Wise Transfer for CoAP
draft-tiloca-t2trg-sw-update-groupcomm-01

Abstract

This document defines a method for efficiently distributing a software update to multiple target devices, by using end-to-end secure group communication over UDP and IP multicast. To this end, the defined method relies on a number of building blocks developed in the Constrained RESTful Environments (CoRE) Working Group of the IETF. Those especially include the Constrained Application Protocol (CoAP), Block-wise transfers for CoAP, and the end-to-end security protocol Group Object Security for Constrained RESTful Environments (Group OSCORE). The method defined in this document is compatible with (but not dependent on) the architecture for software and firmware update developed in the Software Updates for Internet of Things (SUIT) Working Group of the IETF.

Discussion Venues

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

Discussion of this document takes place on the Thing-to-Thing Research Group mailing list (t2trg@irtf.org), which is archived at https://mailarchive.ietf.org/arch/browse/t2trg/.

Source for this draft and an issue tracker can be found at https://gitlab.com/crimson84/draft-tiloca-t2trg-sw-update-groupcomm.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

Throughout the operational phase of their lifecycle, it is vital that devices can effectively receive and install required software (SW) updates. This is important not only in order to add and extend features or to improve performance, but also and especially in order to address and prevent security vulnerabilities. In turn, the distribution of SW updates in itself has to be a secure and efficient process that scales well with the size of the software update and with the number of target devices.

This document defines a method for efficiently distributing a SW update to multiple target devices, by using end-to-end secure group communication over UDP and IP multicast. To this end, the defined method relies on a number of building blocks developed in the Constrained RESTful Environments (CoRE) Working Group of the IETF. Those especially include the Constrained Application Protocol (CoAP) [RFC7252], Block-wise transfers for CoAP [RFC7959], and the end-to-end security protocol Group Object Security for Constrained RESTful Environments (Group OSCORE) [I-D.ietf-core-oscore-groupcomm].

The defined method leverages a CoAP-to-CoAP proxy deployed between the CoAP target devices to update and a CoAP server acting as Distributor of the SW update. The proxy communicates with the Distributor using CoAP over TCP [RFC8323] and retrieves the next-in-line "inner chunk" of the SW update from the Distributor, by using Block-wise Extension for Reliable Transport (BERT) [RFC8323]. A CoAP response originated by the Distributor and conveying an inner chunk is protected end-to-end between the Distributor and the target devices, by using Group OSCORE.

When a target device contacts the proxy for obtaining the latest SW update, the proxy relies on the use of Group OSCORE defined in [I-D.ietf-core-cacheable-oscore]. That is, it retrieves the next-in-line inner chunk from the Distributor if not already available in its cache, and then caches the response that conveys the inner chunk. After that, building on concepts from [I-D.ietf-core-observe-multicast-notifications], the proxy replies to the target device with an error response, informing about the time when it is going to distribute the inner chunk and providing transport-specific information for receiving that inner chunk.

When that time comes, the proxy transmits the inner chunk to all the target devices by further splitting it into smaller "outer chunks", each of which is conveyed by a CoAP response over UDP and IP multicast using Block-wise transfer [RFC7959]. At the end of such transfer, the target devices are allowed to selectively request outer chunks that they have missed for the current inner chunk.

After the proxy declares the transfer of the current inner chunk completed, the process is repeated for the next inner chunk, which the proxy retrieves from the Distributor and transmits to the target devices as above. Eventually, the proxy completes the transfer of the last inner chunk. After that, as a new request comes from a target device to retrieve the latest SW update, the proxy restarts the process by retrieving the first inner chunk and providing it to the target devices.

This document also defines how to counteract an attack against availability that an active adversary could easily perform by manipulating the CoAP responses sent by the proxy to the target devices and conveying the small outer chunks. The attack is neutralized by having a short checksum value computed by the proxy and included in such responses. By recomputing and verifying the checksum, target devices can thus promptly detect a possible manipulation of an outer chunk and discard the response conveying it as invalid.

The method defined in this document is compatible with (but not dependent on) the architecture for SW and firmware update specified in [RFC9019] and developed in the Software Updates for Internet of Things (SUIT) Working Group of the IETF.

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], also used for group communication [I-D.ietf-core-groupcomm-bis] and over TCP [RFC8323].

* The CoAP extensions Observe [RFC7641] and Block-wise [RFC7959].

* The security protocols OSCORE [RFC8613] and Group OSCORE [I-D.ietf-core-oscore-groupcomm], and the use of the latter to enable cacheability of protected CoAP responses [I-D.ietf-core-cacheable-oscore].

* The Concise Data Definition Language (CDDL) [RFC8610], Concise Binary Object Representation (CBOR) [RFC8949], and CBOR Object Signing and Encryption (COSE) [RFC9052][RFC9053].

This document also relies on the following terminology:

* Image: a binary that can contain the complete SW of a device, or part of it. The image might be in turn structured in multiple images, and the corresponding SW might specifically be a firmware. Also, it might consist of a differential update in the interest of performance.

* Manifest: a collection of metadata about the image and author. The manifest is generated by the author and protected against modifications.

* Author: the entity that generates the image and the associated manifest.

* Device: target of a SW update as intended to obtain and consume an image.

* Distributor: the entity that distributes the image and the associated manifest on behalf of the author.

* Manifest resource: a resource hosted at the Distributor, with a manifest as its representation. This resource is observable [RFC7641].

* Image resource: a resource hosted at the Distributor, with an image as its representation.

2. Building Blocks

The distribution method defined in this document largely relies on a number of building blocks that are summarized in the following subsections.

2.1. CoAP

CoAP is a web-transfer protocol specified in [RFC7252]. It relies on the client-server message exchange model and builds on the Representational State Transfer (REST) paradigm for accessing and manipulating resource representations hosted at a server. CoAP messages can be very compact and, besides a payload and a mandatory header, can include CoAP options that indicate the additional use of protocol extensions and optional features. The mandatory header includes a variable-length Token field whose value is used to associate a response with a corresponding request. CoAP is typically transported over UDP, but it can also be used over reliable transports such as TCP and WebSockets [RFC8323].

CoAP natively supports proxies deployed between origin client endpoints and origin server endpoints. Main reasons to deploy proxies include: relaying messages between origin endpoints that cannot directly interact with one another; caching response messages to serve requests from origin clients more efficiently and avoiding repeatedly interacting with origin servers; and performing protocol translation across different communication legs. Proxy operations for CoAP are detailed in Section 5.7 of [RFC7252].

CoAP also natively supports group communication [I-D.ietf-core-groupcomm-bis]. That is, an origin client can send a single group request targeting multiple recipient servers at once, e.g., over UDP and IP multicast. The servers can individually reply to that group request by sending their unicast responses, each of which is associated by Token value with the same group request.

2.2. CoAP Observe

Observe is an extension for CoAP specified in [RFC7641]. When using Observe, a client accesses a resource at a server by additionally requesting to be registered as an observer for that resource. A successful response from the server can confirm the client to have become a registered observer. In such a case, following updates in the resource representation will result in the server sending notification responses to the client. Each of such notification responses conveys the current resource representation and is associated by Token value with the request originally sent by the client to start the observation. This extension relies on the CoAP Observe option included in the original observation request and in each notification response.

2.3. CoAP Block-wise Transfer

Block-wise is an extension for CoAP specified in [RFC7959]. With the intent to avoid message fragmentation at lower layers, Block-wise enables message senders to split their large-size application data to transmit (body) into multiple, smaller data units referred to as blocks. This process occurs at the CoAP layer and results in sequentially sending each block of the same body as the payload of a different CoAP message. The message recipient can re-assemble the original body once all the corresponding blocks are received. This extension relies on the CoAP Block1 and Block2 options. Those are appropriately included in request and response messages to either describe the block conveyed in the present message or to control the Block-wise transfer process.

For the case where CoAP is transported over reliable transports such as TCP, [RFC8323] also specifies Block-wise Extension for Reliable Transport (BERT). This still relies on the same CoAP Block1 and Block2 options as also explicitly indicating the use of BERT. In practice, a BERT message conveys in its payload one or more blocks of size 1024 bytes, with the possible exception of the BERT message conveying the last block that can have a smaller size.

2.4. OSCORE

Object Security for Constrained RESTful Environments (OSCORE) is a security protocol specified in [RFC8613]. OSCORE protects CoAP messages end-to-end between the origin endpoint producing application data and the other origin endpoint consuming that data.

To this end, it takes as input an outgoing CoAP message and produces as output an OSCORE-protected CoAP message that includes the CoAP OSCORE option. When receiving the OSCORE-protected message, the

recipient endpoint relies on the information in the OSCORE option to attempt decrypting and verifying the message. By using CBOR [RFC8949] for data encoding and COSE [RFC9052] for security operations, OSCORE has a lightweight impact on message sizes and performance.

OSCORE ensures end-to-end confidentiality, integrity, and source authentication of messages, as well as replay protection. Each OSCORE-protected response is cryptographically bound to the corresponding request, also when Observe [RFC7641] is used and thus multiple notification responses are bound to the same observation request.

These security properties hold also in the presence of (untrusted) proxies deployed between the two OSCORE endpoints. Since OSCORE selectively protects different parts of a CoAP message, it hides as much as possible from possibly deployed proxies, while keeping the proxies able to perform their intended tasks. Furthermore, since the OSCORE processing of a CoAP message results in another CoAP message, OSCORE is independent of the specific transport underlying CoAP and used to transport CoAP messages (e.g., UDP or TCP). Therefore, OSCORE works wherever CoAP works.

In order to use OSCORE, two CoAP endpoints have to first establish an OSCORE Security Context including the necessary parameters and keying material. OSCORE is agnostic of how exactly the Security Context is established. A possible way is the lightweight authenticated key exchange protocol Ephemeral Diffie-Hellman Over COSE (EDHOC) [RFC9528].

