Workgroup: CoRE Working Group
Internet-Draft:
draft-tiloca-core-oscore-capable-proxies-04
Updates: 8613 (if approved)
Published: 23 September 2022
Intended Status: Standards Track
Expires: 27 March 2023
Authors: M. Tiloca R. Höglund
RISE AB RISE AB

OSCORE-capable Proxies

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 27 March 2023.

Copyright Notice

Table of Contents

1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] supports the presence of intermediaries, such as forward-proxies and reverse-proxies, which assist origin clients by performing requests to origin servers on their behalf, and forwarding back the related responses.

CoAP supports also group communication scenarios [I-D.ietf-core-groupcomm-bis], where clients can send a one-to-many request targeting all the servers in the group, e.g., by using IP multicast. Like for one-to-one communication, group settings can also rely on intermediaries [I-D.tiloca-core-groupcomm-proxy].

The protocol Object Security for Constrained RESTful Environments (OSCORE) [RFC8613] can be used to protect CoAP messages between two endpoints at the application layer, especially achieving end-to-end security in the presence of (non-trusted) intermediaries. When CoAP group communication is used, the same can be achieved by means of the protocol Group OSCORE [I-D.ietf-core-oscore-groupcomm].

For a number of use cases (see Section 2), it is required and/or beneficial that communications are secured also between an application endpoint (i.e., a CoAP origin client/server) and an intermediary, as well as between two adjacent intermediaries in a chain. This especially applies to the communication leg between the CoAP origin client and the adjacent intermediary acting as next hop towards the CoAP origin server.

In such cases, and especially if the origin client already uses OSCORE to achieve end-to-end security with the origin server, it would be convenient that OSCORE is used also to secure communications between the origin client and its next hop. However, the original specification [RFC8613] does not define how OSCORE can be used to protect CoAP messages in such communication leg, which would require to consider also the intermediary as an "OSCORE endpoint".

This document fills this gap, and updates [RFC8613] as follows.

* It defines how to use OSCORE for protecting a CoAP message in the communication leg between: i) an origin client/server and an intermediary; or ii) two adjacent intermediaries in an intermediary chain. That is, besides origin clients/servers, it allows also intermediaries to be possible "OSCORE endpoints".

* It admits a CoAP message to be secured by multiple, nested OSCORE protections applied in sequence, as an "OSCORE-in-OSCORE" process. For instance, this is the case when the message is OSCORE-protected end-to-end between the origin client and origin server, and the result is further OSCORE-protected over the leg between the current and next hop (e.g., the origin client and the adjacent intermediary acting as next hop towards the origin server).

This document does not specify any new signaling method to guide the message processing on the different endpoints. In particular, every endpoint is always able to understand what steps to take on an incoming message depending on the presence of the OSCORE Option, as exclusively included or instead combined together with CoAP options intended for an intermediary.

The approach defined in this document can be seamlessly adopted also when Group OSCORE is used, for protecting CoAP messages in group communication scenarios that rely on intermediaries.

1.1. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

Readers are expected to be familiar with the terms and concepts related to CoAP [RFC7252]; OSCORE [RFC8613] and Group OSCORE [I-D.ietf-core-oscore-groupcomm]. This document especially builds on concepts and mechanics related to intermediaries such as CoAP forward-proxies.

In addition, this document uses the following terms.

* Source application endpoint: an origin client producing a request, or an origin server producing a response.

* Destination application endpoint: an origin server intended to consume a request, or an origin client intended to consume a response.

* Application endpoint: a source or destination application endpoint.

* Source OSCORE endpoint: an endpoint protecting a message with OSCORE or Group OSCORE.

* Destination OSCORE endpoint: an endpoint unprotecting a message with OSCORE or Group OSCORE.

* OSCORE endpoint: a source/destination OSCORE endpoint. An OSCORE endpoint is not necessarily also an application endpoint with respect to a certain message.

* Proxy-related options: either of the following (set of) CoAP options used for proxying a CoAP request.

  - The Proxy-Uri Option. This is relevant when using a forward-proxy.

  - The set of CoAP options comprising the Proxy-Scheme Option together with any of the Uri-* Options. This is relevant when using a forward-proxy.

  - One or more Uri-Path Options, when used not together with the Proxy-Scheme Option. This is relevant when using a reverse-proxy.

* OSCORE-in-OSCORE: the process by which a message protected with (Group) OSCORE is further protected with (Group) OSCORE. This means that, if such a process is used, a successful decryption/verification of an OSCORE-protected message might yield an OSCORE-protected message.

2. Use Cases

The approach defined in this document has been motivated by a number of use cases, which are summarized below.

2.1. CoAP Group Communication with Proxies

CoAP supports also one-to-many group communication, e.g., over IP multicast [I-D.ietf-core-groupcomm-bis], which can be protected end-to-end between origin client and origin servers by using Group OSCORE [I-D.ietf-core-oscore-groupcomm].

This communication model can be assisted by intermediaries such as a CoAP forward-proxy or reverse-proxy, which relays a group request to the origin servers. If Group OSCORE is used, the proxy is intentionally not a member of the OSCORE group. Furthermore, [I-D.tiloca-core-groupcomm-proxy] defines a signaling protocol between origin client and proxy, to ensure that responses from the different origin servers are forwarded back to the origin client within a time interval set by the client, and that they can be distinguished from one another.

In particular, it is required that the proxy identifies the origin client as allowed-listed, before forwarding a group request to the servers (see Section 4 of [I-D.tiloca-core-groupcomm-proxy]). This requires a security association between the origin client and the proxy, which would be convenient to provide with a dedicated OSCORE Security Context between the two, since the client is possibly using also Group OSCORE with the origin servers.

2.2. CoAP Observe Notifications over Multicast

The Observe extension for CoAP [RFC7641] allows a client to register its interest in "observing" a resource at a server. The server can then send back notification responses upon changes to the resource representation, all matching with the original observation request.

In some applications, such as pub-sub [I-D.ietf-core-coap-pubsub], multiple clients are interested to observe the same resource at the same server. Hence, [I-D.ietf-core-observe-multicast-notifications] defines a method that allows the server to send a multicast notification to all the observer clients at once, e.g., over IP multicast. To this end, the server synchronizes the clients by providing them with a common "phantom observation request", against which the following multicast notifications will match.

