LAKE Working Group
Internet-Draft
Intended status: Informational
Expires: 8 January 2026

M. Tiloca
RISE AB

7 July 2025

Implementation Considerations for Ephemeral Diffie-Hellman Over COSE (EDHOC)

draft-ietf-lake-edhoc-impl-cons-04

Abstract

This document provides considerations for guiding the implementation of the authenticated key exchange protocol Ephemeral Diffie-Hellman Over COSE (EDHOC).

Discussion Venues

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

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

Source for this draft and an issue tracker can be found at https://github.com/lake-wg/edhoc-impl-cons.

Status of This Memo

Table of Contents

1. Introduction

Ephemeral Diffie-Hellman Over COSE (EDHOC) [RFC9528] is a lightweight authenticated key exchange protocol, especially intended for use in constrained scenarios.

During the development of EDHOC, a number of side topics were raised and discussed, as emerging from reviews of the protocol latest design and from implementation activities. These topics were identified as strongly pertaining to the implementation of EDHOC rather than to the protocol in itself. Hence, they are not discussed in [RFC9528], which rightly focuses on specifying the actual protocol.

At the same time, implementors of an application using the EDHOC protocol or of an "EDHOC library" enabling its use cannot simply ignore such topics and will have to take them into account throughout their implementation work.

In order to prevent multiple, independent re-discoveries and assessments of those topics, as well as to facilitate and guide implementation activities, this document collects such topics and discusses them through considerations about the implementation of EDHOC. At a high-level, the topics in question are summarized below.

* Handling of completed EDHOC sessions when they become invalid and of application keys derived from an EDHOC session when those become invalid. This topic is discussed in Section 2.

* Enforcement of different trust policies, with respect to learning new authentication credentials during an execution of EDHOC. This topic is discussed in Section 3.

* Branched-off side processing of incoming EDHOC messages, with particular reference to: i) fetching and validation of authentication credentials; and ii) processing of External Authorization Data (EAD) items, which in turn might play a role in the fetching and validation of authentication credentials. This topic is discussed in Section 4.

* Effectively using EDHOC over the Constrained Application Protocol (CoAP) [RFC7252] in combination with Block-wise transfers for CoAP [RFC7959], possibly together with the optimized EDHOC execution workflow defined in [RFC9668]. This topic is discussed in Section 5.

1.1. Terminology

The reader is expected to be familiar with terms and concepts related to the EDHOC protocol [RFC9528], (CoAP) [RFC7252], and Block-wise transfers for CoAP [RFC7959].

2. Handling of Invalid EDHOC Sessions and Application Keys

This section considers the most common situation where, given a certain peer, only the application at that peer has visibility and control of both:

* The EDHOC sessions at that peer; and

* The application keys for that application at that peer, including the knowledge of whether they have been derived from an EDHOC session, i.e., by means of the EDHOC_Exporter interface after the successful completion of an execution of EDHOC (see Section 4.2 of [RFC9528]).

Building on the above, the following expands on three relevant cases concerning the handling of EDHOC sessions and application keys, in the event that any of those becomes invalid.

To provide more concrete guidance, the following also considers the case where "applications keys" stands for the keying material and parameters that compose a Security Context for the security protocol Object Security for Constrained RESTful Environments (OSCORE) [RFC8613], i.e., when specifically those application keys are derived from an EDHOC session (see Appendix A.1 of [RFC9528]).

Nevertheless, the same considerations are applicable if EDHOC is used to derive other application keys, e.g., when used to key different security protocols than OSCORE or to provide the application with secure values that are bound to an EDHOC session.

2.1. EDHOC Sessions Become Invalid

The application at a peer P may have learned that a completed EDHOC session S has to be invalidated. When S is marked as invalid, the application at P purges S and deletes each set of application keys (e.g., the OSCORE Security Context) that was generated from S.

Then, the application runs a new execution of the EDHOC protocol with the other peer. Upon successfully completing the EDHOC execution, the two peers derive and install a new set of application keys from this latest EDHOC session.

The flowchart in Figure 1 shows the handling of an EDHOC session that has become invalid.

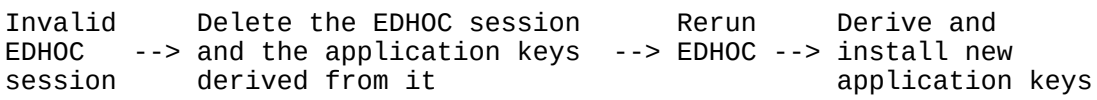


Figure 1: Handling of an EDHOC Session that Has Become Invalid.

An EDHOC session may have become invalid, for example, because an authentication credential CRED_X may have expired, or because the peer P may have learned from a trusted source that CRED_X has been revoked. This effectively invalidates CRED_X, and therefore also invalidates any EDHOC session where CRED_X was used as authentication credential associated with either peer in the session (i.e., P itself or the other peer). In such a case, the application at P has to additionally delete CRED_X and any stored, corresponding credential identifier.

2.2. Application Keys Become Invalid

The application at a peer P may have learned that a set of application keys is not safe to use anymore. When such a set is specifically an OSCORE Security Context, the application may have learned that from the used OSCORE library or from an OSCORE layer that takes part to the communication stack.

A current set SET of application keys shared with another peer can become unsafe to use, for example, due to the following reasons:

* SET has reached a pre-determined expiration time;

* SET has been established to use for a now elapsed amount of time, according to enforced application policies; or

* Some elements of SET have been used enough times to approach cryptographic limits that should not be passed, e.g., according to the properties of the security algorithms specifically used. With particular reference to an OSCORE Security Context, such limits are discussed in [I-D.ietf-core-oscore-key-limits].

When this happens, the application at the peer P proceeds as follows.

1. If the following conditions both hold, then the application moves to Step 2. Otherwise, it moves to Step 3.

   * Let us define S as the EDHOC session from which the peer P has derived SET or the eldest SET's ancestor set of application keys. Then, since the completion of S with the other peer, the application at P has received from the other peer at least one message protected with any set of application keys derived from S. That is, P has persisted S (see Section 5.4.2 of [RFC9528]).

   * The peer P supports a key update protocol, as an alternative to performing a new execution of EDHOC with the other peer. When SET is an OSCORE Security Context, the key update protocol supported by the peer P can be KUDOS [I-D.ietf-core-oscore-key-update].

2. The application at P runs the key update protocol mentioned at Step 1 with the other peer, in order to update SET. When SET is an OSCORE Security Context, the application at P can run the key update protocol KUDOS with the other peer.

   If the key update protocol terminates successfully, the updated application keys are installed and no further actions are taken. Otherwise, the application at P moves to Step 3.

3. The application at the peer P performs the following actions:

   * It deletes SET.

   * It deletes the EDHOC session from which SET was generated, or from which the eldest SET's ancestor set of application keys was generated before any key update occurred (e.g., by means of the EDHOC_KeyUpdate interface defined in Appendix H of [RFC9528] or other key update methods).

   * It runs a new execution of the EDHOC protocol with the other peer. Upon successfully completing the EDHOC execution, the two peers derive and install a new set of application keys from this latest EDHOC session.

The flowchart in Figure 2 shows the handling of a set of application keys that has become invalid. In particular, it assumes such a set to be an OSCORE Security Context and the key update protocol to be KUDOS.