2.5. Group OSCORE

Group OSCORE is a security protocol specified in [I-D.ietf-core-oscore-groupcomm]. By building on OSCORE [RFC8613] and extending its construct, Group OSCORE protects CoAP messages end-to-end between endpoints that use CoAP in a group communication setup [I-D.ietf-core-groupcomm-bis].

Also by relying on CBOR [RFC8949] and COSE [RFC9052], Group OSCORE ensures end-to-end confidentiality, integrity, and source authentication of messages, as well as replay protection. All the protected responses originated by different servers and corresponding to the same group request are cryptographically bound to such request.

Messages protected with Group OSCORE also include a CoAP OSCORE option that indicates the recipient endpoint how to attempt decrypting and verifying an incoming message. Like OSCORE, Group OSCORE works wherever CoAP works, also in the presence of proxies.

Group OSCORE provides two modes for protecting messages, allowing to choose the mode to use on a per-message basis. The main difference between the two modes is about the way used to ensure source authentication of the protected message:

* When using the group mode (see Section 7 of [I-D.ietf-core-oscore-groupcomm]), the message is first encrypted with keying material that every group member can derive, and then is signed by using the private key of the sender endpoint. The resulting signature is placed at the end of the CoAP payload of the protected message and then separately encrypted in order to contrast tracking of endpoints across different groups. As a result, all group members are able to decrypt the message and to verify that the message sender is the alleged group member. The group mode is typically used to protect requests that are sent to the whole group, i.e., that are intended to all CoAP servers in the group.

* When using the pairwise mode (see Section 8 of [I-D.ietf-core-oscore-groupcomm]), the message is protected by using an authenticated encryption algorithm. The encryption key to use is derived from the asymmetric authentication credentials of the sender endpoint and the single recipient endpoint, by means of a static-static Diffie-Hellman key derivation performed locally. As a result, only the intended recipient is able to decrypt the message and to verify that the message sender is the alleged group member. The pairwise mode is typically used to protect a response that is individually sent by a server in the group.

In order to use Group OSCORE, a CoAP endpoint has to join an OSCORE group and effectively become a member. The join process is typically assisted by a Group Manager entity that is responsible for the OSCORE group and that might enforce access control when deciding whether to admit new endpoints requesting to join the group. As a result of a successful join process, a CoAP endpoint obtains the necessary parameters and keying material to set up a Group OSCORE Security Context and consequently use Group OSCORE with the other group members. A possible realization of Group Manager is specified in [I-D.ietf-ace-key-groupcomm-oscore], where the join process is based on the ACE framework for authentication and authorization in constrained environments [RFC9200].

2.6. Observe multicast notifications

According to the original use of CoAP Observe [RFC7641], two CoAP clients interested in registering a resource observation at a server will yield two distinct observations.

However, some applications involve multiple clients that are all interested in observing the same resource at the same server. While the original approach remains usable, an alternative and more scalable approach is specified in [I-D.ietf-core-observe-multicast-notifications].

This second approach takes advantage of group communication for CoAP and relies on the CoAP server to initiate a group observation at itself, e.g., upon receiving a first traditional observation request from a client. To this end, the server composes a cosmetic phantom observation request and sends it to itself without hitting the wire, in order to start the corresponding group observation.

When a client sends a traditional observation request targeting that resource, the server replies with an informative error response conveying: i) the phantom observation request associated with the group observation; and ii) transport-specific information that is needed for receiving the notification responses in the group observation. That information includes the CoAP Token value used for the phantom observation request and the multicast address where observe notifications will be sent to. This effectively aligns all the interested clients to the same common knowledge required for participating in the group observation and receiving the multicast notification responses.

Consequently, when the representation of the observed resource at the server changes, the server sends a single observe notification response over UDP and IP multicast, thus targeting all the clients that are taking part in the group observation. All such observe multicast notifications include the same CoAP Token value used in the phantom observation request. The recipient clients have been instructed on how to receive such observe multicast notifications when obtaining the individual informative error response from the server.

The observe multicast notifications can be protected end-to-end between the server and the clients, by using Group OSCORE in group mode [I-D.ietf-core-oscore-groupcomm]. Since the server uses Group OSCORE also to protect the phantom observation request that started the group observation, all the observe notifications in the group observation are cryptographically bound to such phantom observation request, and thereby verifiable to pertain to the group observation in question.

2.7. Cacheable OSCORE

It is typically possible to rely on a deployed proxy to store in its cache some classes of CoAP responses received from origin servers. That allows the proxy to quicker serve later requests from CoAP clients that produce a cache hit, by replying with the cached response instead of obtaining a new response from the origin server.

If CoAP messages are protected with OSCORE [RFC8613] or Group OSCORE [I-D.ietf-core-oscore-groupcomm], effective caching of responses is not achievable anymore. That is, even if two clients wish to send the same plain CoAP request, the two resulting protected requests will be different and thus will not result in a cache hit at the proxy. Therefore, separately for each of the two clients, the proxy has to retrieve a new response from the origin server. The same holds also when considering the same client that wishes to send the same plain CoAP request at different points in time.

In order to overcome this limitation, the approach defined in [I-D.ietf-core-cacheable-oscore] builds on Group OSCORE and restores the effective cacheability of protected responses.

According to this approach, any client in the OSCORE group can also act as a specific "Deterministic Client" and use the corresponding keying material known to all the group members. When acting as Deterministic Client, any two clients in the group that protect the same plain CoAP request will produce the same protected "Deterministic Request", which is thereby usable to produce a cache hit at a caching proxy.

The construct that makes this possible relies on the following design points: i) Deterministic Requests are computed by using a variation of the pairwise mode of Group OSCORE; ii) responses to a Deterministic Request are protected by using the group mode of Group OSCORE.

This approach restores cacheability of protected responses while sacrificing some security properties of the protected Deterministic Requests, which in fact are replays and have no source

authentication. However, this is deemed acceptable and harmless for the particular, narrow scope targeted by this approach, whose applicability is limited to content retrieval through read-only requests that are safe in the REST sense. Servers that want to be even more conservative can additionally limit themselves to accept only Deterministic Requests that target specific resources, or even that match byte-by-byte with Deterministic Requests that are known in advance.

3. Architecture

This section describes the architecture considered in the rest of this document when defining the method for distributing SW updates.

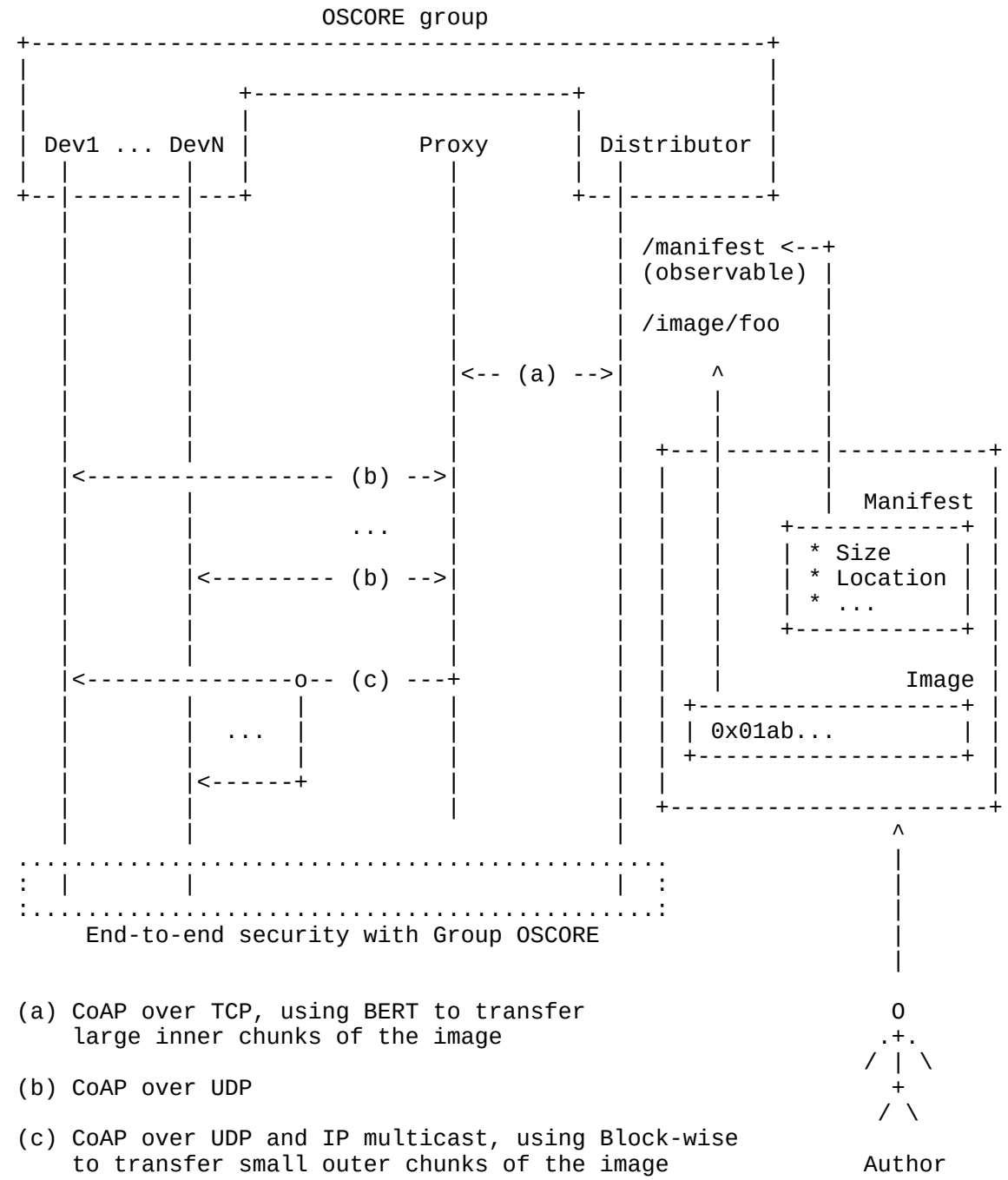

Figure 1: Architecture Overview

As shown in Figure 1, the architecture consists of the following actors:

* The Author is responsible for building and issuing the new version of the image, together with the corresponding manifest.