In case the clients and the server use Group OSCORE for end-to-end security and a proxy is also involved, an additional step is required (see Section 12 of [I-D.ietf-core-observe-multicast-notifications]). That is, clients are in turn required to provide the proxy with the obtained "phantom observation request", thus enabling the proxy to receive the multicast notifications from the server.

Therefore, it is preferable to have a security association also between each client and the proxy, to especially ensure the integrity of that information provided to the proxy (see Section 15.3 of [I-D.ietf-core-observe-multicast-notifications]). Like for the use case in Section 2.1, this would be conveniently achieved with a dedicated OSCORE Security Context between a client and the proxy, since the client is also using Group OSCORE with the origin server.

2.3. LwM2M Client and External Application Server

The Lightweight Machine-to-Machine (LwM2M) protocol [LwM2M-Core] enables a LwM2M Client device to securely bootstrap and then register at a LwM2M Server, with which it will perform most of its following communication exchanges. As per the transport bindings specification of LwM2M [LwM2M-Transport], the LwM2M Client and LwM2M

Server can use CoAP and OSCORE to secure their communications at the application layer, including during the device registration process.

Furthermore, Section 5.5.1 of [LwM2M-Transport] specifies that: "OSCORE MAY also be used between LwM2M endpoint and non-LwM2M endpoint, e.g., between an Application Server and a LwM2M Client via a LwM2M server. Both the LwM2M endpoint and non-LwM2M endpoint MUST implement OSCORE and be provisioned with an OSCORE Security Context."

In such a case, the LwM2M Server can practically act as forward-proxy between the LwM2M Client and the external Application Server. At the same time, the LwM2M Client and LwM2M Server must continue protecting communications on their leg using their Security Context. Like for the use case in Section 2.1, this also allows the LwM2M Server to identify the LwM2M Client, before forwarding its request outside the LwM2M domain and towards the external Application Server.

2.4. LwM2M Gateway

The specification [LwM2M-Gateway] extends the LwM2M architecture by defining the LwM2M Gateway functionality. That is, a LwM2M Server can manage end IoT devices "behind" the LwM2M Gateway. While it is outside the scope of such specification, it is possible for the LwM2M Gateway to use any suitable protocol with its connected end IoT devices, as well as to carry out any required protocol translation.

Practically, the LwM2M Server can send a request to the LwM2M Gateway, asking to forward it to an end IoT device. With particular reference to the CoAP protocol and the related transport binding specified in [LwM2M-Transport], the LwM2M Server acting as CoAP client sends its request to the LwM2M Gateway acting as CoAP server.

If CoAP is used in the communication leg between the LwM2M Gateway and the end IoT devices, then the LwM2M Gateway fundamentally acts as a reverse-proxy (see Section 5.7.3 of [RFC7252]). That is, in addition to its own resources, the LwM2M Gateway serves the resources of each end IoT device behind itself, as exposed under a dedicated URI-Path. As per [LwM2M-Gateway], the first URI-Path segment is used as "prefix" to identify the specific IoT device, while the remaining URI-Path segments specify the target resource at the IoT device.

As per Section 7 of [LwM2M-Gateway], message exchanges between the LwM2M Server and the L2M2M Gateway are secured using the LwM2M-defined technologies, while the LwM2M protocol does not provide end-to-end security between the LwM2M Server and the end IoT devices.

However, the approach defined in this document makes it possible to achieve both goals, by allowing the LwM2M Server to use OSCORE for protecting a message both end-to-end for the targeted end IoT device as well as for the LwM2M Gateway acting as reverse-proxy.

2.5. Further Use Cases

The approach defined in this document can be useful also in the following use cases relying on a proxy.

* A server aware of a suitable cross proxy can rely on it as a third-party service, in order to indicate transports for CoAP available to that server (see Section 4 of [I-D.ietf-core-transport-indication]).

  From a security point of view, it would be convenient if the proxy could provide suitable credentials to the client, as a general trusted proxy for the system. At the same time, it can be desirable to limit the use of such a proxy to a set of clients which have permission to use it, and that the proxy can identify through a secure communication association.

  However, in order for OSCORE to be an applicable security mechanism for this, it has to be terminated at the proxy. That is, it would be required for a client and the proxy to share a dedicated OSCORE Security Context and to use it for protecting their communication leg.

* A proxy may be deployed to act as an entry point to a firewalled network, which only authenticated clients can join. In particular, authentication can rely on the used secure communication association between a client and the proxy. If the proxy could share a dedicated OSCORE Security Context with each client, the proxy can rely on it to identify the client, before forwarding its messages to any other member of the firewalled network.

* The approach defined in this document does not pose a limit to the number of OSCORE protections applied to the same CoAP message. This enables more privacy-oriented scenarios based on proxy chains, where the origin client protects a CoAP request using first the OSCORE Security Context shared with the origin server, and then the dedicated OSCORE Security Context shared with each of the different chain hops. Once received at a chain hop, the request would be stripped of the OSCORE protection associated with that hop before being forwarded to the next one.

3. Message Processing

As mentioned in Section 1, this document introduces the following two main deviations from the original OSCORE specification [RFC8613].

1. An "OSCORE endpoint", i.e., a producer/consumer of an OSCORE Option can be not only an application endpoint (i.e., an origin client or server), but also an intermediary such as a proxy.

   Hence, OSCORE can 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.

Appendix A provides a number of examples where the approach defined in this document is used to protect message exchanges.

3.1. General Rules on Protecting Options

Let us consider a sender endpoint that, when protecting an outgoing message, applies the i-th OSCORE layer in sequence, by using the OSCORE Security Context shared with another OSCORE endpoint X.

In addition to the CoAP options already specified as class E in [RFC8613] or in the document defining them, the sender endpoint MUST encrypt and integrity-protect the following CoAP options, even though they are originally specified as class U or class I for

OSCORE. That is, such options are processed like if they were specified as class E for OSCORE.

* An OSCORE Option, which is present as the result of the j-th OSCORE layer immediately previously applied, i.e., j = (i-1).

* The EDHOC Option defined in [I-D.ietf-core-oscore-edhoc], only if it is not intended to be consumed by the other OSCORE endpoint X.

  That is, the EDHOC Option is not protected when actually intended to be consumed by the other OSCORE endpoint X. In such a case, the EDHOC Option will still correctly signal to the other endpoint X to extract part of the message payload, and to use it for completing an ongoing execution of the EDHOC key establishment protocol [I-D.ietf-lake-edhoc], before proceeding with the removal of the i-th OSCORE layer.