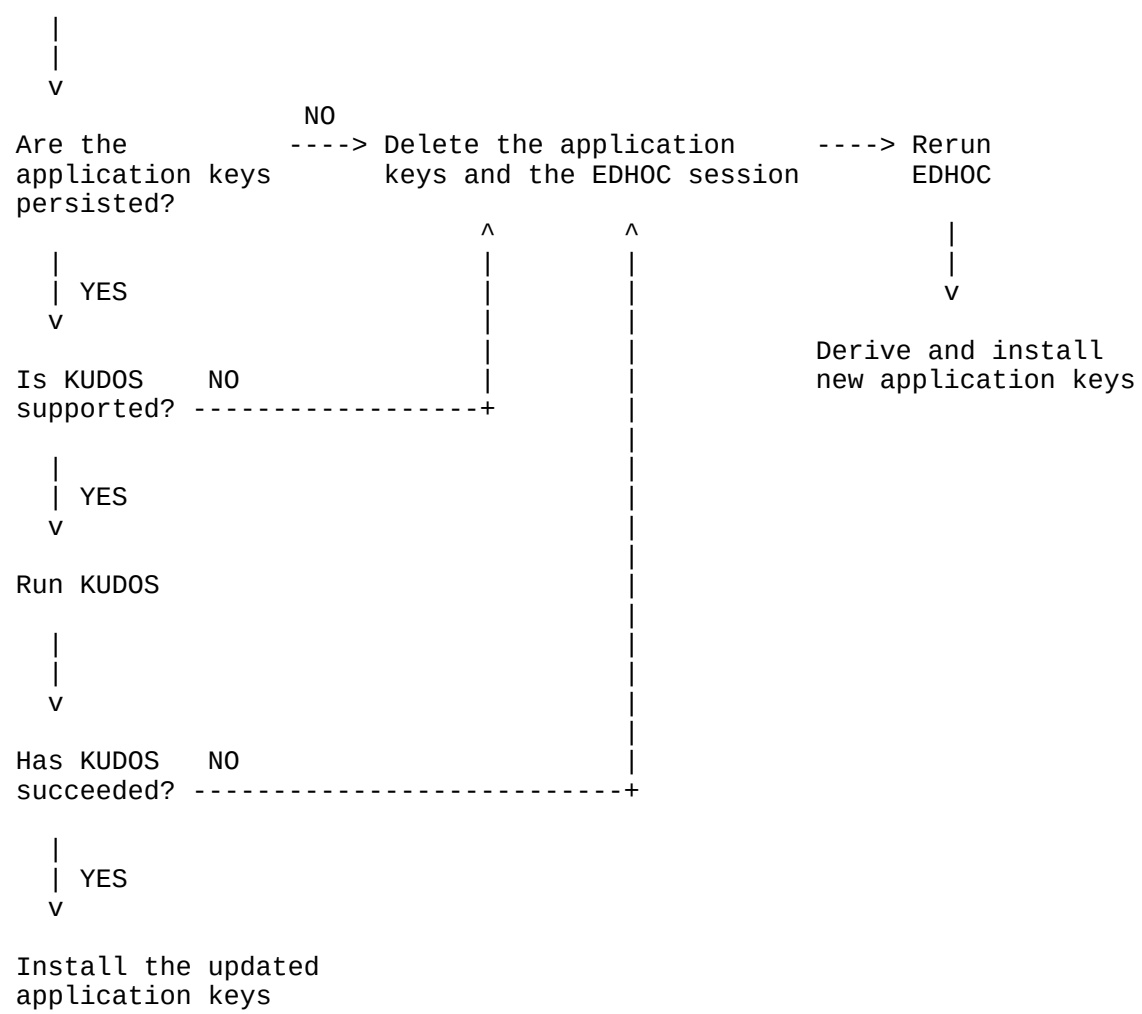


Figure 2: Handling of a Set of Application Keys that Has Become Invalid.

2.3. Application Keys or Bound Access Rights Become Invalid

The following considers two peers that use the ACE framework for authentication and authorization in constrained environments [RFC9200] and specifically the EDHOC and OSCORE profile of ACE defined in [I-D.ietf-ace-edhoc-oscore-profile].

When doing so, one of the two peers acts as ACE resource server (RS) hosting protected resources. The other peer acts as ACE client and requests an access token from an ACE authorization server (AS) that is in trust relationship with the RS. The access token specifies access rights for accessing protected resources at the RS as well as the public authentication credential of the client, namely AUTH_CRED_C.

After that, C uploads the access token to the RS, by means of an EAD item included in an EDHOC message during the EDHOC execution (see below). Alternatively, the AS can upload the access token to the RS on behalf of the client, as per the alternative Short Distribution Chain (SDC) workflow defined in [I-D.ietf-ace-workflow-and-params].

Consistent with the EDHOC and OSCORE profile of ACE, the two peers run EDHOC in order to specifically derive an OSCORE Security Context as their shared set of application keys (see Appendix A.1 of [RFC9528]). At the RS, the access token is bound to the successfully completed EDHOC session and to the established OSCORE Security Context.

After that, the peer acting as ACE client can access the protected resources hosted at the other peer acting as RS, according to the access rights specified in the access token. The communications between the two peers are protected by means of the established OSCORE Security Context.

Later on, the application at one of the two peers P may have learned that the established OSCORE Security Context CTX is not safe to use anymore, e.g., from the used OSCORE library or from an OSCORE layer that takes part to the communication stack. The reasons that make CTX not safe to use anymore are the same ones discussed in Section 2.2 when considering a set of application keys in general, plus the event where the access token bound to CTX becomes invalid (e.g., it has expired or it has been revoked).

When this happens, the application at the peer P proceeds as follows.

1. If the following conditions both hold, then the application moves to Step 2. Otherwise, it moves to Step 3:

* The access token is still believed to be valid. That is, it has not expired yet and the peer P is not aware that it has been revoked.

* Let us define S as the EDHOC session from which the peer P has derived CTX or the eldest CTX's ancestor OSCORE Security Context. Then, since the completion of S with the other peer, the application at P has received from the other peer at least one message protected with any set of application keys derived from S. That is, P has persisted S (see Section 5.4.2 of [RFC9528]).

2. If the peer P supports the key update protocol KUDOS [I-D.ietf-core-oscore-key-update], then P runs KUDOS with the other peer, in order to update CTX. If the execution of KUDOS terminates successfully, the updated OSCORE Security Context is installed and no further actions are taken.

   If the execution of KUDOS does not terminate successfully or if the peer P does not support KUDOS altogether, then the application at P moves to Step 3.

3. The application at the peer P performs the following actions.

   * If the access token is not believed to be valid anymore, the peer P deletes all the EDHOC sessions associated with the access token as well as the OSCORE Security Context derived from each of those sessions.

     Note that, when specifically considering the EDHOC and OSCORE profile of ACE, an access token is associated with at most one EDHOC session (see Section 4.2 of [I-D.ietf-ace-edhoc-oscore-profile]).

     If the peer P acted as ACE client, then P obtains from the ACE AS a new access token, which is uploaded to the other peer acting as ACE RS.

     Finally, the application at P moves to Step 4.

   * If the access token is valid while the OSCORE Security Context CTX is not, then the peer P deletes CTX.

