CoRE Working Group
Internet-Draft
Updates: 8613 (if approved)
Intended status: Standards Track
Expires: 13 January 2022

M. Tiloca
R. Hoeglund
RISE AB

12 July 2021

OSCORE-capable Proxies
draft-tiloca-core-oscore-capable-proxies-00

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 OSCORE is used to protect CoAP messages also between an origin application endpoint and an intermediary, or between two intermediaries. Besides, it defines how a CoAP message can be double-protected through "OSCORE-in-OSCORE", i.e., 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 applies to Group OSCORE, for protecting CoAP messages when group communication with intermediaries is used.

Discussion Venues

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

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

Source for this draft and an issue tracker can be found at https://gitlab.com/crimson84/draft-tiloca-core-oscore-to-proxies.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

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 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, i.e., by considering the intermediary as an "OSCORE endpoint".

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

* It defines how OSCORE is used to protect 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 explicitly admits a CoAP message to be double-protected through "OSCORE-in-OSCORE". This is the case when the message is first OSCORE-protected end-to-end between the origin client and origin server, and then 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).

What defined in this document is applicable 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 to the following terms.

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

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

* Application endpoint: a source or destination application endpoint.

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

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

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

* Proxy-related option: the Proxy-URI Option, the Proxy-Scheme Option, or any of the Uri-* Options.

* OSCORE-in-OSCORE: the process by which a message protected with (Group) OSCORE is further protected with (Group) OSCORE. This means that, after the first OSCORE decryption/verification, the resulting message is again an OSCORE-protected message.

2. Use Cases

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

2.1. UC1 - 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. UC2 - 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".

In case the clients and the server use Group OSCORE for end-to-end security and a proxy is also involved, an additional step is required (see Section 10 of [I-D.ietf-core-observe-multicast-notifications]). That is, clients are in turn required to provide the proxy with the obtained "phantom observation request", thus enabling the proxy to receive the multicast notifications from the server.

Therefore, it is preferable to have a security associations also between each client and the proxy, to especially ensure the integrity of that information provided to the proxy (see Section 13.1 of [I-D.ietf-core-observe-multicast-notifications]). Like for the use case UC1 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. UC3 - 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 UC1 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.

3. Possible Configurations

This section provides an overview of different security configurations referred in the document. The configurations differ on whether OSCORE is used or not in a certain communication leg.

For simplicity, only one intermediary is considered, as a CoAP-to-CoAP forward-proxy standing between one CoAP client and one CoAP server. The same can be extended to cover a chain of intermediaries, or a group communication scenario where CoAP requests are intended to multiple servers and possibly protected end-to-end with Group OSCORE.

The used notation denotes the origin client with C, the origin server with S and the proxy with P. Each configuration is denoted by CF-x, with x a positive integer number expressed by the bits (b2, b1, b0), i.e., x = b0 + (2 * b1) + (4 * b2).

In particular, for each configuration CF-x:

* C and P use OSCORE between themselves if and only if b0 = 1.

* P and S use OSCORE between themselves if and only if b1 = 1.

* C and S use OSCORE between themselves if and only if b2 = 1.

For convenience, the configurations are split into two sets. That is, Section 3.1 overviews those where C and S do not use OSCORE between themselves (b2 = 0), while Section 3.2 overviews those where C and S use OSCORE between themselves (b2 = 1).

3.1. Configurations without End-to-End Security

Figure 1 shows the different configurations where OSCORE is not used end-to-end by the two application endpoints C and S. That is, none of the shown configurations include the leg "C-S", i.e., b2 = 0.

In the configurations CF-1, CF-2 and CF-3, the proxy uses OSCORE with C and/or S, hence the legs "C-P" and/or "P-S" are also included. None of these configurations results in using "OSCORE-in-OSCORE".

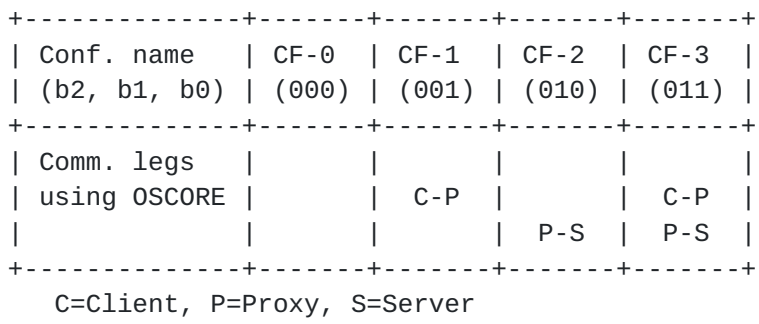

Figure 1: Configurations without end-to-end security.