* Any CoAP option such that both the following conditions hold, thus ensuring that as many options as possible are protected.

  1. The option is intended to be consumed by the other OSCORE endpoint X.

  2. At the other OSCORE endpoint X, the option does not play a role in processing the message before having removed the i-th OSCORE layer or in removing the i-th OSCORE layer altogether.

  Examples of such CoAP options are:

  - The Proxy-Uri, Proxy-Scheme, Uri-Host and Uri-Port Options defined in [RFC7252].

  - The Listen-To-Multicast-Notifications Option defined in [I-D.ietf-core-observe-multicast-notifications].

  - The Multicast-Timeout, Response-Forwarding and Group-ETag Options defined in [I-D.tiloca-core-groupcomm-proxy].

3.2. Processing an Outgoing Request

The rules from Section 3.1 apply when processing an outgoing request message, with the following addition.

When an application endpoint applies multiple OSCORE layers in sequence to protect an outgoing request, and it uses an OSCORE Security Context shared with the other application endpoint, then the first OSCORE layer MUST be applied by using that Security Context.

3.3. Processing an Incoming Request

Upon receiving a request REQ, the recipient endpoint performs the actions described in the following steps.

1. If REQ includes proxy-related options, the endpoint moves to step 2. Otherwise, the endpoint moves to step 3.

2. The endpoint proceeds as defined below, depending on which of the following conditions holds.

   * REQ includes either the Proxy-Uri Option, or the Proxy-Scheme Option together with any of the Uri-* Options.

     If the endpoint is not configured to be a forward-proxy, it MUST stop processing the request and MUST respond with a 5.05 (Proxying Not Supported) error response to (the previous hop towards) the origin client, as per Section 5.10.2 of [RFC7252]. This may result in protecting the error response over that communication leg, as per Section 3.4.

     Otherwise, the endpoint consumes the proxy-related options as per Section 5.7.2 of [RFC7252], 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.

     In either case, the endpoint does not take any further action.

   * REQ includes one or more Uri-Path Options but not the Proxy-Scheme Option.

     If the endpoint is not configured to be a reverse-proxy or its resource targeted by the Uri-Path Options is not intended to support reverse-proxy functionalities, then the endpoint proceeds to step 3.

     Otherwise, the endpoint consumes the Uri-Path options as per Section 5.7.3 of [RFC7252], 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.

     After that, the endpoint does not take any further action.

     Note that, when forwarding REQ, the endpoint might not remove all the Uri-Path Options originally present, e.g., in

case the next hop towards the origin server is a further reverse-proxy.

3. The endpoint proceeds as defined below, depending on which of the following conditions holds.

   *REQ does not include an OSCORE Option.

   If the endpoint does not have an application to handle REQ, it MUST stop processing the request and MAY respond with a 4.00 (Bad Request) error response to (the previous hop towards) the origin client. This may result in protecting the error response over that communication leg, as per Section 3.4.

   Otherwise, the endpoint delivers REQ to the application.

   *REQ includes an OSCORE Option.

   If REQ includes any URI-Path Options, the endpoint MUST stop processing the request and MAY respond with a 4.00 (Bad Request) error response to (the previous hop towards) the origin client. This may result in protecting the error response over that communication leg, as per Section 3.4.

   The endpoint decrypts REQ using the OSCORE Security Context indicated by the OSCORE Option, i.e., REQ* = dec(REQ). After that, the possible presence of an OSCORE Option in the decrypted request REQ* is not treated as an error situation.

   If the OSCORE processing results in an error, the endpoint MUST stop processing the request and performs error handling as per Section 8.2 of [RFC8613] or Sections 8.2 and 9.4 of [I-D.ietf-core-oscore-groupcomm], in case OSCORE or Group OSCORE is used, respectively. In case the endpoint sends an error response to (the previous hop towards) the origin client, this may result in protecting the error response over that communication leg, as per Section 3.4.

   Otherwise, REQ takes REQ*, and the endpoint moves to step 1.

3.4. Processing an Outgoing Response

The rules from Section 3.1 apply when processing an outgoing response message, with the following additions.

When an application endpoint applies multiple OSCORE layers in sequence to protect an outgoing response, and it uses an OSCORE Security Context shared with the other application endpoint, then

the first OSCORE layer MUST be applied by using that Security Context.

The sender endpoint protects the response by applying the same OSCORE layers that it removed from the corresponding incoming request, but in the reverse order than the one they were removed.

In case the response is an error response, the sender endpoint protects it by applying the same OSCORE layers that it successfully removed from the corresponding incoming request, but in the reverse order than the one they were removed.

3.5. Processing an Incoming Response

The recipient endpoint removes the same OSCORE layers that it added when protecting the corresponding outgoing request, but in the reverse order than the one they were removed.

When doing so, the possible presence of an OSCORE Option in the decrypted response following the removal of an OSCORE layer is not treated as an error situation, unless it occurs after having removed as many OSCORE layers as were added in the outgoing request. In such a case, the endpoint MUST stop processing the response.

4. Caching of OSCORE-Protected Responses

Although not possible as per the original OSCORE specification [RFC8613], cacheability of OSCORE-protected responses at proxies can be achieved. To this end, the approach defined in [I-D.amsuess-core-cachable-oscore] can be used, as based on Deterministic Requests protected with the pairwise mode of Group OSCORE [I-D.ietf-core-oscore-groupcomm] used end-to-end between an origin client and an origin server. The applicability of this approach is limited to requests that are safe (in the RESTful sense) to process and do not yield side effects at the origin server.

In particular, 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 step 2 in Section 3.3 produces a cache hit.

  That is, such a message: i) is exactly the one to be forwarded to (the next hop towards) the origin server if no cache hit has occurred; and ii) is the result of an OSCORE decryption at the proxy, if OSCORE is used on the communication leg between the proxy and (the previous hop towards) the origin client.

* Upon receiving a response from (the next hop towards) the origin server, the proxy first removes the same OSCORE layers that it added when protecting the corresponding outgoing request, as defined in Section 3.5.

  Then, the proxy stores specifically that resulting response message in its cache. That is, such a message is exactly the one to be forwarded to (the previous hop towards) the origin client.