  The manifest MUST be signed by the author and MUST include at least the following information:

  - The total size of the image.

  - A location URI, i.e., the URI of the image resource at the Distributor from where it is possible to retrieve the image.

  - The Author’s digital signature computed over (a digest of) the image.

  A possible manifest format that can be used is specified in [RFC9124], with its corresponding CBOR-based serialization format specified in [I-D.ietf-suit-manifest].

* The Distributor acts as a CoAP server, hosts the manifest and the image issued by the Author, and distributes those to the target Devices via the Proxy, as described in Section 4.

  At the Distributor, the following applies separately for each SW component X that can be updated.

  - The Distributor hosts one observable manifest resource. At any point in time, the representation of the manifest resource is the latest manifest issued for X.

  - For each different image released for X, the Distributor hosts one different image resource, whose representation is that image of X.

    More generally, the Distributor stores any two released images of any SW component as representations of two different image resources, hence identified by different URIs. For example, the URIs can differ as to their path components or query components.

    With reference to the manifest available as current representation of the manifest resource for X, the location URI specified within that manifest identifies the image resource whose representation is the image of X that has been released latest and is associated with that manifest.

For simplicity, the architecture in Figure 1 refers to a Distributor that is responsible for a single SW component, whose latest manifest is stored as the representation of the manifest resource at /manifest. Within that manifest, the location URI for the image associated with the manifest identifies the resource at /image/foo, whose representation is the latest released image in question.

The criteria for retaining and eventually deleting image resources for old images are to be defined by application policies.

* The Proxy is responsible for retrieving the manifest and the image from the Distributor and for practically sending those to the target Devices, as described in Section 4.

  The Proxy communicates with the Distributor using CoAP over TCP [RFC8323]. In particular, the Proxy retrieves the image from the Distributor by using BERT [RFC8323]. Each of such BERT responses from the Distributor includes exactly one block and conveys an "inner chunk" of the image.

  The Proxy communicates with the target Devices using CoAP over UDP. Specifically when providing the Devices with the image, the Proxy sends CoAP responses over UDP and IP multicast, as described in Section 4.3. Each of such responses uses Block-wise transfer for CoAP [RFC7959] and conveys an "outer chunk" of the currently transferred inner chunk of the image.

* The Devices obtain SW updates of interest from the Distributor, by directly interacting with the Proxy, as described in Section 4.

  At a high-level, a Device first learns about the availability of a new image for a SW component of interest, by receiving the corresponding manifest in an Observe notification response pertaining to the manifest resource at the Distributor.

  After that, each interested Device independently enrolls in a distribution process driven by the Proxy. During that process, the Device receives the image as a set of inner chunks that the Distributor provides to the Proxy.

  In particular, the Proxy provides the Devices with the inner chunks by further splitting each of those into smaller outer chunks, which constitute the actual unit of distribution. Each outer chunk is conveyed by a Block-wise response sent by the Proxy to all the enrolled target Devices over UDP and IP multicast.

Once received all the inner chunks, the Devices rebuild the actual image and perform further checks against the corresponding manifest before consuming the image.

Secure communication is ensured end-to-end between the Distributor and the Devices. To this end, the Distributor and the Devices have to be members of the same OSCORE group, and therefore able to protect their exchanged message using the security protocol Group OSCORE [I-D.ietf-core-oscore-groupcomm]. In particular, the following applies:

* Every response originated by the Distributor is protected by using the group mode of Group OSCORE (see Section 7 of [I-D.ietf-core-oscore-groupcomm]).

* Every request originated by the Devices and targeting a resource at the Distributor are Deterministic Requests protected with Group OSCORE, according to the construct defined in [I-D.ietf-core-cacheable-oscore].

The process of joining the OSCORE group is driven by a responsible Group Manager. For example, it can rely on the realization of the Group Manager specified in [I-D.ietf-ace-key-groupcomm-oscore], where the join process is based on the ACE framework for authentication and authorization in constrained environments [RFC9200].

The method defined in this document is agnostic of such a join process and related provisioning of keying material for Group OSCORE. At the same time, it does benefit from the Group Manager entity taking care of the provisioning of keying material. In particular, the Distributor does not need to know the exact Devices that are members of the OSCORE group, or any keying material or authentication credential related to those. This allows the Distributor to seamlessly distribute SW updates over time, without needing to be aware of possible membership changes within the OSCORE group.

As ultimately in charge with the membership of the OSCORE Group, the responsible Group Manager has to appropriately admit/refuse new members and evict current members as needed, thus ensuring that only Devices that are supposed to obtain a SW update can effectively do so.

4. Distribution Process

This section describes the distribution process in detail, building on the architecture presented in Section 3. After an overview of its design goals in Section 4.1, the process is presented as composed of two main parts.

The first part is described in Section 4.2 and concerns the advertisement of a newly available SW update, by providing target Devices with the corresponding manifest through Observe notification responses.

The second part is described in Section 4.3 and concerns the actual distribution of the image to the target Devices. This is based on the Distributor providing large inner chunks of the image to the Proxy, which further splits each of those into smaller outer chunks that are sent to the target Devices.

For simplicity, but with no loss of generality, the following considers a Distributor as responsible for a single SW component and only two Devices as target of updates for that SW component.

4.1. Design Goals

The distribution method builds on the following design goals.

* Ensure end-to-end secure communication between the Devices and the Distributor.

  This goal is addressed by using Group OSCORE [I-D.ietf-core-oscore-groupcomm] to protect the messages exchanged by the Devices and the Distributor through the Proxy.

  The use of Group OSCORE is facilitated by a Group Manager that provides the necessary parameters and keying material to the intended group members, which is therefore not a concern for the Distributor.

* Limit interactions and exchanges with the Distributor.

  This goal is achieved by means of the Proxy deployed between the Devices and the Distributor and specifically relying on:

  - Large-size inner chunks that the Distributor provides to the Proxy, by using BERT [RFC8323].

  - Cacheable protected responses that are cached at the Proxy as per the construct defined in [I-D.ietf-core-cacheable-oscore], thereby sparing the retrieval of a cached inner chunk from the Distributor.

* Limit interactions and exchanges with the Devices.

  This goal is achieved by means of the Proxy deployed between the Devices and the Distributor and specifically relying on:

- Cacheable protected responses that are cached at the Proxy as per the construct defined in [I-D.ietf-core-cacheable-oscore], thereby enabling the Proxy to quicker serve Devices' requests from its cache.

- Responses sent by the Proxy to the Devices over UDP and IP multicast, building on concepts defined for observe multicast notifications in [I-D.ietf-core-observe-multicast-notifications].

- Transferring outer chunks from the Proxy to the Devices in an uninterrupted, back-to-back fashion, thus avoiding the downsides of a synchronous, lock-step process based on the original Block-wise transfer [RFC7959].

* Accommodate Devices over constrained network links.

  This goal is achieved by means of the Proxy using small outer chunks (e.g., 64 bytes each in size) as the actual unit of distribution, when providing a SW update to the Devices.

4.2. Release and Notification of SW Update

This part of the process is devoted to inform the Devices about a newly released SW update as available at the Distributor.

A Device interested in obtaining SW updates for that SW component has to know the URI of the corresponding manifest resource at the Distributor and be able to access that resource through the Proxy.

In order to be informed of a newly available SW update, the Device performs the following actions.

1. The Device composes an observation request [RFC7641] targeting the manifest resource at the Distributor.

2. The Device protects the observation request from Step 1 with Group OSCORE, specifically using the construct defined in [I-D.ietf-core-cacheable-oscore] and therefore producing a Deterministic Request.

3. The Device sends the Deterministic Request from Step 2 to the Proxy.

If the Deterministic Request does not produce a cache hit at the Proxy (e.g., as the first of this kind sent by any Device), the Proxy forwards the Deterministic Request to the Distributor, thereby registering itself as the actual observer at the Distributor. The Proxy caches the corresponding successful response from the Distributor and forwards it back to the Device.

Otherwise, the Proxy does not interact with the Distributor, but instead promptly replies to the Device, by using the response stored in the identified cache entry.

4. Upon receiving a first successful notification response, the Device obtains the latest manifest and is subscribed for automatically receiving future manifests released for the same SW component.

When releasing a new SW update, the Author uploads the corresponding manifest and image on the Distributor. As discussed in Section 3, those become available at the manifest resource and at a new, dedicated image resource hosted by the Distributor, respectively. After that, the Distributor sends a notification response to the observer Proxy, conveying the newly released manifest.

The Proxy stores the notification response in the cache entry that it uses to store responses for that observation, by overwriting the current content of the entry. Finally, the Proxy forwards that notification response to the observer Devices. Upon receiving the notification response, the observer Devices obtain the new manifest, thus becoming aware of the new SW update and of the corresponding image resource at the Distributor.

In setups where multiple Proxies are deployed and each assists a different set of Devices, the Distributor can use as a possible optimization the approach defined in [I-D.ietf-core-observe-multicast-notifications], whose use in network setups that leverage proxies is described in [I-D.ietf-core-multicast-notifications-proxy]. Consequently, following the release of a new SW update, the Distributor sends a single observe notification response over UDP and IP multicast, thereby providing all the observer Proxies at once with the latest manifest corresponding to the newly released SW update.