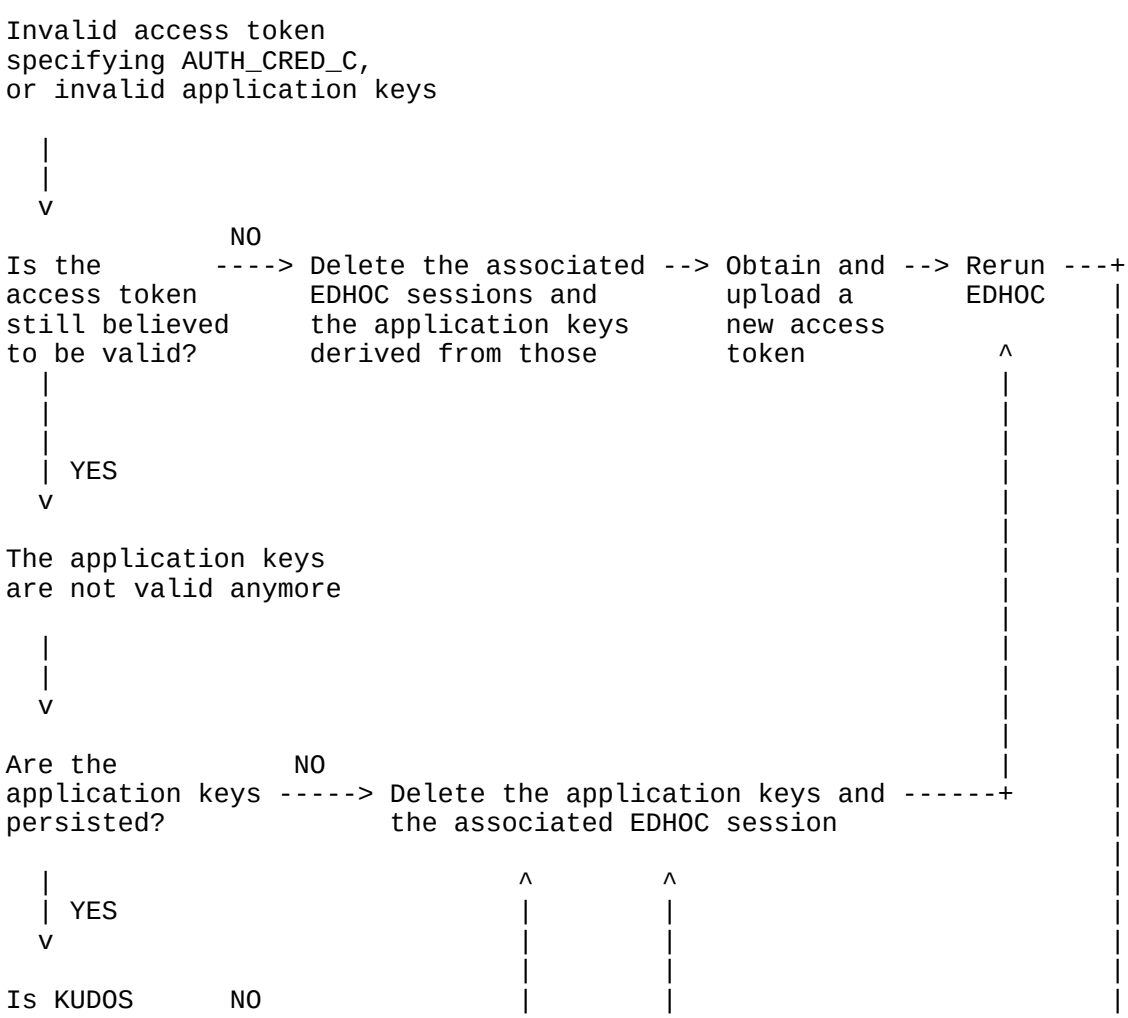
     After that, the peer P deletes the EDHOC session from which CTX was generated, or from which the eldest CTX's ancestor OSCORE Security Context was generated before any key update occurred (e.g., by means of KUDOS or other key update methods).

Finally, the application at P moves to Step 4.

4. The peer P runs a new execution of the EDHOC protocol with the other peer. Upon successfully completing the EDHOC execution, the two peers derive and install a new OSCORE Security Context from this latest EDHOC session.

   At the RS, the access token is bound to this latest EDHOC session and the newly established OSCORE Security Context.

The flowchart in Figure 3 shows the handling of an access token or of a set of application keys that have become invalid, when using the EDHOC and OSCORE profile of the ACE framework.



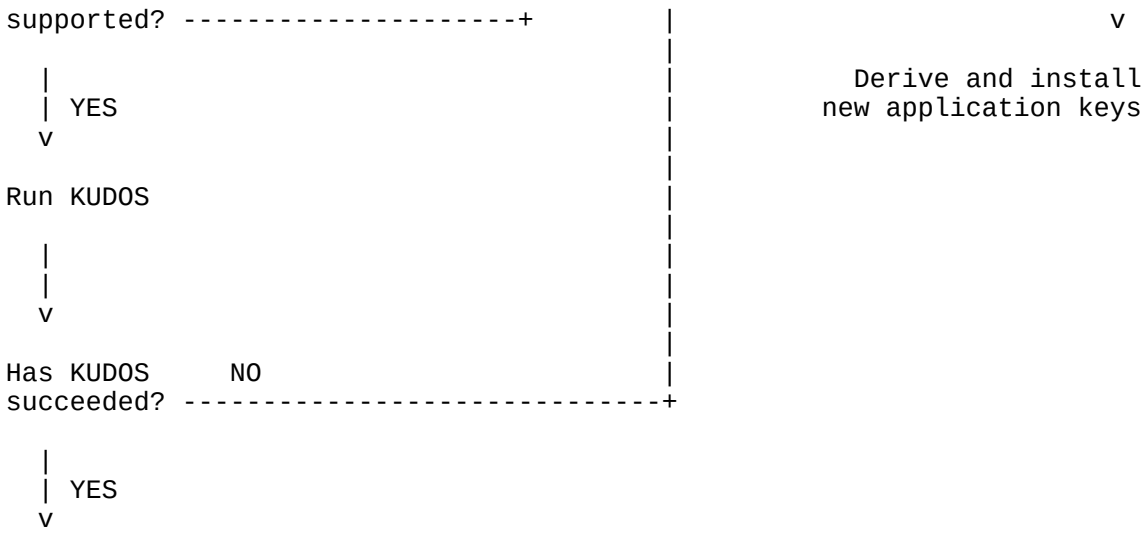


Figure 3: Handling of an Access Token or of a Set of Application Keys that Have Become Invalid.

3. Trust Policies for Learning New Authentication Credentials

A peer P relies on authentication credentials associated with other peers, in order to authenticate those peers when running EDHOC with them.

There are different ways for P to acquire an authentication credential CRED associated with another peer. For example, CRED can be supplied to P out-of-band by a trusted provider.

Alternatively, CRED can be specified by the other peer during the EDHOC execution with P. This can rely on EDHOC message_2 or message_3, whose respective ID_CRED_R and ID_CRED_I field can specify CRED by value, or instead a URI or other external reference where CRED can be retrieved from (see Section 3.5.3 of [RFC9528]).

Also during the EDHOC execution, an External Authorization Data (EAD) field might include an EAD item that specifies CRED by value or reference. This is the case, e.g., for the EAD items defined by the EDHOC and OSCORE profile of the ACE framework [I-D.ietf-ace-edhoc-oscore-profile]. In particular, they can be used for transporting (a reference to) an access token, which in turn specifies by value or by reference the public authentication credential associated with the EDHOC peer acting as ACE client.

When obtaining a new credential CRED, the peer P has to validate it before storing it. The validation steps to perform depend on the specific type of CRED (e.g., a public key certificate [RFC5280][I-D.ietf-cose-cbor-encoded-cert]) and can rely on (the authentication credential associated with) a trusted third party acting as a trust anchor.

Upon retrieving a new CRED through the processing of a received EDHOC message and following the successful validation of CRED, the peer P stores CRED only if it assesses CRED to also be trusted, while it must not store CRED otherwise.

An exception applies for the two unauthenticated operations described in Appendix D.5 of [RFC9528], where a trust relationship with an unknown or not-yet-trusted endpoint is established later. In such a case, CRED is verified out-of-band at a later stage, or an EDHOC session key is bound to an identity out-of-band at a later stage.

When processing a received EDHOC message M that specifies an authentication credential CRED, the peer P can enforce one of the trust policies LEARNING and NO-LEARNING specified in Section 3.1 and Section 3.2, in order to determine whether to trust CRED.

Irrespective of the adopted trust policy, P actually uses CRED only if it is determined to be fine to use in the context of the ongoing EDHOC session, also depending on the specific identity of the other peer (see Sections 3.5 and D.2 of [RFC9528]). If this is not the case, P aborts the EDHOC session with the other peer.

If P stores CRED, then P will consider CRED as valid and trusted until CRED possibly becomes invalid, e.g., because it expires or because P gains knowledge that it has been revoked.

When storing CRED, the peer P should generate the authentication credential identifier(s) corresponding to CRED and store them as associated with CRED. For example, if CRED is a public key certificate, an identifier of CRED can be the hash of the certificate. In general, P should generate and associate with CRED one corresponding identifier for each type of authentication credential identifier that P supports and that is compatible with CRED.

In future executions of EDHOC with the other peer associated with CRED, this allows such other peer to specify CRED by reference, e.g., by indicating its credential identifier as ID_CRED_R/ID_CRED_I in the EDHOC message_2 or message_3 addressed to the peer P. In turn, this allows P to retrieve CRED from its local storage.

3.1. Trust Policy LEARNING

When enforcing the LEARNING policy, the peer P trusts CRED even if P is not already storing CRED at message reception time.

That is, upon receiving M, the peer P performs the following steps.

1. P retrieves CRED, as specified by reference or by value in the ID_CRED_I/ID_CRED_R field of M or in the value of an EAD item of M.