Note that:

* CF-0 is the canonical case defined in [RFC7252] with no security at the application layer.

* CF-1 is relevant for the use cases UC1 and UC3 in Section 2, where end-to-end security is not provided between client and server(s). Instead, it is not relevant for UC2, since that use case would require secure communication between clients and proxy only when Group OSCORE is also used for end-to-security between the observer clients and the server (see Section 2.2).

* CF-2 can be seen as the canonical case of [RFC8613], with P acting as client with S.

* CF-3 can be seen as the canonical case of [RFC8613], applied separately to the two legs "C-P" (with P acting as server) and "P-S" (with P acting as client).

3.2. Configurations with End-to-End Security

Figure 2 shows the different configurations where OSCORE is used end-to-end by the two application endpoints C and S. That is, all the shown configurations include the leg "C-S", i.e., b2 = 1.

In the configurations CF-5, CF-6 and CF-7, the proxy uses OSCORE with C and/or S, hence the legs "C-P" and/or "P-S" are also included. Therefore, these configurations result in double-protecting the CoAP messages originated at C and S, and are thus marked with "(*)" to indicate the use of "OSCORE-in-OSCORE".

Conf. name (b2, b1, b0)	CF-4 (100)	CF-5 (101)	CF-6 (110)	CF-7 (111)
Comm. legs using OSCORE	C-S	C-S C-P (*)	C-S P-S (*)	C-S C-P (*) P-S (*)

C=Client, P=Proxy, S=Server
(*) OSCORE-in-OSCORE

Figure 2: Configurations with end-to-end security.

Note that:

* CF-4 is the canonical case defined in [RFC8613].

* CF-5 is relevant for the use cases UC1, UC2 and UC3 in Section 2.

* CF-7 is relevant for the use cases UC2 and UC3 in Section 2. Instead, it is not relevant for UC1, since that would imply that P is also a member of the OSCORE group including the servers, which is not desirable (see Section 2.1).

4. Request Processing

This section extends the actions performed to protect an outgoing request, with respect to [RFC8613] or [I-D.ietf-core-oscore-groupcomm] when OSCORE or Group OSCORE is used, respectively. Throughout the text, the configurations defined in Section 3 are also recalled by means of the bits (b2, b1, b0), to indicate the communication leg(s) using OSCORE.

The following assumes the presence of a single intermediary acting as CoAP-to-CoAP Forward-Proxy between the origin CoAP client and a single origin CoAP server.

4.1. Protecting the Request at the Client

The client performs the following steps.

1. The client prepares the CoAP request REQ for the origin server, which is the intended destination application endpoint.

2. Since the client knows that a proxy is involved as next hop, the client adds the appropriate proxy-related options to the request.

3. If the client uses (Group) OSCORE with the origin server (b2 = 1), it performs the following actions. Otherwise, it moves to step 4.

   * The client protects the request REQ as it normally happens with OSCORE or Group OSCORE, using the Security Context shared with the server.

   * The result is a protected request REQ* including an OSCORE option. The intended destination OSCORE endpoint is the origin server.

   * REQ takes REQ*, and the client moves to step 4.

4. If the client uses (Group) OSCORE with the proxy (b0 = 1), it performs the following actions. Otherwise, it moves to step 5.

   * The client protects the request REQ with (Group) OSCORE, using the Security Context shared with the Proxy. Note that the Proxy-Uri option, if present at this point in time (b2 = 0 and b0 = 1), is first decomposed as per Section 4.1.3.3 of [RFC8613].

     Unlike in [RFC8613], the following options are processed as Class E, if present.

     - Proxy-Scheme, Uri-Host, Uri-Port, Uri-Path and Uri-Query.

     - OSCORE, which is present if (Group) OSCORE is used between the origin client and the origin server (i.e., b2 = 1 and b0 = 1, hence step 3 above was performed). This is "OSCORE-in-OSCORE" and deviates from [RFC8613].

     - Multicast-Signaling Option, defined in [I-D.tiloca-core-groupcomm-proxy].