Figure 2 shows an example of message exchange where the two Devices obtain the latest manifest from the Distributor through the Proxy.

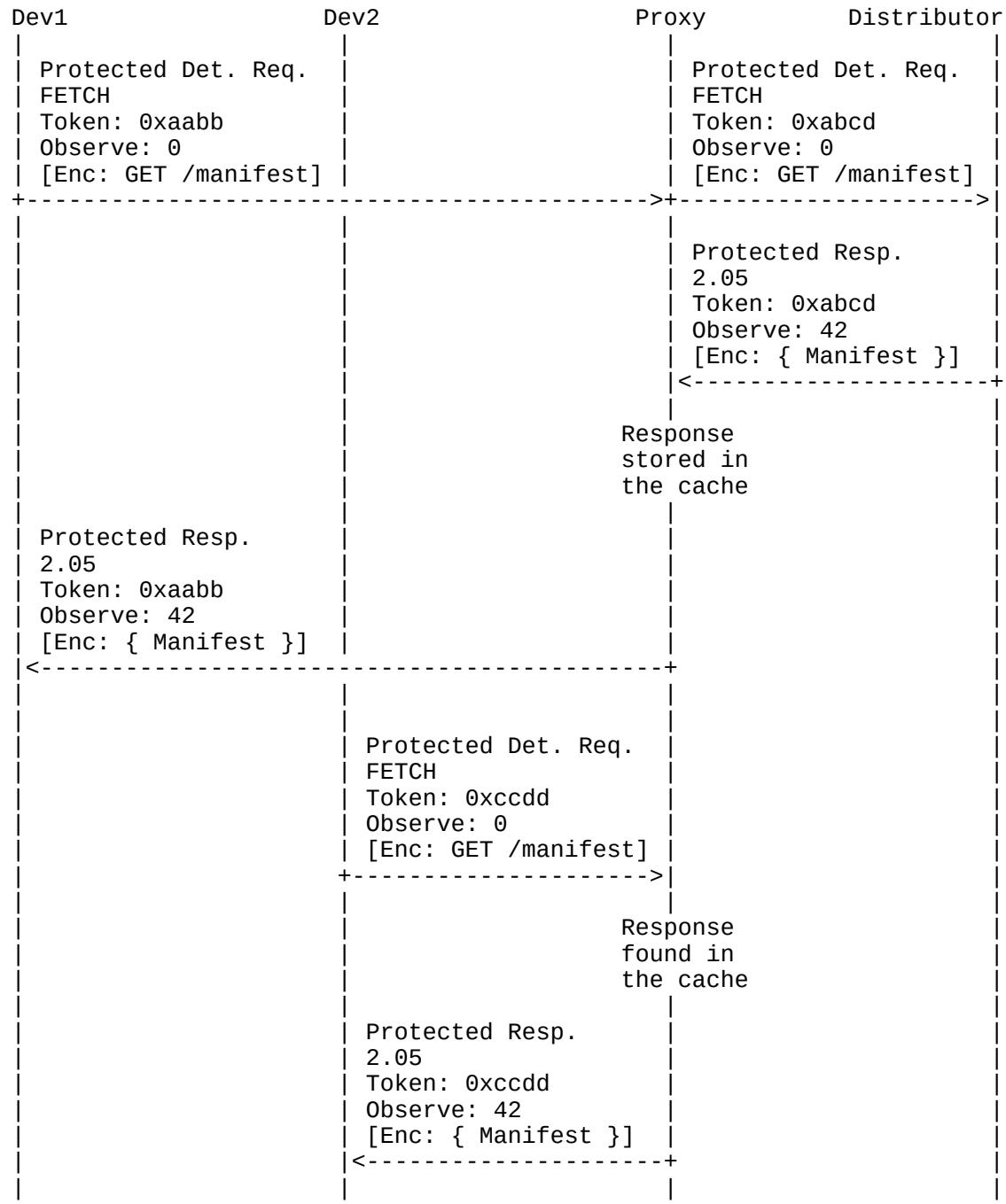

Figure 2: Example of Notification of SW Update

4.3. Distribution of SW Update

The distribution of an image to target Devices is driven by the Proxy and organized into epochs, each of which is in turn composed of different phases, as shown in Figure 3.

Each epoch is devoted to transferring exactly one inner chunk of the image, by providing the target Devices with all the corresponding outer chunks.

After completing the epoch during which the last inner chunk of the image has been transferred, the next epoch is devoted to transfer again the first inner chunk of the image, i.e., the transfer of the whole image is repeated.

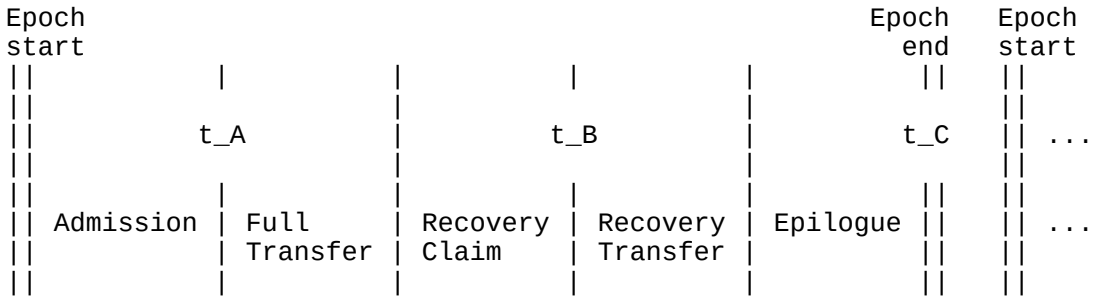

Figure 3: Overview of a Transfer Epoch and its Phases

The following describes in detail the operations performed by a Device and by the Proxy during the transfer of an image.

4.3.1. Device Operations

Following the reception of a manifest as described in Section 4.2, a Device can ask the Proxy to distribute the next inner chunk of the image.

To this end, the Device sends to the Proxy a protected Deterministic Request as defined in [I-D.ietf-core-cacheable-oscore]. The Deterministic Request is computed by using the Group OSCORE Security Context shared between the Device and the Distributor. The original unprotected CoAP request is such that:

* The request method is GET.

* The target is the image resource at the Distributor.

Once produced the protected Deterministic Request, the Device includes the Block2 option as an outer option intended for the Proxy. Its value specifies NUM = 0, M = 0, and SZX = 2 (i.e., a block size of 64 bytes).

When sending such a request, the Device might not know what the current phase of the current epoch is at the Proxy, or what inner chunk is meant to be transferred in the current epoch. As defined in Section 4.3.2, the Proxy guides the Devices about the next actions to take, by providing information on the current status of the image transfer. From a high-level point of view, the following applies to each epoch:

* A Device can enroll in the reception of the inner chunk for the current epoch, by sending its Deterministic Request during the "Admission" phase. Consequently, the Device obtains instructions for receiving that inner chunk, which the Proxy distributes as split into corresponding outer chunks during the immediately following "Full Transfer" phase.

* A Device can ask the Proxy to re-send an outer chunk that was not correctly received (e.g., it was lost in transmission or got corrupted), by sending the Deterministic Request during the "Recovery Claim" phase and indicating the required outer chunk by means of the NUM field of the Block2 outer option. Consequently, the Device obtains instructions for receiving the re-sent outer chunk, which the Proxy distributes during the immediately following "Recovery Transfer" phase.

* If a Device has not enrolled during the "Admission" phase but attempts to enroll on-the-fly later on during the epoch, that Device will be instructed to enroll again at the next epoch.

A Device completes the retrieval of an inner chunk once it has received all the corresponding outer chunks from the Proxy, rebuilt the inner chunk from those, and successfully verified and decrypted the rebuilt CoAP response protected with Group OSCORE and conveying the inner chunk.

A Device completes the retrieval of the whole image once it has received all the corresponding inner chunks from the Proxy. The Device is able to simply verify whether that holds, by checking the cumulated size of the obtained inner chunks against the total size of the image that is specified within the manifest.

If required to, the Device is also able to correctly reorder the obtained inner chunks. To this end, the Device relies on the indication provided during each epoch by the Proxy through the

parameter "progress_indicator", which is included in responses sent by the Proxy during the "Admission" and "Recovery Claim" phases. As detailed in Section 4.3.2, the parameter indicates the index of the inner chunk transferred in the current epoch.

4.3.2. Proxy Operations

The following describes the operations performed by the Proxy.

4.3.2.1. Kick-Off

If the Proxy receives a Deterministic Request targeting the Distributor and that does not produce a cache hit, the Proxy performs the following steps.

For the considered Deterministic Requests, this occurs only if the Proxy has not distributed any inner chunk of the image yet, i.e., it has not retrieved any inner chunk of the image from the Distributor.

1. In the Deterministic Request, replace the value of the Block2 option so that NUM = 0, M = 0, and SZX = 7 (i.e., BERT is used).

2. Forward the Deterministic Request to the Distributor over TCP.

3. Once received a corresponding successful 2.05 (Content) response from the Distributor, create a cache entry ENTRY and store the response in ENTRY as the inner chunk to transfer in the first epoch.

   Note that the value of the Block2 option in the response specifies NUM = 0 and SZX = 7 (i.e., BERT is used).

   Before storing the response in ENTRY, the Proxy MUST remove from the response the Pre-OSCORE-Data option and the associated CBOR data item prepended to the OSCORE ciphertext in the CoAP payload, which were added by the Distributor (see Section 5.6).