2. P checks whether CRED is already being stored and if it is still valid. In such a case, P trusts CRED and can continue the EDHOC execution. Otherwise, P moves to Step 3.

3. P attempts to validate CRED. If the validation process is not successful, P aborts the EDHOC session with the other peer. Otherwise, P trusts and stores CRED, and can continue the EDHOC execution.

3.2. Trust Policy NO-LEARNING

When enforcing the NO-LEARNING policy, the peer P trusts CRED only if P is already storing CRED at message reception time, unless in cases where situation-specific exceptions apply and are deliberately enforced (see below).

That is, upon receiving M, the peer P continues the execution of EDHOC only if both the following conditions hold:

* P currently stores CRED, as specified by reference or by value in the ID_CRED_I/ID_CRED_R field of M or in the value of an EAD item of M; and

* CRED is still valid, i.e., P believes CRED to not be expired or revoked.

Exceptions may apply and be actually enforced in cases where, during an EDHOC execution, P obtains additional information that allows it to trust and successfully validate CRED, even though CRED is not already stored upon receiving M.

Such exceptions typically rely on a trusted party that vouches for CRED, e.g., in the cases discussed in Section 3.3. From the point of view of the peer P, the trusted party might have been involved in the background, so that the vouching information about CRED is conveyed within an EDHOC message received during the EDHOC session (e.g., transported by an EAD item). Alternatively, P might directly interact with the trusted party for retrieving the vouching information about CRED, e.g., after having received M and before continuing the EDHOC execution.

If the peer P admits such an exception and actually enforces it on an authentication credential CRED, then P effectively handles CRED according to the trust policy "LEARNING" specified in Section 3.1. When doing so, P still attempts to validate CRED and aborts the EDHOC session in case of failed validation.

3.3. Enforcement of Trust Policies in Specific Scenarios

The following subsections discuss how an EDHOC peer enforces the trust policies LEARNING and NO-LEARNING in specific scenarios.

3.3.1. In the EDHOC and OSCORE Profile of ACE

As discussed in Section 2.3, two EDHOC peers can be using the ACE framework [RFC9200] and specifically the EDHOC and OSCORE profile of ACE defined in [I-D.ietf-ace-edhoc-oscore-profile].

In this case, one of the two EDHOC peers, namely PEER_RS, acts as ACE resource server (RS). Instead, the other EDHOC peer, namely PEER_C, acts as ACE client and obtains from an ACE authorization server (AS) an access token for accessing protected resources at PEER_RS. The AS and PEER_RS are in a trust relationship.

Together with other information, the access token specifies (by value or by reference) the public authentication credential AUTH_CRED_C associated with PEER_C that PEER_C is going to use when running EDHOC with PEER_RS. Note that AUTH_CRED_C will be used as either CRED_I or CRED_R in EDHOC, depending on whether the two peers use the EDHOC forward message flow (i.e., PEER_C is the EDHOC Initiator) or the EDHOC reverse message flow (i.e., PEER_C is the EDHOC Responder), respectively (see Appendix A.2 of [RFC9528]).

When the AS issues the first access token that specifies AUTH_CRED_C and is intended to be uploaded to PEER_RS, it is expected that the access token specifies AUTH_CRED_C by value and that PEER_RS is not currently storing AUTH_CRED_C, but instead will obtain it and learn it upon receiving the access token.

Although the AS can upload the access token to PEER_RS on behalf of PEER_C as per the alternative SDC workflow defined in [I-D.ietf-ace-workflow-and-params], the access token is typically uploaded to PEER_RS by PEER_C through a dedicated EAD item, when running EDHOC with PEER_RS. Consequently, PEER_RS has to learn AUTH_CRED_C as a new authentication credential during an EDHOC session with PEER_C.

At least for its EDHOC resource used for exchanging the EDHOC messages of the EDHOC session in question, this requires PEER_RS to:

* Enforce the trust policy "LEARNING"; or

* If enforcing the trust policy "NO-LEARNING", additionally enforce an overriding exception when an incoming EDHOC message includes an EAD item conveying (a reference to) a valid access token issued by a trusted AS.

  That is, through a successful verification of the access token, PEER_RS is able to trust AUTH_CRED_C (if found valid), even though it did not already store AUTH_CRED_C upon receiving the EDHOC message with the EAD item.

3.3.2. In the Lightweight Authorization using EDHOC (ELA) Procedure

When the execution of EDHOC embeds the ELA procedure defined in [I-D.ietf-lake-authz], the EDHOC peer U receives an EDHOC message_2 (message_3) where ID_CRED_R (ID_CRED_I) specifies by value the authentication credential CRED associated with the other peer V.

In the same EDHOC message, an EAD item specifies a voucher issued by a trusted enrollment server W. The voucher is an assertion to U that W has authorized V and has endorsed CRED.

Through a successful verification of the voucher, U is able to trust CRED (if found valid), even though it did not already store CRED upon receiving the EDHOC message.

Therefore, if U enforces the trust policy "NO-LEARNING", it can additionally enforce an overriding exception when an incoming EDHOC message includes an EAD item conveying a valid voucher issued by a trusted enrollment server.

4. Side Processing of Incoming EDHOC Messages

This section describes an approach that EDHOC peers can use upon receiving EDHOC messages, in order to fetch/validate authentication credentials and to process External Authorization Data (EAD) items.

As per Section 9.1 of [RFC9528], the EDHOC protocol provides a transport mechanism for conveying EAD items, but specifications defining those items have to set the ground for "agreeing on the surrounding context and the meaning of the information passed to and from the application".

The approach described in this section aims to help implementors navigate the surrounding context mentioned above, irrespective of the specific EAD items conveyed in the EDHOC messages. In particular, the described approach takes into account the following points:

* The fetching and validation of the authentication credential associated with the other peer relies on ID_CRED_I in EDHOC message_2, or on ID_CRED_R in EDHOC message_3, or on the value of an EAD item. When this occurs upon receiving EDHOC message_2 or message_3, the decryption of the EDHOC message has to be completed first.

  The validation of the authentication credential might depend on using the value of an EAD item, which in turn has to be validated first. For instance, an EAD item within the EAD_2 field of EDHOC message_2 may contain a voucher to be used for validating the authentication credential associated with the Responder (see [I-D.ietf-lake-authz]).

* Some EAD items may be processed only after having successfully verified the EDHOC message, i.e., after a successful verification of the Signature_or_MAC field in EDHOC message_2 or message_3.

For instance, an EAD item within the EAD_3 field of EDHOC message_3 may contain a Certificate Signing Request (CSR) [RFC2986]. Hence, such an EAD item can be processed only once the recipient peer has attained proof that the other peer possesses its private key.

In order to conveniently handle such processing, the application can prepare in advance a "side-processor object" (SPO), which takes care of the operations above during the EDHOC execution.

In particular, the application provides EDHOC with the SPO before starting an EDHOC execution, during which EDHOC will temporarily transfer control to the SPO at the right point in time, in order to perform the required side-processing of an incoming EDHOC message.

Furthermore, the application has to instruct the SPO about how to prepare any EAD item such that: it has to be included in an outgoing EDHOC message; and it is independent of the processing of other EAD items included in incoming EDHOC messages. This includes, for instance, the preparation of padding EAD items.

At the right point in time during the processing of an incoming EDHOC message M at the peer P, EDHOC invokes the SPO and provides it with the following input:

* When M is EDHOC message_2 or message_3, an indication of whether this invocation is happening before or after the message verification (i.e., before or after having verified the Signature_or_MAC field).

* The full set of information related to the current EDHOC session. This especially includes the selected cipher suite and the ephemeral Diffie-Hellman public keys G_X and G_Y that the two peers have exchanged in the EDHOC session.

* The authentication credentials that the peer P stores as associated with other peers.

* All the decrypted information elements retrieved from M.

* The EAD items included in M.

  - Note that EDHOC might do some preliminary work on M before invoking the SPO, in order to provide the SPO only with actually relevant EAD items. This requires the application to additionally provide EDHOC with the ead_labels of the EAD items that the peer P recognizes (see Section 3.8 of [RFC9528]).

With such information available, EDHOC can early abort the current session in case M includes any EAD item which is both critical and not recognized by the peer P.