The specific rules about serving a request with a cached response are defined in Section 5.6 of [RFC7252], as well as in Section 7 of [I-D.tiloca-core-groupcomm-proxy] for group communication scenarios.

5. Security Considerations

TODO

6. IANA Considerations

This document has no actions for IANA.

7. References

7.1. 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-15, 5 September 2022, <https://www.ietf.org/archive/id/draft-ietf-core-oscore-groupcomm-15.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>.

7.2. 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-05, 11 July 2022, <https://www.ietf.org/archive/id/draft-amsuess-core-cachable-oscore-05.txt>.

[I-D.ietf-core-coap-pubsub] Koster, M., Keränen, A., and J. Jimenez, "Publish-Subscribe Broker for the Constrained Application Protocol (CoAP)", Work in Progress, Internet-Draft, draft-ietf-core-coap-pubsub-10, 4 May 2022, <https://www.ietf.org/archive/id/draft-ietf-core-coap-pubsub-10.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-07, 11 July 2022, <https://www.ietf.org/archive/id/draft-ietf-core-groupcomm-bis-07.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-04, 11 July 2022, <https://www.ietf.org/archive/id/draft-ietf-core-observe-multicast-notifications-04.txt>.

[I-D.ietf-core-oscore-edhoc] Palombini, F., Tiloca, M., Höglund, R., Hristozov, S., and G. Selander, "Profiling EDHOC for CoAP and OSCORE", Work in Progress, Internet-Draft, draft-

ietf-core-oscore-edhoc-04, 11 July 2022, <https://www.ietf.org/archive/id/draft-ietf-core-oscore-edhoc-04.txt>.

[I-D.ietf-core-transport-indication] Amsüss, C., "CoAP Protocol Indication", Work in Progress, Internet-Draft, draft-ietf-core-transport-indication-01, 11 July 2022, <https://www.ietf.org/archive/id/draft-ietf-core-transport-indication-01.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-15, 10 July 2022, <https://www.ietf.org/archive/id/draft-ietf-lake-edhoc-15.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-07, 5 September 2022, <https://www.ietf.org/archive/id/draft-tiloca-core-groupcomm-proxy-07.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-Gateway] Open Mobile Alliance, "Lightweight Machine to Machine Gateway Technical Specification - Approved Version 1.1, OMA-TS-LWM2M_Gateway-V1_1-20210518-A", May 2021, <https://www.openmobilealliance.org/release/LwM2M_Gateway/V1_1-20210518-A/OMA-TS-LWM2M_Gateway-V1_1-20210518-A.pdf>.

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

[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>.

[RFC8742] Bormann, C., "Concise Binary Object Representation (CBOR) Sequences", RFC 8742, DOI 10.17487/RFC8742, February 2020, <https://www.rfc-editor.org/info/rfc8742>.

Appendix A. Examples

This section provides a number of examples where the approach defined in this document is used to protect message exchanges.

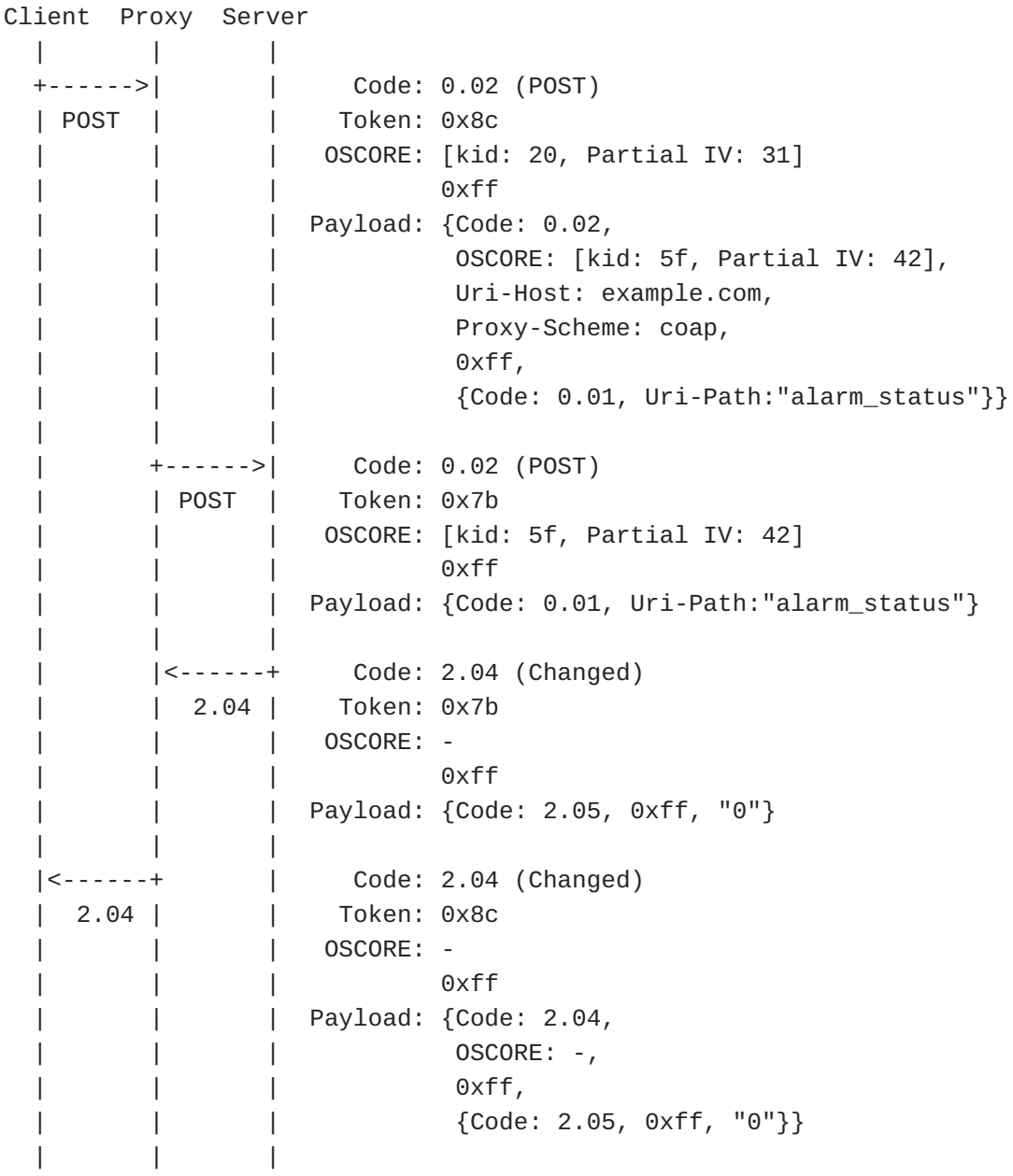
A.1. Example 1

The example in Figure 1 builds on the example from Appendix A.1 of [RFC8613], and illustrates an origin client requesting the alarm status from an origin server, through a forward-proxy.