4. Start an epoch with "Admission" as its current phase and associate it with ENTRY.

5. Initialize an unsigned integer INNER_INDEX to 0 and associate it with ENTRY.

6. Determine the time t_A when the immediately following "Full Transfer" phase will start. Start a timer with expiration set at t_A.

7. The Proxy performs Step 2 in Section 4.3.2.2, thus enrolling the Device that has sent the Deterministic Request in the upcoming transfer scheduled for the immediately following "Full Transfer" phase.

4.3.2.2. "Admission" Phase

If the Proxy receives a Deterministic Request targeting the Distributor and that produces a cache hit for the cache entry ENTRY associated with an epoch with current phase "Admission", the Proxy performs the following steps.

1. If ENTRY is storing the inner chunk to transfer in the current epoch, then move to Step 2. Otherwise, perform the following steps.

   * In the Deterministic Request, replace the value of the Block2 option so that NUM = INNER_INDEX, M = 0, and SZX = 7 (i.e., BERT is used).

   * Forward the Deterministic Request to the Distributor over TCP.

   * Once received a corresponding successful 2.05 (Content) response from the Distributor, store the response in ENTRY as the inner chunk to transfer in the current epoch, by overwriting the current content of ENTRY.

     Note that the value of the Block2 option in the response specifies NUM = INNER_INDEX and SZX = 7 (i.e., BERT is used).

     Before storing the response in ENTRY, the Proxy MUST remove from the response the Pre-OSCORE-Data option and the associated CBOR data item prepended to the OSCORE ciphertext in the CoAP payload, which were added by the Distributor (see Section 5.6).

   * Determine the time t_A when the immediately following "Full Transfer" will start. Start a timer with expiration set at t_A.

   * Move to Step 2.

2. Reply by sending a "Hold-on Response", i.e., a 5.03 (Service Unavailable) error response.

This is specifically an informative response defined in Section 4.2 of [I-D.ietf-core-observe-multicast-notifications], with Content-Format set to "application/informative-response+cbor" (see Section 14.2 of [I-D.ietf-core-observe-multicast-notifications]).

The payload of this informative response is a CBOR map that MUST include the following parameters.

* "tp_info", which is defined in [I-D.ietf-core-observe-multicast-notifications]. This parameter specifies the transport-specific information required to correctly receive CoAP responses over UDP and IP multicast, during the immediately following "Full Transfer" phase in this epoch. Each of those responses will convey one outer chunk of the inner chunk distributed in this epoch.

  Per Section 4.2.1 of [I-D.ietf-core-observe-multicast-notifications], the "tp_info" parameter specifies addressing information as Constrained Resource Identifiers (CRIs) [I-D.ietf-core-href].

  Furthermore, for each epoch, the "tp_info" parameter MUST specify a different Token value to use for the multicast responses. The same Token value MUST NOT be reused in any other epoch of the current transfer of the image or in any epoch of the two following transfers of the same image.

* "next_not_before", which is defined in [I-D.ietf-core-observe-multicast-notifications]. This parameter specifies the amount of seconds that will minimally elapse before the "Full Transfer" phase of this epoch starts. Such a value MUST NOT result in indicating that the "Full Transfer" phase starts before t_A.

* "progress_indicator", with value a numeric indication of progress, encoded as a CBOR unsigned integer. This parameter is defined in this document and registered in Section 8.2.

  This parameter enables clients to unambiguously interpret, reorder, and process the content that will be sent per the information specified in the "tp_info" parameter, e.g., if that content is split into parts identifiable by an index value.

  When used in this document, the "progress_indicator" parameter MUST encode the current value of INNER_INDEX, thereby identifying the inner chunk that is transferred in this epoch.

A Device receiving the "Hold-on Response" can set itself up for receiving the inner chunk during the immediately following "Full Transfer" phase of this epoch. In particular, the Device becomes aware that the "Full Transfer" phase is scheduled to start not before the amount of seconds indicated by the "next_not_before" parameter, that the inner chunk to be transferred is the one with index INNER_INDEX indicated by the "progress_indicator" parameter, and that the responses conveying the corresponding outer chunks will be sent per the information specified in the "tp_info" parameter.

This phase finishes when the time t_A is reached and the related timer expires. After that, the epoch moves to the "Full Transfer" phase (see Section 4.3.2.3).

4.3.2.3. "Full Transfer" Phase

When this phase starts, the Proxy considers the response stored in the cache entry ENTRY associated with the current epoch. Such a response is the inner chunk to distribute in this phase, by using Block-wise transfer [RFC7959] to further split the inner chunk into smaller outer chunks.

In particular, the i-th outer chunk is a CoAP response such that:

* It retains the same CoAP header of the inner chunk response, with the following exceptions:

  - The Message ID has a new value determined by the Proxy;

  - The message type MUST be Non-confirmable; and

  - The Token Length and Token fields MUST specify the length and the value of the Token to use for the multicast responses to send in this epoch, respectively. The Token value was specified within the "tp_info" parameter of the "Hold-on Response" that was sent during the "Admission" phase of this epoch (see Section 4.3.2.2).

* It MUST include a Block2 outer option, whose value specifies NUM = i (i.e., the index of the present outer chunk) and SZX = 2 (i.e., a block size of 64 bytes).

* It MUST include the Checksum option defined in Section 5.3, with value a checksum computed by the Proxy on the outer chunk, as defined in Section 5.

Before distributing the outer chunks, the Proxy determines the time t_C when the current epoch is expected to end. This takes into account the expected duration of the current "Full Transfer" phase and of the immediately following "Recovery Claim" and "Recovery Transfer" phases.

Then, the Proxy sequentially sends the outer chunks to the enrolled Devices, according to their indexes sorted in ascending order. That is, the Proxy first sends the outer chunk with i = 0, then continues with the outer chunk with i = 1, and so on until all the outer chunks have been sent. In the last outer chunk, the value of the Block2 outer option specifies M = 0.

Each outer chunk is transmitted as a response over UDP and IP multicast, according to the addressing information specified within the "tp_info" parameter of the "Hold-on Response" that was sent during the "Admission" phase of this epoch (see Section 4.3.2.2).

This phase finishes when the Proxy has sent all the outer chunks corresponding to the inner chunk of this epoch. After that, the epoch moves to the "Recovery Claim" phase (see Section 4.3.2.4).

If the Proxy receives a Deterministic Request targeting the Distributor and that produces a cache hit for the cache entry ENTRY associated with an epoch with current phase "Full Transfer", the Proxy MUST reply with a 5.03 (Service Unavailable) error response that has no payload. The error response MUST include the Max-Age option, with value the amount of seconds that will minimally elapse before the current epoch ends. The value of the Max-Age option MUST NOT result in indicating that the current epoch ends before t_C.

4.3.2.4. "Recovery Claim" Phase

When this phase starts, the Proxy creates a set of unsigned integers, namely RECOVERY_INDEXES, and initializes it as empty.

Also, the Proxy determines the time t_B when the immediately following "Recovery Transfer" phase will start. This takes into account the expected duration of the current "Recovery Claim" phase and the already determined time t_C when the current epoch is expected to end. In particular, the time t_B MUST NOT be after the time t_C. Finally, the Proxy starts a timer with expiration set at t_B.

If the Proxy receives a Deterministic Request targeting the Distributor and that produces a cache hit for the cache entry ENTRY associated with an epoch with current phase "Recovery Claim", the Proxy performs the following steps.

1. The Proxy considers the received Deterministic Request and in particular its Block2 outer option. The value specified by the NUM field of the option value is added to RECOVERY_INDEXES if it is not included there already.

2. The Proxy replies by sending a 5.03 (Service Unavailable) error response.

   This is specifically an informative response defined in Section 4.2 of [I-D.ietf-core-observe-multicast-notifications], with Content-Format set to "application/informative-response+cbor" (see Section 14.2 of [I-D.ietf-core-observe-multicast-notifications]).

   The payload of this informative response is a CBOR map that MUST include the following parameters.

   * "tp_info", which is defined in [I-D.ietf-core-observe-multicast-notifications]. This parameter provides the same information as in the "Hold-on Response" sent during the "Admission" phase of the current epoch (see Step 2 of Section 4.3.2.2), with the difference that it includes only server-side information and it MUST NOT include client-side information.

     That is, the parameter still specifies server-side addressing information related to the Proxy as the sender of multicast responses. However, the parameter MUST NOT specify addressing information as to where the multicast responses are sent or the Token value used for those in this epoch.

     Consequently, only Devices that participated in the immediately previous "Full Transfer" phase and missed some outer chunks will participate in the immediately following "Recovery Transfer" phase. Instead, other Devices will be pointed to the start of the next epoch, according to what is specified by the Max-Age option (see below).

   * "next_not_before", which is defined in [I-D.ietf-core-observe-multicast-notifications]. This parameter specifies the amount of seconds that will minimally elapse before the "Recovery Transfer" phase of this epoch starts. Such a value MUST NOT result in indicating that the "Recovery Transfer" phase starts before t_B.

   * "progress_indicator", with value a numeric indication of progress, encoded as a CBOR unsigned integer. This parameter is defined in this document and registered in Section 8.2.

Like in the "Hold-on Response" sent during the "Admission" phase of the current epoch (see Step 2 of Section 4.3.2.2), the "progress_indicator" parameter MUST encode the current value of INNER_INDEX, thereby identifying the inner chunk that is transferred in this epoch.

Furthermore, the error response MUST include the Max-Age option, with value the amount of seconds that will minimally elapse before the current epoch ends. The value of the Max-Age option MUST NOT result in indicating that the current epoch ends before t_C.

A Device receiving this error response gains knowledge of when the immediately following "Recovery Transfer" phase starts and when the next epoch starts.