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

     - Any other option that is intended to be accessed and consumed by the proxy.

   * The result is a protected request REQ** including its own outer OSCORE option, i.e., REQ** = Enc(REQ). The intended destination OSCORE endpoint is the proxy.

   * REQ takes REQ**, and the client moves to step 5.

5. The client sends the request REQ to the proxy.

4.2. Verifying the Request at the Proxy

The proxy performs the following steps.

1. The proxy receives the request REQ from the origin client.

2. The proxy assesses which alternative it is.

   * Alt 2.1 - The request REQ includes proxy-related options as already visible to the proxy (b0 = 0). Hence, the request has to be forwarded to the origin server.

     In this case, the proxy moves to step 4.

     Note that:

     - The request REQ includes an OSCORE option if the proxy is neither the destination application endpoint nor the destination OSCORE endpoint (b2 = 1 and b0 = 0).

     - The request REQ does not include an OSCORE option if the proxy is not the destination application endpoint and no OSCORE is involved (b2 = 0 and b0 = 0).

   * Alt 2.2 - The request REQ does not include proxy-related options but it includes an OSCORE option, as visible to the proxy (b0 = 1). Hence, the proxy is the destination OSCORE endpoint.

     In this case, the proxy moves to step 3.

     Note that, at this point in time, the proxy does not know yet if it is going to act as proxy by forwarding the request, or if it is instead the actual destination application endpoint.

   * Alt 2.3 - The request REQ does not include proxy-related options and it does not include an OSCORE option, as visible to the proxy (b2 = 0 and b0 = 0). Hence, the proxy is the destination application endpoint with no OSCORE is involved.

     In this case, the proxy delivers the request REQ to the application for processing.

3. Coming from alternative 2.2 of step 2, the proxy unprotects the request REQ using the OSCORE Security Context shared with the origin client.

In case of an OSCORE-related error, the proxy responds to the client with an unprotected error response. Following a successful processing, i.e. REQ* = Dec(REQ), three alternatives are possible.

* Alt 3.1 - The decrypted request REQ* includes proxy-related options. Hence, the request has to be forwarded to the origin server.

  In this case, REQ takes REQ*, and the proxy moves to step 4.

  Note that:

  - If the decrypted request REQ* includes an OSCORE option as well as proxy-related options, then the proxy is not the destination application endpoint, and OSCORE is used end-to-end between the origin client and origin server (b2 = 1 and b0 = 1). This is "OSCORE-in-OSCORE" and deviates from [RFC8613], i.e., under this specific outcome, it is fine for the proxy to find an OSCORE option in the decrypted request REQ*, and the message is not rejected.

  - If the decrypted request REQ* does not include an OSCORE option, but it includes proxy-related options, then the proxy is not the destination application endpoint, and OSCORE is not used end-to-end between the origin client and origin server (b2 = 0 and b0 = 1).

* Alt 3.2 - The decrypted request REQ* does not include an OSCORE option, and it does not include any proxy-related option. Hence, the proxy is the actual destination application endpoint.

  In this case, the proxy delivers the decrypted request REQ* to the application for processing. The possible following response to the client occurs as per [RFC8613].

* Alt 3.3 - The decrypted request REQ* includes an OSCORE option, but it does not include any proxy-related option.

  This alternative is not valid, and the proxy MUST respond to the client with an error response. This response MUST be protected with the (Group) OSCORE Security Context shared with the client.

4. The proxy proceeds with the forwarding of the request REQ to the origin server (see Section 4.3).

4.3. Forwarding the Request to the Server

The proxy performs the following steps.

1. The proxy sets the Request URI in the request REQ, by consuming the Proxy-Uri or the Proxy-Scheme option as per Section 5.7.2 of [RFC7252].

2. If the proxy uses OSCORE with the origin server (b1 = 1), the proxy performs the following actions. Otherwise, the proxy moves to step 3.

   * The proxy protects the request REQ with OSCORE, i.e. REQ* = Enc(REQ), using the Security Context shared with the origin server.

     Unlike in [RFC8613], the OSCORE option, if present in the request REQ to protect, is processed as Class E. This is the case if (Group) OSCORE is used between the origin client and the origin server (b2 = 1 and b1 = 1). This is "OSCORE-in-OSCORE" and deviates from [RFC8613].