The message exchanges are protected with OSCORE over the following legs.

* End-to-end, between the client and the server. The client uses the OSCORE Sender ID 0x5f when using OSCORE with the server.

* Between the client and the proxy. The client uses the OSCORE Sender ID 0x20 when using OSCORE with the proxy.



Square brackets [...] indicate content of compressed COSE object.
Curly brackets { ... } indicate encrypted data.

Figure 1: Use of OSCORE between Client-Server and Client-Proxy

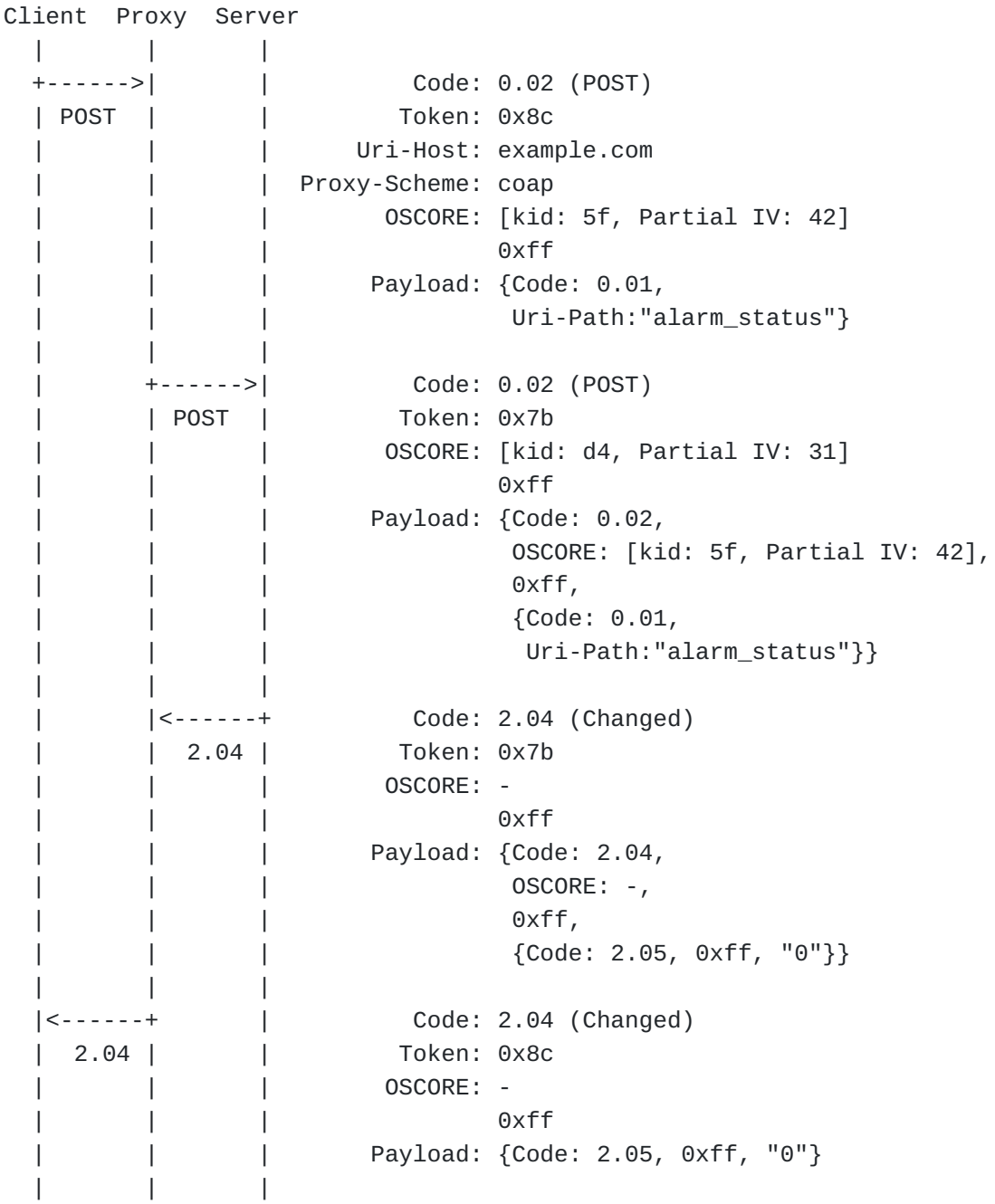
A.2. Example 2

The example in Figure 2 builds on the example from Appendix A.1 of [RFC8613], and illustrates an origin client requesting the alarm status from an origin server, through a forward-proxy.

The message exchanges are protected with OSCORE over the following legs.

- End-to-end between the client and the server. The client uses the OSCORE Sender ID 0x5f when using OSCORE with the server.

- Between the proxy and the server. The proxy uses the OSCORE Sender ID 0xd4 when using OSCORE with the server.



Square brackets [...] indicate content of compressed COSE object.
Curly brackets { ... } indicate encrypted data.