The former information is useful for a Device that participated in the immediately previous "Full Transfer" phase and missed some outer chunks. That Device can thus set itself up for receiving such outer chunks, which will be distributed during the immediately following "Recovery Transfer" phase.

In particular, the Device becomes aware that the "Recovery Transfer" phase is scheduled to start not before the amount of seconds indicated by the "next_not_before" parameter and that the outer chunks to be transferred pertain to the inner chunk with index INNER_INDEX indicated by the "progress_indicator" parameter. Having participated in the immediately previous "Full Transfer" phase, the Device is aware that the responses representing the outer chunks will be sent per the information specified in the "tp_info" parameter of the "Hold-on Response" that the Device received during the "Admission" phase of the current epoch.

This phase finishes when the time t_B is reached and the related timer expires. After that, the epoch moves to the "Recovery Transfer" phase (see Section 4.3.2.5).

4.3.2.5. "Recovery Transfer" Phase

When this phase starts, the Proxy prepares a set of outer chunks, namely RECOVERY_CHUNKS. The set includes a selection of the outer chunks that were sent during the "Full Transfer" phase of the current epoch (see Section 4.3.2.3). In particular, each of such outer chunks is added to RECOVERY_CHUNKS if and only if the value specified by the NUM field in the value of its Block2 outer option is an element of the RECOVERY_INDEXES set.

After that, the Proxy distributes only the outer chunks included in RECOVERY_CHUNKS, just like it distributed the outer chunks during the "Full Transfer" phase of the current epoch (see Section 4.3.2.3). In particular, the outer chunks are sent according to their indexes sorted in ascending order. Their transmission occurs over UDP and IP multicast, according to the same information specified in the "tp_info" parameter of the "Hold-on Responses" that were sent during the "Admission" phase of the current epoch.

The Proxy MUST NOT alter the time t_C that was determined during the "Full Transfer" phase, as the time when the current epoch is expected to end.

After the Proxy has sent all the outer chunks that are an element of RECOVERY_CHUNKS, the Proxy deletes the RECOVERY_INDEXES and RECOVERY_CHUNKS sets, and the epoch moves to the final "Epilogue" (see Section 4.3.2.6).

If the Proxy receives a Deterministic Request targeting the Distributor and that produces a cache hit for the cache entry ENTRY associated with an epoch with current phase "Recovery Transfer" or in its "Epilogue", the Proxy MUST reply with a 5.03 (Service Unavailable) error response that has no payload. The error response MUST include the Max-Age option, with value the amount of seconds that will minimally elapse before the current epoch ends. The value of the Max-Age option MUST NOT result in indicating that the current epoch ends before t_C.

4.3.2.6. Epilogue

To conclude the current epoch, the Proxy performs the following steps.

1. The Proxy considers the response stored in the cache entry ENTRY associated with the current epoch, and particularly the field M in the value of the Block2 option.

2. The Proxy updates the value of INNER_INDEX associated with ENTRY as follows:

   * If M is 1, then INNER_INDEX is incremented by 1.

   * If M is 0, then INNER_INDEX takes 0 as its new value.

3. When the time t_C is reached, the Proxy concludes the current epoch, starts a new epoch with "Admission" as its current phase, deletes the association between the old epoch end ENTRY, and associates the new epoch with ENTRY.

5. Checksum on Outer Chunks

This section defines how the Proxy computes a checksum value over each outer chunk that it sends to the target Devices during the "Full Transfer" phase (see Section 4.3.2.3) and the "Recovery Transfer" phase (see Section 4.3.2.5).

As described in Section 5.4, the computed checksum value is specified as the value of the CoAP Checksum option defined in Section 5.3, which the Proxy includes in the response sent as outer chunk to the Devices.

Upon receiving an outer chunk, a Device recomputes the checksum and compares it against the value conveyed in the Checksum option, in order to check whether the outer chunk was altered in transit.

The checksum value of an outer chunk is computed by using a checksum key, whose derivation is defined in Section 5.2. The same checksum key is used to compute the checksum values for all the outer chunks of the same inner chunk, i.e., during a whole epoch (see Section 4.3).

While a checksum key is _used_ by the Proxy and the Devices, the checksum key is _derived_ by the Distributor and the Devices, by using keying material in their shared Group OSCORE Security Context.

The Distributor provides the Proxy with the checksum key to use during the current epoch (see Section 5.6), when sending to the Proxy a CoAP response over TCP as the inner chunk to distribute in that epoch (see Section 4.3.2.1 and Section 4.3.2.2). In particular, the Distributor wraps the checksum key in a CBOR data item and prepends that data item to the OSCORE ciphertext in the CoAP payload of the response sent to the Proxy. The Distributor indicates the presence of the prepended CBOR data item by including in the response the CoAP Pre-OSCORE-Data option defined in Section 5.5.

The use of checksums counteracts an attack against availability that an active adversary could easily perform by manipulating the CoAP responses sent by the Proxy to the Devices as outer chunks. By recomputing a checksum and verifying it against the one included in the response received from the Proxy, target Devices can promptly detect a possible manipulation of the outer chunk and discard the response as invalid.

5.1. Root Checksum Key

When using the method defined in this document, the Group OSCORE Security Context shared by the Distributor and the Devices is extended with one additional parameter in the Common Context.

The new parameter Root Checksum Key specifies a secret symmetric key. This is used for deriving checksum keys that are in turn used for computing checksums of outer chunks (see Section 5.4).

The Root Checksum Key is derived as defined for the 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 'type' element of the 'info' array is "RCKey". The label is an ASCII string and does not include a trailing NUL byte.

* The 'alg_aead' element of the 'info' array specifies the Group Encryption Algorithm from the Common Context (see Section 2.1.7 of [I-D.ietf-core-oscore-groupcomm]) encoded as a CBOR integer or text string, consistently with the "Value" field in the entry of the "COSE Algorithms" Registry for this algorithm [COSE.Algorithms].

* The L parameter of the HKDF and the 'L' element of the 'info' array are the length in bytes of the key for the Group Encryption Algorithm specified in the Common Context. While the obtained Root Checksum Key is never used with the Group Encryption Algorithm, its length was chosen to obtain a matching level of security.

5.2. Derivation of Checksum Keys

The same checksum key K is used for computing the checksum value on all the outer chunks of the same inner chunk.

In particular, K is derived:

* By the Distributor, upon sending to the Proxy a CoAP response over TCP as the inner chunk to distribute in the current epoch (see Section 4.3.2.1 and Section 4.3.2.2).

  Note that the Distributor provides the Proxy with the checksum key K as embedded in that response (see Section 5.6).

* By each target Device, when receiving an outer chunk of the inner chunk distributed in the current epoch (see Section 4.3.2.3 and Section 4.3.2.5).

A checksum key K SHALL be derived as follows, by using the HKDF Algorithm from the Common Context of the Group OSCORE Security Context (see Section 2.1.2 of [I-D.ietf-core-oscore-groupcomm]), which consists of composing the HKDF-Extract and HKDF-Expand steps [RFC5869].

K = HKDF(salt, IKM, info, L)

The input parameters of HKDF are as follows.

* salt takes as value the index of the inner chunk to distribute in the current epoch, i.e., INNER_INDEX, represented in the smallest number of bytes needed.

  From the Distributor point of view, this value is specified by the NUM field of the value of the Block2 outer option, which is included in the response sent to the Proxy as the inner chunk.

  From the Device point of view, this value is encoded as a CBOR unsigned integer by the "progress_indicator" parameter, which is conveyed in the payload of the error informative responses that the Proxy sends during the "Admission" phase (see Section 4.3.2.2) and the "Recovery Claim" phase (see Section 4.3.2.4).

* IKM is the Root Checksum Key from the Common Context (see Section 5.1).

* info is the serialization of a CBOR array with the structure defined below, following the notation of [RFC8610]):

  ```
  info = [
    piv : bstr,
    L : uint
  ]
  ```

where:

* piv is the Partial IV field in the OSCORE option of the following messages:

  - From the Distributor point of view, the response that the Distributor sends to the Proxy as the inner chunk to distribute in the current epoch.

Note that such a message is specifically a response to a Deterministic Request. Therefore, it always includes a Partial IV in the OSCORE option (see [I-D.ietf-core-cacheable-oscore]).

- From the Device point of view, the responses received from the Proxy during the "Full Transfer" phase (see Section 4.3.2.3) and "Recovery Transfer" phase (see Section 4.3.2.5) as the outer chunks of the inner chunk distributed in the current epoch.

  Note that all such responses include a Partial IV in the OSCORE option. The value of the Partial IV is the same one of the Partial IV in the OSCORE option of the response that the Proxy received from the Distributor as the inner chunk to distribute in the current epoch.

* The L parameter of the HKDF and the 'L' element of the 'info' array are the length in bytes of the Root Checksum Key from the Common Context.

5.3. Checksum Option

The Checksum option defined in this section has the properties summarized in Table 1, which extends Table 4 of [RFC7252]. The option is Elective, Safe-to-Forward, and part of the Cache-Key. The option MUST NOT occur more than once.

No.	C	U	N	R	Name	Format	Length	Default
TBD256					Checksum	opaque	1-8	(none)

Table 1: Checksum Option. C=Critical, U=Unsafe, N=NoCacheKey, R=Repeatable

The option value is a checksum computed over (part of) the message by the endpoint that added the option, thereby enabling the message recipient to perform an integrity check on the message.

The Checksum option is of class U for OSCORE [RFC8613][I-D.ietf-core-oscore-groupcomm].

5.4. Computation and Embodiment of Checksums

The Proxy computes the checksum on an outer chunk before sending that outer chunk to the target Devices, during the "Full Transfer" phase (see Section 4.3.2.3) and "Recovery Transfer" phase (see Section 4.3.2.5).