   * The result is a protected request REQ* including an OSCORE option. The intended destination OSCORE endpoint is the origin server.

   * REQ takes REQ*, and the proxy moves to step 3.

3. The proxy forwards the request REQ to the origin server.

4.4. Verifying the Request at the Server

The server performs the following steps.

1. The server receives the request REQ from the proxy.

   In the considered scenario, the server is the actual origin server, hence it never encounters proxy-related options when processing the request (i.e., neither as outer nor as inner options). This means that the server is the destination OSCORE endpoint with respect to any OSCORE option it encounters when processing the request.

2. The server assesses which alternative it is.

   * Alt 2.1 - The request REQ does not include an OSCORE option (b2 = 0 and b1 = 0).

If proxy-related options are found at this point in time, this endpoint is not the actual destination application endpoint, and it should follow what described in Section 4.2, if it supports acting as proxy.

Otherwise, the server is the destination application endpoint. Hence, the server moves to step 5.

* Alt 2.2 - The request REQ includes an OSCORE option (b2 = 1 or b1 = 1).

  In this case, the server is the destination OSCORE endpoint. Hence, the server moves to step 3.

3. Coming from alternative 2.2 of step 2, the server unprotects the request REQ using the OSCORE Security Context pointed by the OSCORE option, i.e. REQ* = Dec(REQ).

   In case of an OSCORE-related error, the server responds to the proxy with an unprotected error response. Instead, following a successful processing, the two valid alternatives below are possible.

   If proxy-related options are found at this point in time, this endpoint is not the actual destination application endpoint. Hence, it should rather follow what described in Section 4.2, if it supports acting as proxy.

   * Alt 3.1 - The decrypted request REQ* does not include an OSCORE option (b2 XOR b1 = 1). Hence, the server is the destination application endpoint.

     In this case, REQ takes REQ*, and the server moves to step 5.

   * Alt 3.2 - The decrypted request REQ* includes an OSCORE option (b2 = 1 and b1 = 1). Hence, since the decrypted request REQ* does not include any proxy-related option, the server is the destination application endpoint, but it has to perform the actual end-to-end OSCORE decryption first.

     In this case, the server moves to step 4.

     Note that, this is "OSCORE-in-OSCORE" and deviates from [RFC8613], i.e., under this specific outcome, it is fine for the server to find an OSCORE option in the decrypted message, and the message is not rejected.

4. Coming from alternative 3.2 of step 3, the server unprotects the inner request REQ* using the OSCORE Security Context pointed by the OSCORE option in REQ*, i.e. REQ** = Dec(REQ*).

   If any of the following cases occurs, the server MUST discard the request and MUST reply with an error response, which MUST be protected using the OSCORE Security Context shared with the proxy.

   * An OSCORE-related error occurred when unprotecting REQ*, i.e. when using the OSCORE Security Context shared with the client.

   * The decrypted request REQ** includes any proxy-related option. That is, proxy-related options cannot appear after having performed OSCORE decryption twice.

   * The decrypted request REQ** includes an OSCORE option. That is, at most two layers of OSCORE protection are admitted, i.e. one with the other application endpoint and one with the adjacent transport hop.

   Otherwise, REQ takes REQ**, and the server moves to step 5.

5. The server delivers the request REQ to the application for processing. The possible following response to the proxy will happen as per [RFC7252], and possibly [RFC8613] or [I-D.ietf-core-oscore-groupcomm].

5. Response Processing

Following up on the process in Section 4, this section extends the actions performed to protect an outgoing response, with respect to [RFC8613] or [I-D.ietf-core-oscore-groupcomm] when OSCORE or Group OSCORE is used, respectively. Throughout the text, the configurations defined in Section 3 are also recalled by means of the bits (b2, b1, b0), to indicate the communication leg(s) using OSCORE.

The following assumes the presence of a single intermediary acting as CoAP-to-CoAP Forward-Proxy between the origin CoAP client and a single origin CoAP server.

5.1. Protecting the Response at the Server

The server performs the following steps.

1. The server possibly prepares a CoAP response RESP to send to the proxy, to be forwarded to the origin client as intended destination application endpoint.

2. The server proceeds as follows.

   * If the server performed only one OSCORE processing when receiving the request (b2 XOR b1 = 1), it moves to step 3.

   * If the server performed two OSCORE processings when receiving the request (b2 = 1 and b1 = 1), it moves to step 4.