Figure 2: Use of OSCORE between Client-Server and Proxy-Server

A.3. Example 3

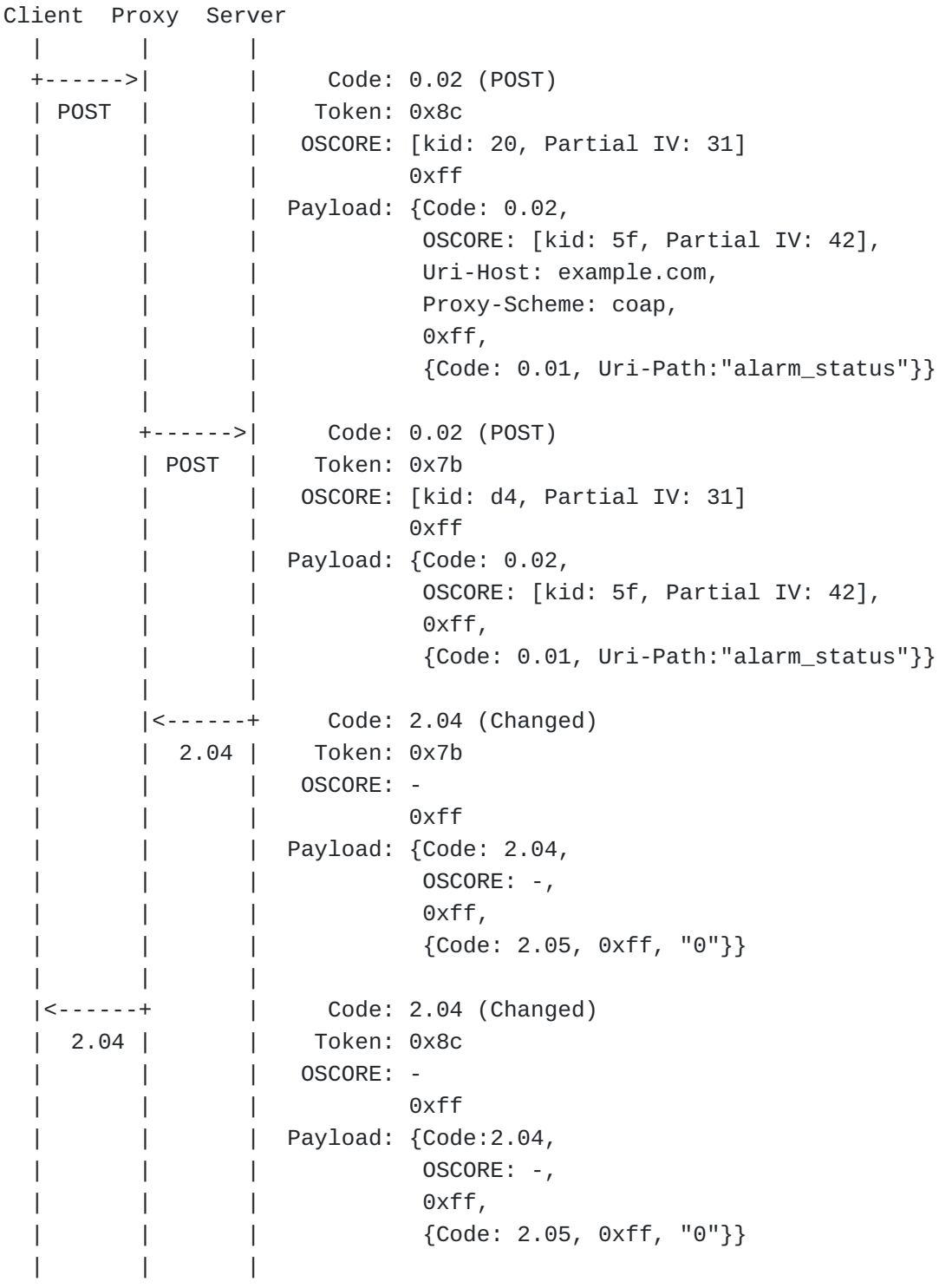
The example in Figure 3 builds on the example from Appendix A.1 of [RFC8613], and illustrates an origin client requesting the alarm status from an origin server, through a forward-proxy.

The message exchanges are protected with OSCORE over the following legs.

* End-to-end between the client and the server. The client uses the OSCORE Sender ID 0x5f when using OSCORE with the server.

* Between the client and the proxy. The client uses the OSCORE Sender ID 0x20 when using OSCORE with the proxy.

* Between the proxy and the server. The proxy uses the OSCORE Sender ID 0xd4 when using OSCORE with the server.



Square brackets [...] indicate content of compressed COSE object.
Curly brackets { ... } indicate encrypted data.

Figure 3: Use of OSCORE between Client-Server, Client-Proxy and Proxy-Server

A.4. Example 4

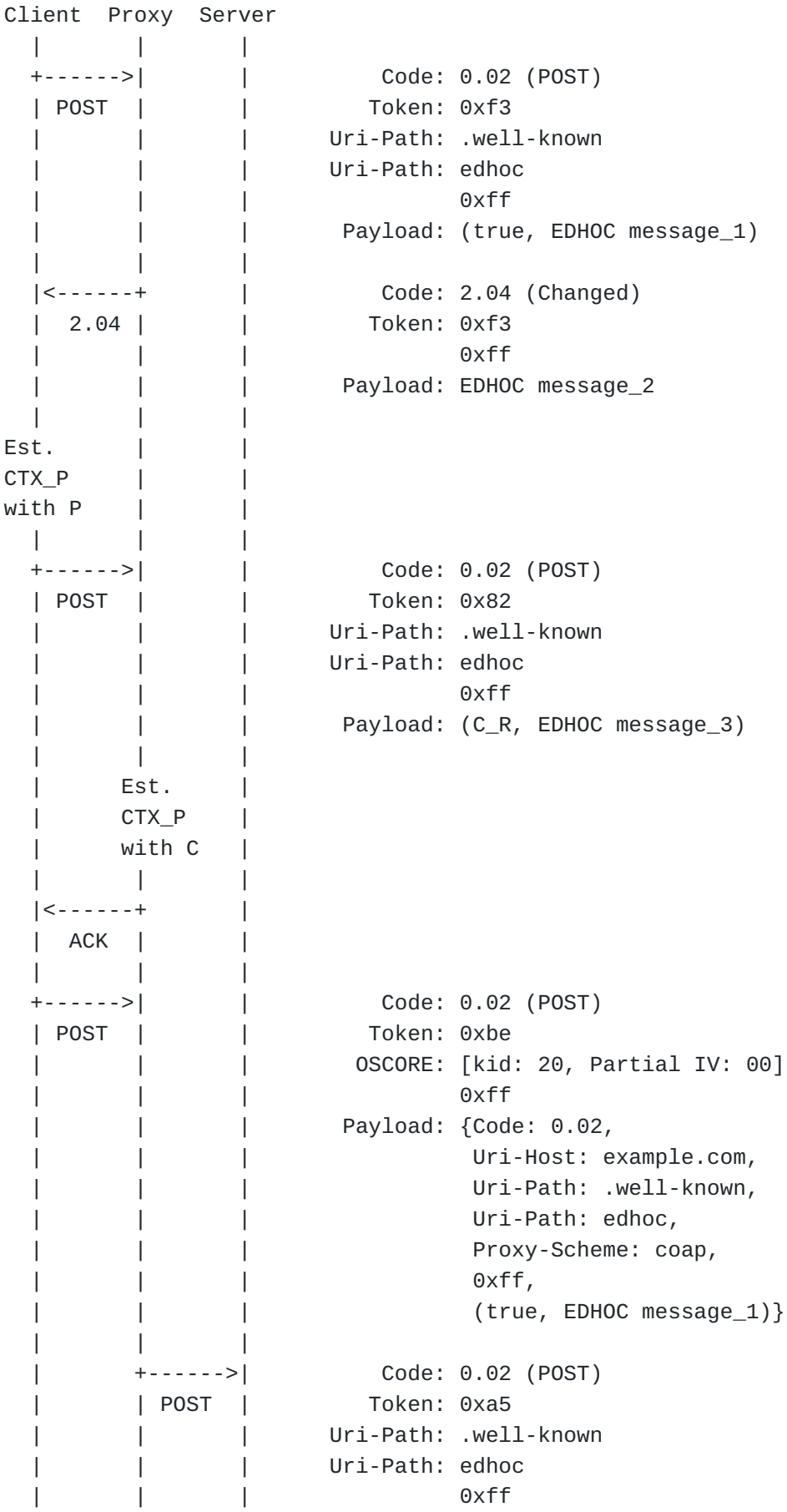
The example in Figure 4 builds on the example from Appendix A.1 of [RFC8613], and illustrates an origin client requesting the alarm status from an origin server, through a forward-proxy.