If no such EAD items are found, EDHOC can remove any padding EAD item (see Section 3.8.1 of [RFC9528]), as well as any EAD item which is neither critical nor recognized (since the SPO is going to ignore it anyway). This results in EDHOC providing the SPO only with EAD items that will be recognized and that require actual processing.

- Note that, after having processed the EAD items, the SPO might actually need to store them throughout the whole EDHOC execution, e.g., in order to refer to them also when processing later EDHOC messages in the current EDHOC session.

The SPO performs the following tasks on an incoming EDHOC message M:

* The SPO fetches and/or validates the authentication credential CRED associated with the other peer, based on a dedicated EAD item of M or on the ID_CRED field of M (for EDHOC message_2 or message_3).

* The SPO processes the EAD items conveyed in the EAD field of M.

* The SPO stores the results of the performed operations and makes such results available to the application.

* When the SPO has completed its side processing and transfers control back to EDHOC, the SPO provides EDHOC with the produced EAD items to include in the EAD field of the next outgoing EDHOC message. The production of such EAD items can be triggered, e.g., by:

  - The consumption of EAD items included in M.

  - The execution of instructions that the SPO has received from the application, concerning EAD items to produce irrespective of other EAD items included in M.

In the following, Section 4.1 to Section 4.3 describe more in detail the actions performed by the SPO on the different incoming EDHOC messages. Then, Section 4.4 describes further special handling of incoming EDHOC messages, as to consistency checks concerning authentication credentials in particular situations.

After completing the EDHOC execution, control is transferred back to the application. In particular, the application is provided with the overall outcome of the EDHOC execution (i.e., successful completion or failure), together with possible specific results produced by the SPO throughout the EDHOC execution (e.g., due to the processing of EAD items).

After that, the application might need to perform follow-up actions, depending on the outcome of the EDHOC execution. For example, the SPO might have preliminarily filled application-level data structures, as a result of processing EAD items. In case of successful EDHOC execution, the application might need to finalize such data structures. Instead, in case of unsuccessful EDHOC execution, the application might need to clean-up or amend such data structures, or even roll back what the SPO did, unless the SPO already performed such actions before control was transferred back to the application.

4.1. EDHOC message_1

During the processing of an incoming EDHOC message_1, EDHOC invokes the SPO only once, after the Responder peer has successfully decoded the message and accepted the selected cipher suite.

If the EAD_1 field is present, the SPO processes the EAD items included therein.

Once all such EAD items have been processed, the SPO transfers control back to EDHOC. When doing so, the SPO also provides EDHOC with any produced EAD items to include in the EAD field of the next outgoing EDHOC message.

Then, EDHOC resumes its execution and advances its protocol state.

Future extensions of EDHOC or external security applications integrated into EDHOC might require a processing of EDHOC message_1 that is more advanced than the currently expected one. In particular, an EAD item conveyed in EDHOC message_1 might specify the authentication credential CRED associated with the Initiator (by value or by reference), as wrapped in a cryptographically protected "envelope". In such a case, the processing of an incoming EDHOC message_1 shares similarities with that of an incoming EDHOC message_2 or message_3 (see Section 4.3), as it is further elaborated in Appendix A.

4.2. EDHOC message_4

During the processing of an incoming EDHOC message_4, EDHOC invokes the SPO only once, after the Initiator peer has successfully decrypted the message.

If the EAD_4 field is present, the SPO processes the EAD items included therein.

Once all such EAD items have been processed, the SPO transfers control back to EDHOC, which resumes its execution and advances its protocol state.

4.3. EDHOC message_2 and message_3

The following refers to "message_X" as an incoming EDHOC message_2 or message_3, and to "message verification" as the verification of Signature_or_MAC_X in message_X.

During the processing of a message_X, EDHOC invokes the SPO two times:

* Right after message_X has been decrypted and before its verification starts. Following this invocation, the SPO performs the actions described in Section 4.3.1.

* Right after message_X has been successfully verified. Following this invocation, the SPO performs the actions described in Section 4.3.4.

The flowcharts in Section 4.3.5 show the high-level interaction between the core EDHOC processing and the SPO, as well as the different steps taken for processing an incoming message_X.

4.3.1. Pre-Verification Side Processing

The pre-verification side processing occurs in two sequential phases, namely PHASE_1 (see Section 4.3.2) and PHASE_2 (see Section 4.3.3).

4.3.2. PHASE_1

During PHASE_1, the SPO at the recipient peer P determines CRED, i.e., the authentication credential associated with the other peer to be used in the ongoing EDHOC session. In particular, the SPO performs the following steps.

1. The SPO determines CRED based on ID_CRED_X or on an EAD item in message_X.

Those may specify CRED by value or by reference, including a URI or other external reference where CRED can be retrieved from.

If CRED is already installed, the SPO moves to Step 2. Otherwise, the SPO moves to Step 3.

2. The SPO determines if the stored CRED is currently valid, e.g., by verifying that CRED has not expired and has not been revoked.

   Performing such a validation may require the SPO to first process an EAD item included in message_X. For example, it can be an EAD item in EDHOC message_2 that confirms or revokes the validity of CRED_R specified by ID_CRED_R, as the result of an OCSP process [RFC6960].

   In case CRED is determined to be valid, the SPO moves to Step 9. Otherwise, the SPO moves to Step 11.

3. The SPO attempts to retrieve CRED via ID_CRED_X or an EAD item considered at Step 1. Then, the SPO moves to Step 4.

4. If the retrieval of CRED has succeeded, the SPO moves to Step 5. Otherwise, the SPO moves to Step 11.

5. If the enforced trust policy for new authentication credentials is "NO-LEARNING" and P does not admit any exceptions that are acceptable to enforce for message_X (see Section 3.2), the SPO moves to Step 11. Otherwise, the SPO moves to Step 6.

6. If this step has been reached, the peer P is not already storing the retrieved CRED and, at the same time, it enforces either the trust policy "LEARNING" or the trust policy "NO-LEARNING" while also enforcing an exception acceptable for message_X (see Section 3.2).

   Consistently, the SPO determines if CRED is currently valid, e.g., by verifying that CRED has not expired and has not been revoked. Then, the SPO moves to Step 7.

   Validating CRED may require the SPO to first process an EAD item included in message_X. For example, it can be an EAD item in EDHOC message_2 that: i) specifies a voucher for validating CRED_R as a CWT Claims Set (CCS) [RFC8392] transported by value in ID_CRED_R (see [I-D.ietf-lake-authz]); or instead ii) an OCSP response [RFC6960] for validating CRED_R as a public key certificate transported by value or reference in ID_CRED_R.

7. If CRED has been determined valid, the SPO moves to Step 8. Otherwise, the SPO moves to Step 11.

8. The SPO stores CRED as a valid and trusted authentication credential associated with the other peer, together with corresponding authentication credential identifiers (see Section 3). Then, the SPO moves to Step 9.

9. The SPO checks if CRED is fine to use in the context of the ongoing EDHOC session, also depending on the specific identity of the other peer (see Sections 3.5 and D.2 of [RFC9528]).

   If this is the case, the SPO moves to Step 10. Otherwise, the SPO moves to Step 11.

10. P uses CRED as authentication credential associated with the other peer in the ongoing EDHOC session.

    Then, PHASE_1 ends and the pre-verification side processing moves to the next PHASE_2 (see Section 4.3.3).

11. The SPO has not found a valid authentication credential associated with the other peer that can be used in the ongoing EDHOC session. Therefore, the EDHOC session with the other peer is aborted.

4.3.3. PHASE_2

During PHASE_2, the SPO processes any EAD item included in message_X such that both the following conditions hold:

* The EAD item has _not_ been already processed during PHASE_1.

* The EAD item can be processed before performing the verification of message_X.

Once all such EAD items have been processed, the SPO transfers control back to EDHOC, which either aborts the ongoing EDHOC session or continues the processing of message_X with its corresponding message verification.

4.3.4. Post-Verification Side Processing

During the post-verification side processing, the SPO processes any EAD item included in message_X such that the processing of that EAD item had to wait for completing the successful message verification.

The late processing of such EAD items is typically due to the fact that a pre-requirement has to be fulfilled first.