   * If the server performed no OSCORE processing when receiving the request (b2 = 0 and b1 = 0), it moves to step 5.

3. The server protects the response RESP using the same OSCORE Security Context used to unprotect the request REQ. The result is a protected response including an OSCORE option, i.e. RESP* = Enc(RESP).

   Then, RESP takes RESP*, and the server moves to step 5.

4. The server proceeds as follows.

   * 4.1. The server protects the response RESP using the same OSCORE Security Context used for the second and last processing of the request, i.e. the one shared with the origin client.

     The result is a protected response including an OSCORE option, i.e. RESP* = Enc(RESP).

   * 4.2. The server protects the encrypted response RESP* using the same OSCORE Security Context used for the first processing of the request, i.e. the one shared with the proxy as adjacent transport hop.

     The OSCORE option in the encrypted response RESP* is processed as class E. If present, any other option that is intended to be accessed and consumed by the proxy is also processed as Class E. This is "OSCORE-in-OSCORE" and deviates from [RFC8613].

     The result is a protected response including its own outer OSCORE option, i.e. RESP** = Enc(RESP*).

   4.3. RESP takes RESP**, and the server moves to step 5.

5. The server sends the response RESP to the proxy.

5.2. Verifying the Response at the Proxy

The proxy performs the following steps.

1. The proxy receives the response RESP from the origin server.

2. The proxy proceeds as follows.

   * If the proxy protected the request forwarded to the server by using OSCORE (b1 = 1), it moves to step 3.

   * If the proxy did not protect the request forwarded to the server by using OSCORE (b1 = 0), it moves to step 4.

3. The proxy proceeds as follows.

   If the received response RESP is unprotected, the proxy interprets it as an OSCORE-related error at the server. This concerns the OSCORE association between proxy and server, and it is up to the proxy to handle this issue.

   Otherwise, i.e., the received response RESP includes an outer OSCORE option, the proxy unprotects the response RESP, with the OSCORE Security Context shared with the server. This results in a decrypted response RESP* = Dec(RESP).

   If the OSCORE protection of the request performed by the proxy did not result in "OSCORE-in-OSCORE" (b2 = 0 and b1 = 1) --- see step 2 of Section 4.3 --- and the decrypted response RESP* includes an OSCORE option, then the proxy MUST discard the response.

   In any other case, RESP takes RESP*, and the proxy moves to step 4.

4. The proxy forwards the response RESP back to the origin client (see Section 5.3).

5.3. Forwarding the Response to the Client

The proxy performs the following steps.

1. The proxy adds possible options to the response RESP to be forwarded.

2. If the proxy uses (Group) OSCORE with the client, and the originally received request was protected with the OSCORE Security Context shared with the client (b0 = 1), then the proxy proceeds as follows.

   * The proxy protects the response RESP with OSCORE, i.e. RESP* = Enc(RESP), using the Security Context shared with the origin client. The following options, if present in the response RESP to protect, are processed as class E:

     - The OSCORE option. This is the case if (Group) OSCORE is used between the origin client and the origin server (b2 = 1 and b0 = 1). This is "OSCORE-in-OSCORE", which deviates from [RFC8613].

     - Any other option that has been added by the proxy, e.g. the Response-Forwarding Option defined in [I-D.tiloca-core-groupcomm-proxy].

   * The result is a protected response RESP* including an OSCORE option. The intended destination OSCORE endpoint is the origin client.

   * RESP takes RESP*, and the proxy moves to step 3.

3. The proxy forwards the response RESP back to the origin client.

5.4. Verifying the Response at the Client

The client performs the following steps.

1. The client receives the response RESP from the proxy.

2. The client proceeds as follows.

   * If the client protected the request with (Group) OSCORE using an OSCORE Security Context shared with the proxy (b0 = 1), it moves to step 3.

   * If the client did not protect the request with (Group) OSCORE using an OSCORE Security Context shared with the proxy (b0 = 0), it moves to step 4.

3. The client proceeds as follows.

   If the received response RESP is an unprotected 4.00/4.01/4.02, the client interprets it as an OSCORE-related error at the proxy. This concerns the OSCORE association between client and proxy,

and it is up to the client to handle this issue. Information specified in the diagnostic payload, if present, might result in the client taking alternative, more appropriate actions.