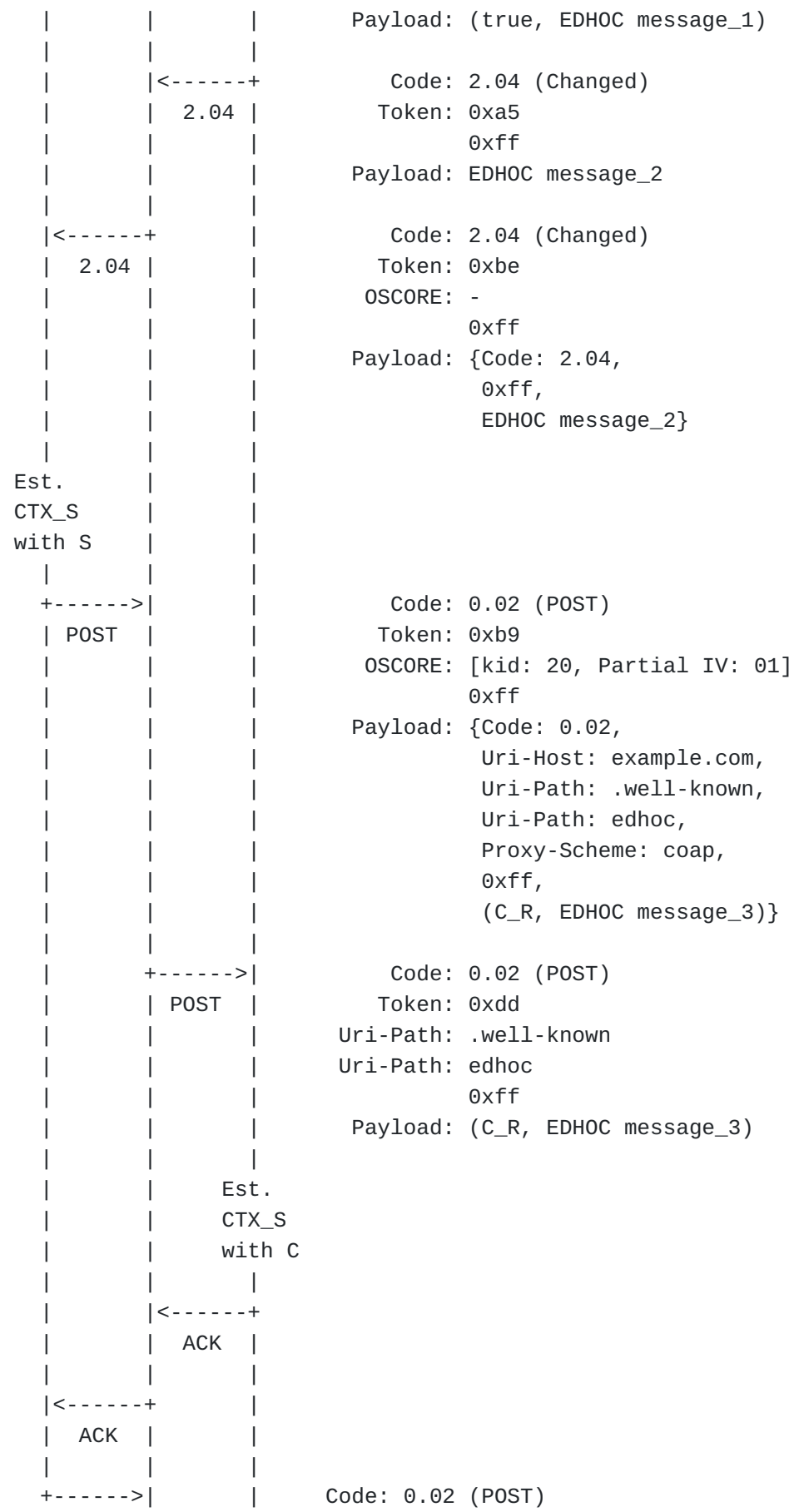
The message exchanges are protected over the following legs.