After having computed the checksum as defined below, the Proxy specifies it as value of the Checksum option (see Section 5.3) and includes the option in the response sent as outer chunk to the Devices. If outer CoAP options were already included in the response and their option number is greater than that of the Checksum option, then the Proxy appropriately updates their Option Delta (see Section 3.1 of [RFC7252]).

A Device computes the checksum on an outer chunk upon receiving that outer chunk from the Proxy, during the "Full Transfer" phase (see Section 4.3.2.3) and "Recovery Transfer" phase (see Section 4.3.2.5).

In particular, the Device first removes the Checksum option from the response received as outer chunk from the Proxy. After that, if outer CoAP options are included in the response and their option number is greater than that of the Checksum option, then the Device appropriately updates their Option Delta. Finally, the Device computes the checksum on the resulting response and compares the result against the checksum specified as value of the removed Checksum option.

If the two checksums are not equal, the Device MUST discard the response without further processing it. If this happens during the "Full Transfer" phase, the Device can send a request to the Proxy during the immediately following "Recovery Claim" phase (see Section 4.3.2.4), thus asking the Proxy to re-send the outer chunk during the immediately following "Recovery Transfer" phase.

Given a response that the Proxy sends as outer chunk, the checksum on that outer chunk is a 2-byte MAC that SHALL be computed as follows by using the HKDF algorithm HKDF SHA-256, which consists of composing the HKDF-Extract and HKDF-Expand steps [RFC5869].

MAC = HKDF(salt, IKM, info, L)

The input parameters of HKDF are as follows.

* salt takes as value the index of the inner chunk to distribute in the current epoch, i.e., INNER_INDEX, represented in the smallest number of bytes needed.

From the Proxy point of view, INNER_INDEX is stored and consistently updated throughout the different epochs of the image distribution (see Section 4.3.2).

From the Device point of view, this value is encoded as a CBOR unsigned integer by the "progress_indicator" parameter, which is conveyed in the payload of the error informative responses that the Proxy sends during the "Admission" phase (see Section 4.3.2.2) and the "Recovery Claim" phase (see Section 4.3.2.4).

* IKM is the Checksum Key to use for this inner chunk, which is derived as defined in Section 5.2.

* info is the serialization of the CoAP response that the Proxy has to send as outer chunk.

  The Proxy MUST use the CoAP response available before the addition of the Checksum option.

  The Device MUST use the CoAP response available after:

  - Removing the Checksum option; and

  - Updating the Option Delta of each outer option whose option number is greater than that of the Checksum option.

* L has value 2.

5.5. Pre-OSCORE-Data Option

The Pre-OSCORE-Data option defined in this section has the properties summarized in Table 2, which extends Table 4 of [RFC7252]. The option is Critical, Safe-to-Forward, part of the Cache-Key, and repeatable.

No.	C	U	N	R	Name	Format	Length	Default
TBD257	x			x	Pre-OSCORE-Data	uint	0-4	(none)

Table 2: Pre-OSCORE-Data Option. C=Critical, U=Unsafe, N=NoCacheKey, R=Repeatable

The presence of this option means that, within the CoAP payload, the OSCORE ciphertext is prepended by a CBOR data item that is intended for the consumer of the option.

The option value is an unsigned integer that identifies the semantics of the data conveyed by the CBOR data item. In particular, the option value MUST be either X or (X + 1), where X is an odd value taken from the 'Value' column of the "Pre-OSCORE Data Semantics" IANA registry defined in Section 8.3.

Both values X and (X + 1) identify the same data semantics. However:

* The odd value X means that the CBOR data item is a CBOR byte string, whose value is the data with the indicated semantics.

* The even value (X + 1) means that the CBOR data item is a COSE object, i.e., a possibly tagged COSE message as defined in Section 2 of [RFC9052].

  The data with the indicated semantics consists of what is available at the recipient once successfully completed all the COSE processing, e.g., the data is a plaintext recovered after a decryption process.

Note that, although even values cannot be registered in the "Pre-OSCORE Data Semantics" IANA registry (see Section 8.3), those values are meaningful semantics identifiers that can be used as option value. That is, the even value Y identifies the same semantics identified by its companion odd value X = (Y - 1).

The recipient of a CoAP message including the Pre-OSCORE-Data option MUST consume the option, i.e., it removes and consumes the prepended CBOR data item from the CoAP payload, after which it removes the option from the message.

The Pre-OSCORE-Data option MAY occur multiple times. In such a case, each occurrence of the option refers to one CBOR data item prepended to the OSCORE ciphertext within the CoAP payload. In particular, the i-th occurrence of the option refers to the i-th prepended CBOR data item.

The Pre-OSCORE-Data option is of class U for OSCORE [RFC8613][I-D.ietf-core-oscore-groupcomm].

5.6. Provisioning of Checksum Keys to the Proxy

When the Distributor sends to the Proxy a CoAP response over TCP as the inner chunk to distribute during an epoch (see Section 4.3), the Distributor also provides the Proxy with a Checksum Key, i.e., the one to use during that epoch for computing the checksums on the outer chunks of that inner chunk.

The Distributor derives the Checksum Key as defined in Section 5.2 and includes it in the response sent to the Proxy, as a CBOR data item prepended to the OSCORE ciphertext that is conveyed by the CoAP payload of the response. In order to indicate the presence of such CBOR data item, the response MUST include the Pre-OSCORE Data option Section 5.5, whose value is set as defined below.

There are two possible ways for the Distributor to provide the Checksum Key in the response to the Proxy as a prepended CBOR data item:

* The CBOR data item is a CBOR byte string, whose value is the Checksum Key. In such a case, the value of the Pre-OSCORE Data option MUST be set to 1.

  This alternative SHOULD be used by the Distributor, if the response to the Proxy is protected by means of a mutually authenticated, secure communication association between the Distributor and the Proxy, in such a way that only the Proxy is able to retrieve the protected content in plain.

* The CBOR data item is a COSE object [RFC9052], which can be tagged or untagged. In such a case, the value of the Pre-OSCORE Data option MUST be set to 2.

  The COSE Object is the result of using HPKE [RFC9180] with COSE. In particular, the HPKE Integrated Encryption Mode specified in Section 3.1.1 of [I-D.ietf-cose-hpke] MUST be used. The input pt for the HPKE Seal operation is the Checksum Key. The resulting COSE object uses a COSE_Encrypt0 structure [RFC9052].

  In order to ensure source authentication of the prepended CBOR data item, the Distributor can additionally rely on a COSE object that uses a COSE_Sign, COSE_Sign1, COSE_MAC, or COSE_MAC0 structure [RFC9052]. In such a case, the COSE_Encrypt0 object is used as payload of the COSE_Sign, COSE_Sign1, COSE_MAC, or COSE_MAC0 structure.

  This alternative MUST be used by the Distributor, unless the response to the Proxy is protected by means of a mutually authenticated, secure communication association between the Distributor and the Proxy, in such a way that only the Proxy is able to retrieve the protected content in plain.

  This alternative requires the Distributor to know:

  - The COSE-HPKE algorithm to use with the Proxy (see Section 4 of [I-D.ietf-cose-hpke]).

- The static public key of the Proxy to use as the input pkR of the HPKE Seal operation (see Section 3.1.1 of [I-D.ietf-cose-hpke]).

Editor's note: describe examples of how the Distributor can obtain the static public key of the Proxy and possibly select the right one to use at runtime.

Secure communication associations between the Distributor and the Proxy can rely, for example, on a TLS [RFC8446] channel where the Distributor has been authenticated during the secure channel establishment, or on a pairwise OSCORE [RFC8613] Security Context shared between the Distributor and the Proxy, as defined in [I-D.ietf-core-oscore-capable-proxies].

6. Pre-OSCORE Data Semantics

This document defines the following semantics for data prepended to the ciphertext conveyed in the CoAP payload of a message protected with OSCORE [RFC8613] or Group OSCORE [I-D.ietf-core-oscore-groupcomm].

Value	Description	Reference
1	Checksum key	[RFC-XXXX], Section 5.6

Table 3: Pre-OSCORE Data Semantics.

7. Security Considerations

Security considerations are inherited from [RFC7252], [I-D.ietf-core-groupcomm-bis], and [RFC8323] as to the use of CoAP also for group communication and over reliable transports.

Security considerations are also inherited from [RFC7641] as to the use of CoAP Observe, from [RFC7959] as to the use of Block-wise transfer for CoAP, and from [RFC8323] as to the use of BERT.

Security considerations are also inherited from [I-D.ietf-core-oscore-groupcomm] for the use of Group OSCORE and from [I-D.ietf-core-cacheable-oscore] as to the specific use of protected Deterministic Requests and the caching of corresponding protected responses.

Furthermore, the following security considerations also apply.

Editor’s note: add more security considerations.

8. IANA Considerations

This document has the following actions for IANA.

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

8.1. CoAP Option Numbers Registry

IANA is asked to enter the following option numbers to the "CoAP Option Numbers" registry within the "Constrained RESTful Environments (CoRE) Parameters" registry group.

Number	Name	Reference
TBD256	Checksum	[RFC-XXXX]
TBD257	Pre-OSCORE-Data	[RFC-XXXX]

Table 4: Registrations in the CoAP Option Numbers Registry

8.2. Informative Response Parameters Registry

IANA is asked to enter the following entry to the "Informative Response Parameters" registry defined in [I-D.ietf-core-observe-multicast-notifications] within the "Constrained RESTful Environments (CoRE) Parameters" registry group.

* Name: progress_indicator
* CBOR Key: TBD23
* CBOR Type: uint
* Reference: [RFC-XXXX]

8.3. Pre-OSCORE Data Semantics Registry

This document establishes the "Pre-OSCORE Data Semantics" registry within the "Constrained RESTful Environments (CoRE) Parameters" registry group.