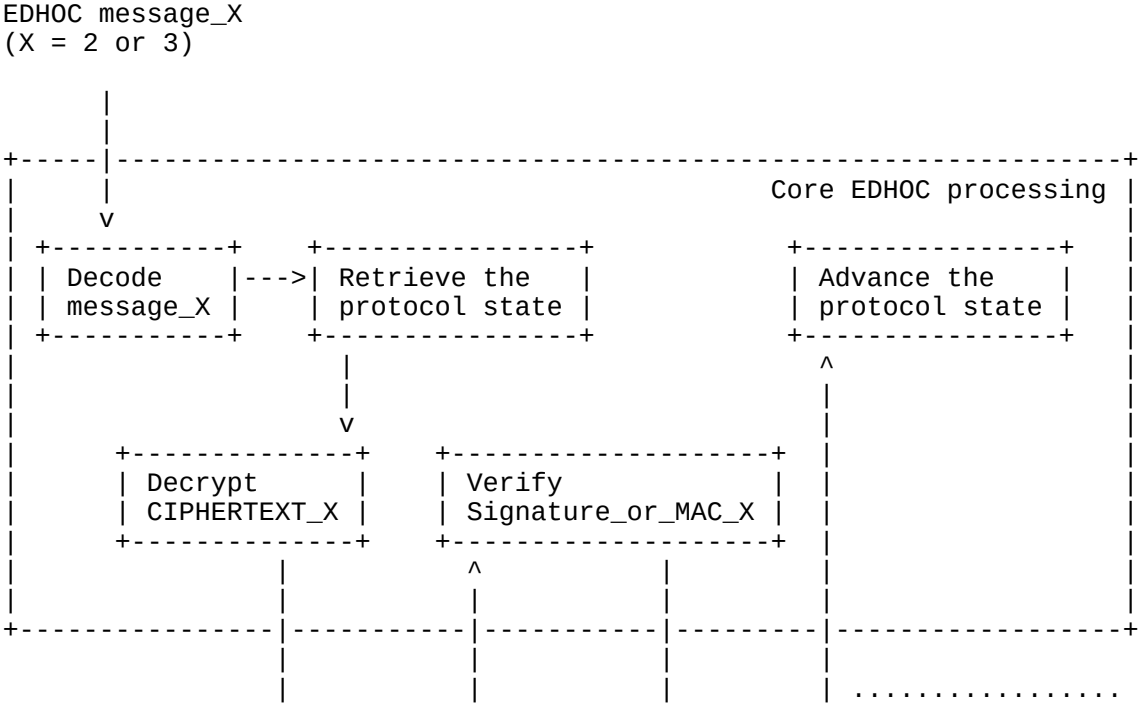
For example, the recipient peer P has to have first verified that the other peer does possess the private key corresponding to the public key specified by CRED, i.e., the authentication credential associated with the other peer that was determined during the pre-verification side processing (see Section 4.3.1). This requirement is fulfilled after a successful verification of message_X.

Once all such EAD items have been processed, the SPO transfers control back to EDHOC. When doing so, the SPO also provides EDHOC with any produced EAD items to include in the EAD field of the next outgoing EDHOC message.

Then, EDHOC resumes its execution and advances its protocol state.

4.3.5. Flowcharts

The flowchart in Figure 4 shows the high-level interaction between the core EDHOC processing and the SPO, with particular reference to an incoming EDHOC message_2 or message_3.