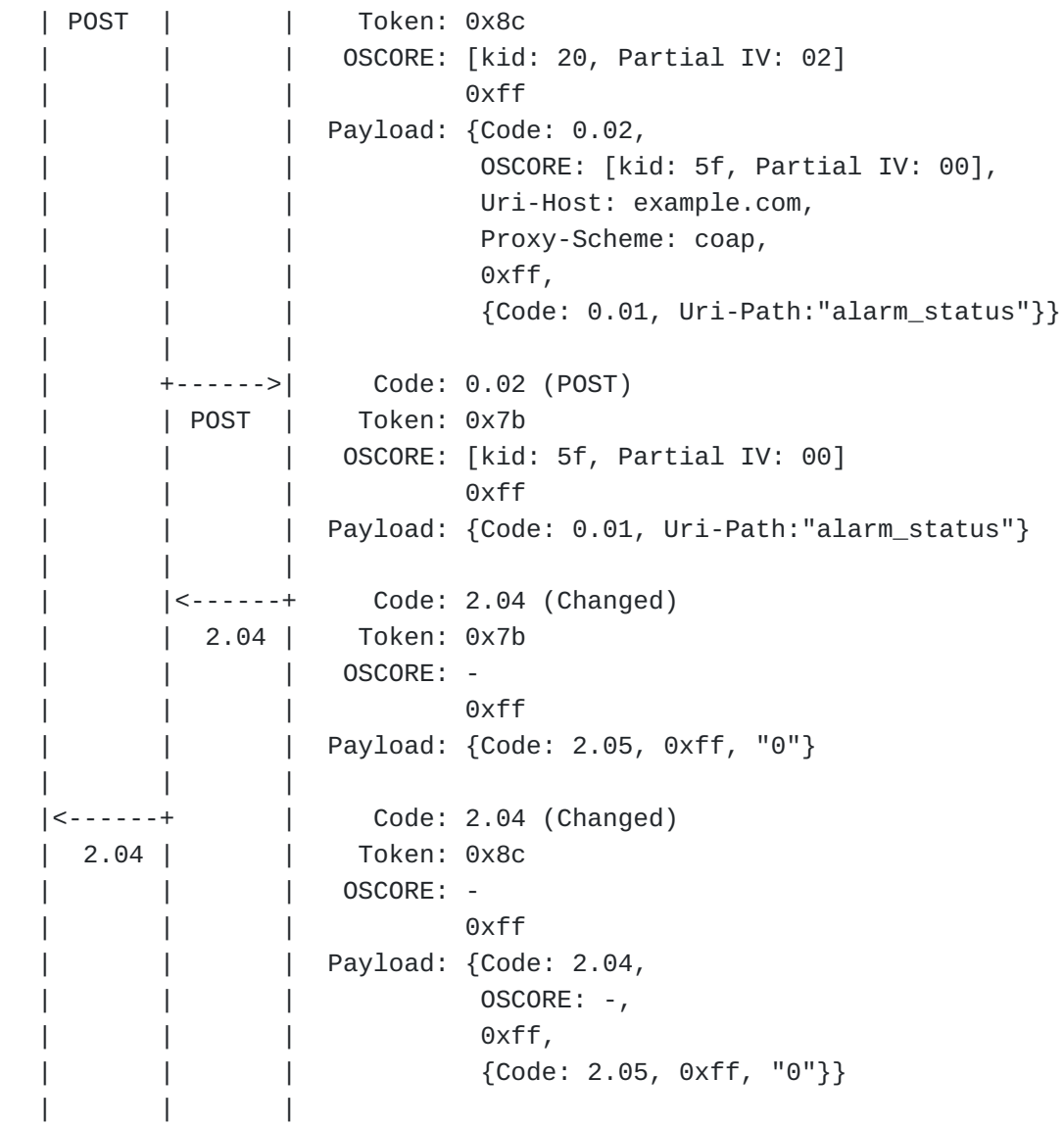
- End-to-end, between the client and the server. The client uses the OSCORE Sender ID 0x5f when using OSCORE with the server.

- Between the client and the proxy. The client uses the OSCORE Sender ID 0x20 when using OSCORE with the proxy.

The example also shows how the client establishes an OSCORE Security Context with the proxy and with the server, by using the key establishment protocol EDHOC [I-D.ietf-lake-edhoc].


Client Proxy Server
POST
Code: 0.02 (POST)
Token: 0xf3
Uri-Path: .well-known
Uri-Path: edhoc
0xff
Payload: (true, EDHOC message_1)
2.04
Code: 2.04 (Changed)
Token: 0xf3
0xff
Payload: EDHOC message_2
Est.
CTX_P
with P
POST
Code: 0.02 (POST)
Token: 0x82
Uri-Path: .well-known
Uri-Path: edhoc
0xff
Payload: (C_R, EDHOC message_3)
Est.
CTX_P
with C
ACK
POST
Code: 0.02 (POST)
Token: 0xbe
OSCORE: [kid: 20, Partial IV: 00]
0xff
Payload: {Code: 0.02,
Uri-Host: example.com,
Uri-Path: .well-known,
Uri-Path: edhoc,
Proxy-Scheme: coap,
0xff,
(true, EDHOC message_1)}
POST
Code: 0.02 (POST)
Token: 0xa5
Uri-Path: .well-known
Uri-Path: edhoc
0xff


Payload: (true, EDHOC message_1)
Code: 2.04 (Changed)
2.04
Token: 0xa5
0xff
Payload: EDHOC message_2
Code: 2.04 (Changed)
Token: 0xbe
OSCORE: -
0xff
Payload: {Code: 2.04,
0xff,
EDHOC message_2}
Est.
CTX_S
with S
Code: 0.02 (POST)
POST
Token: 0xb9
OSCORE: [kid: 20, Partial IV: 01]
0xff
Payload: {Code: 0.02,
Uri-Host: example.com,
Uri-Path: .well-known,
Uri-Path: edhoc,
Proxy-Scheme: coap,
0xff,
(C_R, EDHOC message_3)}
Code: 0.02 (POST)
POST
Token: 0xdd
Uri-Path: .well-known
Uri-Path: edhoc
0xff
Payload: (C_R, EDHOC message_3)
Est.
CTX_S
with C
ACK
ACK
Code: 0.02 (POST)



Square brackets [...] indicate content of compressed COSE object.
Curly brackets { ... } indicate encrypted data.
Round brackets (...) indicate a CBOR sequence [RFC 8742].

Figure 4: Use of OSCORE between Client-Server and Proxy-Server, with OSCORE Security Contexts established through EDHOC

Appendix B. 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.

If the EDHOC key establishment protocol is used (see [I-D.ietf-lake-edhoc]), 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 4 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:

  - 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).

  - 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).

  - 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 Amsüss, Peter Blomqvist, David Navarro and Göran Selander for their comments and feedback.

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

Authors' Addresses

Marco Tiloca
RISE AB
Isafjordsgatan 22
SE-16440 Kista
Sweden

Email: marco.tiloca@ri.se

Rikard Höglund
RISE AB
Isafjordsgatan 22
SE-16440 Kista
Sweden

Email: rikard.hoglund@ri.se