The registration policy is either "Private Use", "RFC Required With Expert Review", or "Specification Required" per [RFC8126]. "Expert Review" guidelines are provided in Section 8.4.

All assignments according to "RFC Required With Expert Review" are made on an "RFC Required" basis per Section 4.7 of [RFC8126] with "Expert Review" additionally required per Section 4.5 of [RFC8126]. The procedure for early IANA allocation of "standards track code points" defined in [RFC7120] also applies. When such a procedure is used, IANA will ask the designated expert(s) to approve the early allocation before registration. In addition, working group chairs are encouraged to consult the expert(s) early during the process outlined in Section 3.1 of [RFC7120].

The columns of this registry are:

* Value: This field contains the value used to identify the semantics of the data that is prepended to the ciphertext conveyed in the CoAP payload of a message protected with OSCORE [RFC8613] or Group OSCORE [I-D.ietf-core-oscore-groupcomm]. These values MUST be unique. The value MUST be an odd unsigned integer, with minimum value 1 and maximum value 4294967293 (i.e., 2^32 - 3). Odd unsigned integer values from 1 to 255 are designated as "RFC Required With Expert Review". Odd unsigned integer values from 257 to 4294965293 are designated as "Specification Required". Odd unsigned integer values from 4294965295 to 4294967293 are marked as "Private Use".

* Description: This field contains a brief description of the semantics.

* Reference: This field contains a pointer to the public specification defining the semantics, if one exists.

This registry has been initially populated by the values in Section 6.

8.4. Expert Review Instructions

"RFC Required With Expert Review" and "Specification Required" are two of the registration policies defined for the IANA registry established in this document. This section gives some general guidelines for what the experts should be looking for, but they are being designated as experts for a reason, so they should be given substantial latitude.

* Point squatting should be discouraged. Reviewers are encouraged to get sufficient information for registration requests to ensure that the usage is not going to duplicate one that is already registered and that the point is likely to be used in deployments. The zones tagged as "Private Use" are intended for testing purposes and closed environments. Code points in other ranges should not be assigned for testing.

* Specifications are required for the "RFC Required With Expert Review" range of point assignment. Specifications should exist for "Specification Required" ranges, but early assignment before a specification is available is considered to be permissible. When specifications are not provided, the description provided needs to have sufficient information to identify what the point is being used for.

* Experts should take into account the expected usage of fields when approving point assignment. The fact that there is a range for RFC documents does not mean that an RFC document cannot have points assigned outside of that range. The length of the encoded value should be weighed against how many code points of that length are left, the size of device it will be used on, and the number of code points left that encode to that size.

9. References

9.1. Normative References

[COSE.Algorithms]
IANA, "COSE Algorithms", <https://www.iana.org/assignments/cose/cose.xhtml#algorithms>.

[I-D.ietf-core-cacheable-oscore]
Amsüss, C. and M. Tiloca, "Cacheable OSCORE", Work in Progress, Internet-Draft, draft-ietf-core-cacheable-oscore-00, 22 September 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-core-cacheable-oscore-00>.

[I-D.ietf-core-groupcomm-bis]
Dijk, E. and M. Tiloca, "Group Communication for the Constrained Application Protocol (CoAP)", Work in Progress, Internet-Draft, draft-ietf-core-groupcomm-bis-15, 25 September 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-core-groupcomm-bis-15>.

[I-D.ietf-core-href]
Bormann, C. and H. Birkholz, "Constrained Resource Identifiers", Work in Progress, Internet-Draft, draft-ietf-core-href-30, 21 November 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-core-href-30>.

[I-D.ietf-core-observe-multicast-notifications]
Tiloca, M., H旦glund, R., Ams端ss, C., and F. Palombini, "Observe Notifications as CoAP Multicast Responses", Work in Progress, Internet-Draft, draft-ietf-core-observe-multicast-notifications-13, 20 October 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-core-observe-multicast-notifications-13>.

[I-D.ietf-core-oscore-groupcomm]
Tiloca, M., Selander, G., Palombini, F., Mattsson, J. P., and R. H旦glund, "Group Object Security for Constrained RESTful Environments (Group OSCORE)", Work in Progress, Internet-Draft, draft-ietf-core-oscore-groupcomm-28, 23 December 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-core-oscore-groupcomm-28>.

[I-D.ietf-cose-hpke]
Tschofenig, H., Steele, O., Daisuke, A., Lundblade, L., and M. B. Jones, "Use of Hybrid Public-Key Encryption (HPKE) with CBOR Object Signing and Encryption (COSE)", Work in Progress, Internet-Draft, draft-ietf-cose-hpke-19, 15 December 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-cose-hpke-19>.

[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>.

[RFC5869] Krawczyk, H. and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)", RFC 5869, DOI 10.17487/RFC5869, May 2010, <https://www.rfc-editor.org/rfc/rfc5869>.

[RFC7120] Cotton, M., "Early IANA Allocation of Standards Track Code Points", BCP 100, RFC 7120, DOI 10.17487/RFC7120, January 2014, <https://www.rfc-editor.org/rfc/rfc7120>.

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

[RFC7641] Hartke, K., "Observing Resources in the Constrained Application Protocol (CoAP)", RFC 7641, DOI 10.17487/RFC7641, September 2015, <https://www.rfc-editor.org/rfc/rfc7641>.

[RFC7959] Bormann, C. and Z. Shelby, Ed., "Block-Wise Transfers in the Constrained Application Protocol (CoAP)", RFC 7959, DOI 10.17487/RFC7959, August 2016, <https://www.rfc-editor.org/rfc/rfc7959>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/rfc/rfc8126>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/rfc/rfc8174>.

[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>.

[RFC8610] Birkholz, H., Vigano, C., and C. Bormann, "Concise Data Definition Language (CDDL): A Notational Convention to Express Concise Binary Object Representation (CBOR) and JSON Data Structures", RFC 8610, DOI 10.17487/RFC8610, June 2019, <https://www.rfc-editor.org/rfc/rfc8610>.

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

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

[RFC9052] Schaad, J., "CBOR Object Signing and Encryption (COSE): Structures and Process", STD 96, RFC 9052, DOI 10.17487/RFC9052, August 2022, <https://www.rfc-editor.org/rfc/rfc9052>.

[RFC9053] Schaad, J., "CBOR Object Signing and Encryption (COSE): Initial Algorithms", RFC 9053, DOI 10.17487/RFC9053, August 2022, <https://www.rfc-editor.org/rfc/rfc9053>.

[RFC9180] Barnes, R., Bhargavan, K., Lipp, B., and C. Wood, "Hybrid Public Key Encryption", RFC 9180, DOI 10.17487/RFC9180, February 2022, <https://www.rfc-editor.org/rfc/rfc9180>.

9.2. Informative References

[I-D.ietf-ace-key-groupcomm-oscore]
Tiloca, M. and F. Palombini, "Key Management for Group Object Security for Constrained RESTful Environments (Group OSCORE) Using Authentication and Authorization for Constrained Environments (ACE)", Work in Progress, Internet-Draft, draft-ietf-ace-key-groupcomm-oscore-18, 28 August 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-ace-key-groupcomm-oscore-18>.

[I-D.ietf-core-multicast-notifications-proxy]
Tiloca, M., H旦glund, R., Ams端ss, C., and F. Palombini, "Using Proxies for Observe Notifications as CoAP Multicast Responses", Work in Progress, Internet-Draft, draft-ietf-core-multicast-notifications-proxy-00, 20 October 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-core-multicast-notifications-proxy-00>.

[I-D.ietf-core-oscore-capable-proxies]
Tiloca, M. and R. H旦glund, "OSCORE-capable Proxies", Work in Progress, Internet-Draft, draft-ietf-core-oscore-capable-proxies-05, 3 September 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-core-oscore-capable-proxies-05>.

[I-D.ietf-suit-manifest]
Moran, B., Tschofenig, H., Birkholz, H., Zandberg, K., and O. R淡nningstad, "A Concise Binary Object Representation (CBOR)-based Serialization Format for the Software Updates for Internet of Things (SUIT) Manifest", Work in Progress, Internet-Draft, draft-ietf-suit-manifest-34, 28 May 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-suit-manifest-34>.

[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>.

[RFC9019] Moran, B., Tschofenig, H., Brown, D., and M. Meriac, "A Firmware Update Architecture for Internet of Things", RFC 9019, DOI 10.17487/RFC9019, April 2021, <https://www.rfc-editor.org/rfc/rfc9019>.

[RFC9124] Moran, B., Tschofenig, H., and H. Birkholz, "A Manifest Information Model for Firmware Updates in Internet of Things (IoT) Devices", RFC 9124, DOI 10.17487/RFC9124, January 2022, <https://www.rfc-editor.org/rfc/rfc9124>.

[RFC9200] Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "Authentication and Authorization for Constrained Environments Using the OAuth 2.0 Framework (ACE-OAuth)", RFC 9200, DOI 10.17487/RFC9200, August 2022, <https://www.rfc-editor.org/rfc/rfc9200>.

[RFC9528] Selander, G., Preu Mattsson, J., and F. Palombini, "Ephemeral Diffie-Hellman Over COSE (EDHOC)", RFC 9528, DOI 10.17487/RFC9528, March 2024, <https://www.rfc-editor.org/rfc/rfc9528>.

Acknowledgments

The author sincerely thanks Christian Amsss, Peter Blomqvist, Carsten Bormann, Rikard Hglund, Gran Selander, and Malia Vuini for their comments and feedback.

The work on this document has been partly supported by the Sweden’s Innovation Agency VINNOVA and the Celtic-Next projects CRITISEC and CYPRESS.

Author’s Address

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