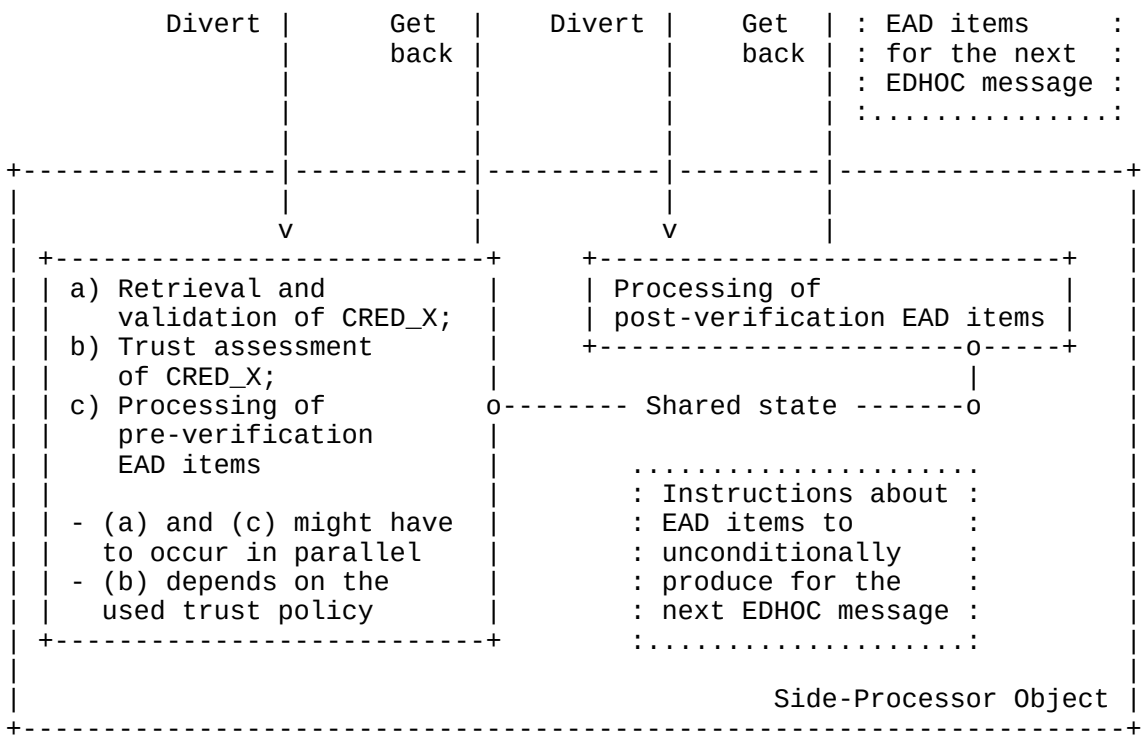
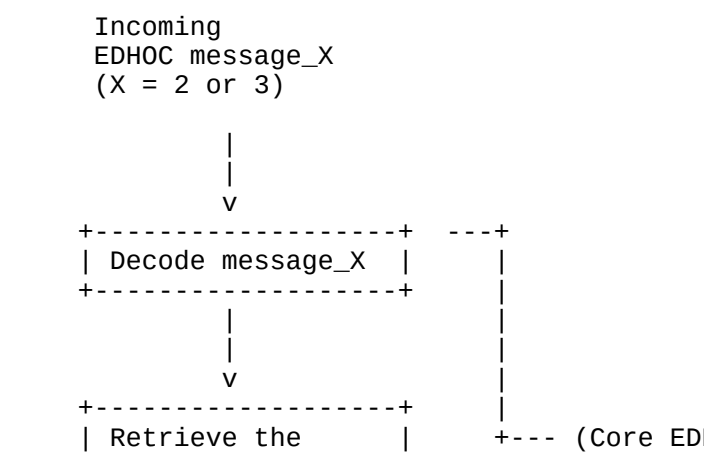
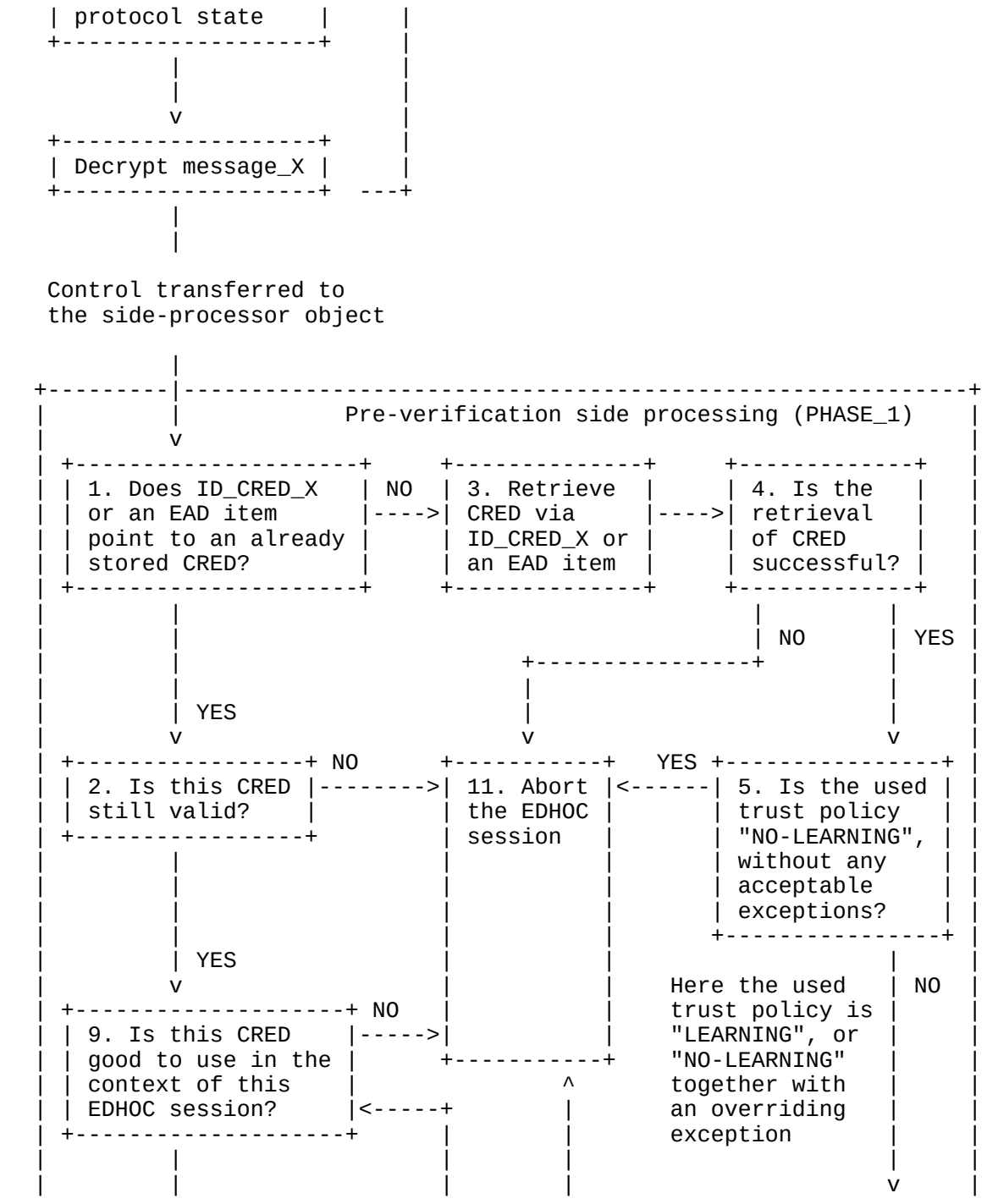
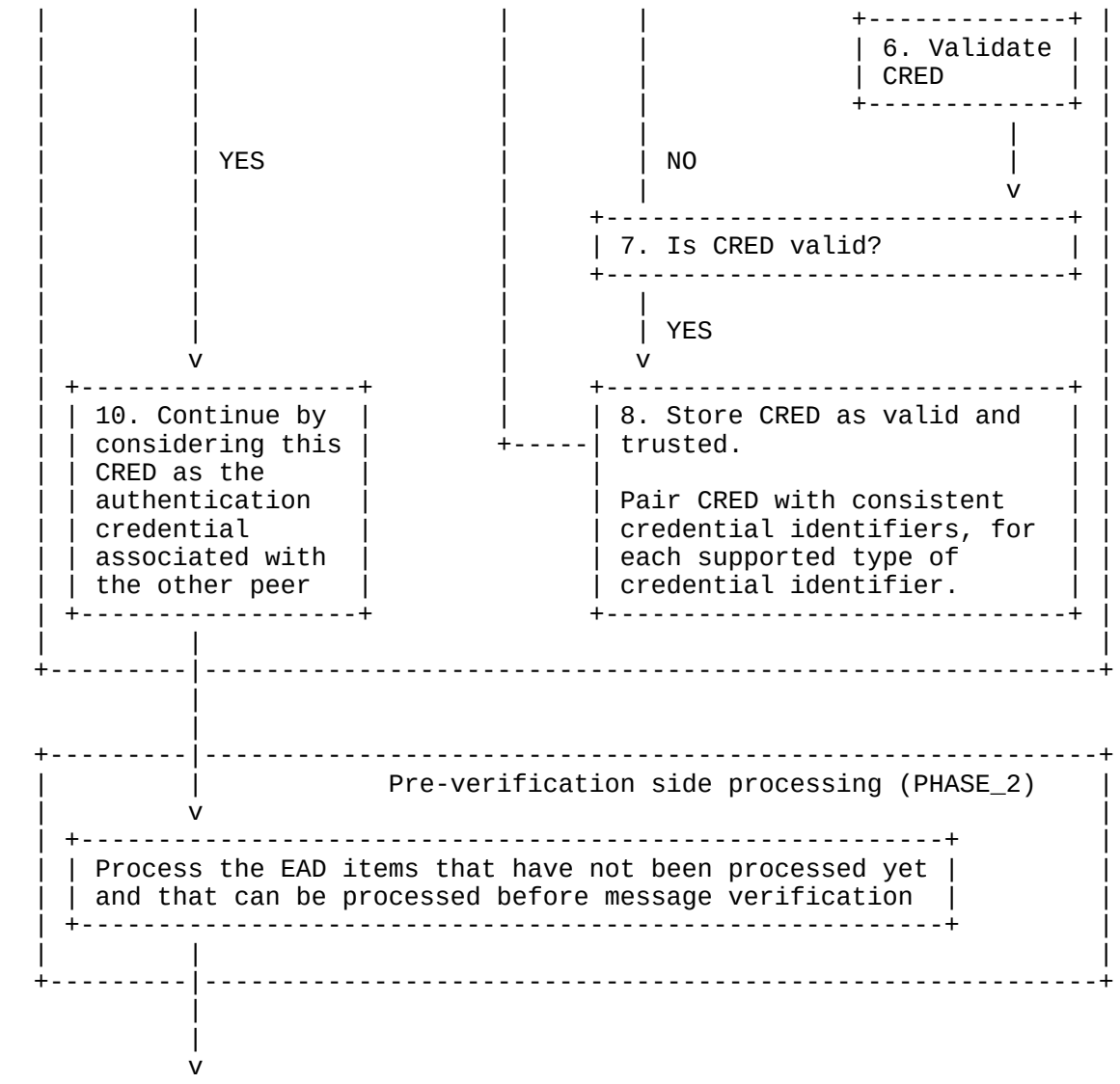


Figure 4: High-Level Interaction Between the Core EDHOC Processing and the Side-Processor Object (SPO), for EDHOC message_2 and message_3.

The flowchart in Figure 5 shows the different steps taken for processing an incoming EDHOC message_2 and message_3.




protocol state
Decrypt message_X
Control transferred to
the side-processor object
Pre-verification side processing (PHASE_1)
1. Does ID_CRED_X or an EAD item point to an already stored CRED?
NO
3. Retrieve CRED via ID_CRED_X or an EAD item
4. Is the retrieval of CRED successful?
NO
YES
YES
2. Is this CRED still valid?
NO
11. Abort the EDHOC session
YES
5. Is the used trust policy "NO-LEARNING", without any acceptable exceptions?
YES
9. Is this CRED good to use in the context of this EDHOC session?
NO
Here the used trust policy is "LEARNING", or "NO-LEARNING" together with an overriding exception
NO


6. Validate CRED
NO
7. Is CRED valid?
YES
8. Store CRED as valid and trusted.
Pair CRED with consistent credential identifiers, for each supported type of credential identifier.
YES
10. Continue by considering this CRED as the authentication credential associated with the other peer
Pre-verification side processing (PHASE_2)
Process the EAD items that have not been processed yet and that can be processed before message verification
Control transferred back to the core EDHOC processing
Verify message_X
(Core EDHOC processing)