Otherwise, i.e. the received response RESP includes an outer OSCORE option, the client unprotects the response RESP, with the OSCORE Security Context shared with the proxy. This results in a decrypted response RESP* = Dec(RESP).

If the OSCORE protection of the request performed by the client did not result in "OSCORE-in-OSCORE" (b2 = 0 and b0 = 1) --- see step 4 of Section 4.1 --- and the decrypted response RESP* includes an OSCORE option, then the client MUST discard the message and does not perform any further processing.

If the OSCORE protection of the request performed by the client resulted in "OSCORE-in-OSCORE" (b2 = 1 and b0 = 1) --- see step 4 of Section 4.1 --- then the decrypted response RESP* may include an OSCORE option. This is "OSCORE-in-OSCORE" and deviates from [RFC8613], i.e., under this specific outcome, it is fine for the client to find an OSCORE option in the decrypted messages, and the message is not rejected.

Finally, RESP takes RESP*, and the client moves to step 4.

4. The client proceeds as follows.

   * If the client did not protect the original CoAP request with (Group) OSCORE using an OSCORE Security Context shared with the origin server (b2 = 0), the client moves to step 5.

   * If the client protected the original CoAP request with (Group) OSCORE using an OSCORE Security Context shared with the origin server (b2 = 1), the client proceeds as follows.

     If the response RESP is an unprotected 4.00/4.01/4.02, the client interprets it as an OSCORE-related error at the origin server. This concerns the OSCORE association between client and server, and it is up to the client to handle this issue. Information specified in the diagnostic payload, if present, might result in the client taking alternative, more appropriate actions.

     Otherwise, i.e. the response RESP includes an OSCORE option, the client unprotects the response RESP, with the OSCORE Security Context shared with the origin server. This results in a decrypted response RESP* = Dec(RESP).

The client MUST discard the decrypted response RESP*, in case RESP* is the result of two OSCORE decryptions in a row (b2 = 1 and b0 = 1) and includes an OSCORE option. That is, at most two layers of OSCORE protection are admitted, i.e. one with the other application endpoint and one with the adjacent transport hop.

Otherwise, RESP takes RESP*, and the client moves to step 5.

5. The client delivers the response RESP to the application for processing.

6. Response Caching

TBD

7. Chain of Intermediaries

TBD

8. Security Considerations

TBD

9. IANA Considerations

This document has no actions for IANA.

Acknowledgments

The authors sincerely thank Christian Amsuess, Peter Blomqvist and Goeran Selander for the initial discussions that allowed shaping this document.

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).

References

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-12, 12 July 2021, <https://www.ietf.org/archive/id/draft-ietf-core-oscore-groupcomm-12.txt>.

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

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

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

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

Informative References

[I-D.ietf-core-coap-pubsub]
Koster, M., Keranen, A., and J. Jimenez, "Publish-Subscribe Broker for the Constrained Application Protocol (CoAP)", Work in Progress, Internet-Draft, draft-ietf-core-coap-pubsub-09, 30 September 2019, <https://www.ietf.org/archive/id/draft-ietf-core-coap-pubsub-09.txt>.

[I-D.ietf-core-groupcomm-bis]
Dijk, E., Wang, C., and M. Tiloca, "Group Communication for the Constrained Application Protocol (CoAP)", Work in Progress, Internet-Draft, draft-ietf-core-groupcomm-bis-04, 12 July 2021, <https://www.ietf.org/archive/id/draft-ietf-core-groupcomm-bis-04.txt>.

[I-D.ietf-core-observe-multicast-notifications]
Tiloca, M., Hoeglund, R., Amsuess, C., and F. Palombini, "Observe Notifications as CoAP Multicast Responses", Work in Progress, Internet-Draft, draft-ietf-core-observe-

multicast-notifications-01, 12 July 2021,
<https://www.ietf.org/archive/id/draft-ietf-core-observe-multicast-notifications-01.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-04, 12 July 2021,
<https://www.ietf.org/archive/id/draft-tiloca-core-groupcomm-proxy-04.txt>.

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

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

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

Authors' Addresses

Marco Tiloca
RISE AB
Isafjordsgatan 22
SE-16440 Kista
Sweden

Email: marco.tiloca@ri.se

Rikard Hoeglund
RISE AB
Isafjordsgatan 22
SE-16440 Kista
Sweden

Email: rikard.hoglund@ri.se