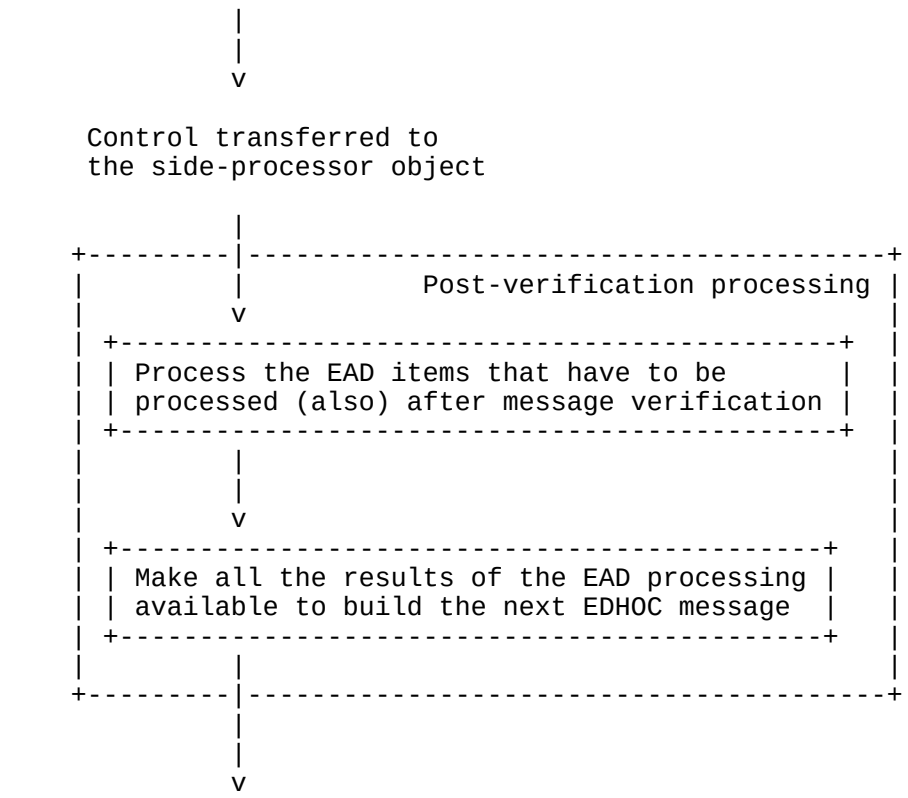


Figure 5: Processing Steps for EDHOC message_2 and message_3.

4.4. Consistency Checks of Authentication Credentials from ID_CRED and EAD Items

Typically, an EDHOC peer specifies its associated authentication credential (by value or by reference) only in the ID_CRED field of EDHOC message_2 (if acting as Responder) or EDHOC message_3 (if acting as Initiator).

In addition to that, there may be cases where an EDHOC peer provides the authentication credential also in an EAD item. In particular, such an EAD item can specify a cryptographically protected "envelope" (by value or by reference), which in turn specifies the authentication credential (by value or by reference).

A case in point is the EDHOC and OSCORE profile of the ACE framework [I-D.ietf-ace-edhoc-oscore-profile], where the envelope in question is an access token issued to the ACE client. In such a case, the ACE client can rely on an EAD item specifying the access token by value, or instead on a different EAD item specifying a session identifier as a reference to such access token. In either case, the access token specifies the authentication credential (by value or by reference) associated with the client.

During an EDHOC session, an EDHOC peer P1 might therefore receive the authentication credential CRED associated with the other EDHOC peer P2 as specified by two items:

* ITEM_A: the ID_CRED field specifying CRED. If P2 acts as Initiator (Responder), then ITEM_A is the ID_CRED_I (ID_CRED_R) field.

* ITEM_B: the envelope specified in an EAD item within an EDHOC message sent by P2.

As part of the process where P1 validates CRED during the EDHOC session, P1 must check that both ITEM_A and ITEM_B specify the same authentication credential, and it must abort the EDHOC session if such a consistency check fails.

The consistency check is successful only if one of the following conditions hold and it fails otherwise:

* If both ITEM_A and ITEM_B specify an authentication credential by value, then both ITEM_A and ITEM_B specify the same value.

* If one among ITEM_A and ITEM_B specifies an authentication credential by value VALUE while the other one specifies an authentication credential by reference REF, then REF is a valid reference for VALUE.

* If ITEM_A specifies an authentication credential by reference REF_A and ITEM_B specifies an authentication credential by reference REF_B, then REF_A or REF_B allows to retrieving the value VALUE of an authentication credential from a local or remote storage, such that both REF_A and REF_B are a valid reference for VALUE.

The peer P1 performs the consistency check above as soon as it has both ITEM_A and ITEM_B available. If P1 acts as Responder, that is the case when processing the incoming EDHOC message_3. If P1 acts as Initiator, that is the case when processing the incoming EDHOC message_2 or message_4, i.e., whichever of the two messages includes ITEM_B in an EAD item of its EAD field.

5. Using EDHOC over CoAP with Block-Wise

Appendix A.2 of [RFC9528] specifies how to transfer EDHOC over CoAP [RFC7252]. In such a case, the EDHOC messages (possibly prepended by an EDHOC connection identifier) are transported in the payload of CoAP requests and responses, according to the EDHOC forward message flow or the EDHOC reverse message flow. Furthermore, Appendix A.1 of [RFC9528] specifies how to derive an OSCORE Security Context [RFC8613] from an EDHOC session.

Building on that, [RFC9668] further details the use of EDHOC with CoAP and OSCORE. In particular, it specifies an optimization approach for the EDHOC forward message flow that combines the EDHOC execution with the first subsequent OSCORE transaction. This is achieved by means of an "EDHOC + OSCORE request", i.e., a single CoAP request message that conveys both EDHOC message_3 of the ongoing EDHOC session and the OSCORE-protected application data, where the latter is protected with the OSCORE Security Context derived from that EDHOC session.

This section provides guidelines and recommendations for CoAP endpoints supporting Block-wise transfers for CoAP [RFC7959] and specifically for CoAP clients supporting the EDHOC + OSCORE request defined in [RFC9668]. The use of Block-wise transfers can be desirable, e.g., for EDHOC messages that include a large ID_CRED_I or ID_CRED_R, or that include a large EAD field.

The following especially considers a CoAP endpoint that may perform only "inner" Block-wise, but not "outer" Block-wise operations (see Section 4.1.3.4 of [RFC8613]). That is, the considered CoAP endpoint does not (further) split an OSCORE-protected message like an intermediary (e.g., a proxy) might do. This is the typical case for CoAP endpoints using OSCORE (see Section 4.1.3.4 of [RFC8613]).

5.1. Notation

The rest of this section refers to the following notation:

* SIZE_BODY: the size in bytes of the data to be transmitted with CoAP. When Block-wise is used, such data is referred to as the "body" to be fragmented into blocks, each of which to be transmitted in one CoAP message.

  With the exception of EDHOC message_3, the considered body can also be an EDHOC message, possibly prepended by an EDHOC connection identifier encoded as per Section 3.3 of [RFC9528].

  When specifically using the EDHOC + OSCORE request, the considered body is the application data to be protected with OSCORE, (whose first block is) to be sent together with EDHOC message_3 as part of the EDHOC + OSCORE request.

* SIZE_EDHOC_M3: the size in bytes of EDHOC message_3, if this is sent as part of the EDHOC + OSCORE request. Otherwise, the size in bytes of EDHOC message_3, plus, if included, the size in bytes of a prepended EDHOC connection identifier encoded as per Section 3.3 of [RFC9528].

* SIZE_MTU: the maximum amount of transmittable bytes before having to use Block-wise. This is, for example, 64 KiB as maximum datagram size when using UDP, or 1280 bytes as the maximum size for an IPv6 MTU.

* SIZE_OH: the size in bytes of the overall overhead due to all the communication layers underlying the application. This takes into account also the overhead introduced by the OSCORE processing.

* LIMIT = (SIZE_MTU - SIZE_OH): the practical maximum size in bytes to be considered by the application before using Block-wise.

* SIZE_BLOCK: the size in bytes of inner blocks.

* ceil(): the ceiling function.

5.2. Pre-requirements for the EDHOC + OSCORE Request

Before sending an EDHOC + OSCORE request, a CoAP client has to perform the following checks. Note that, while the CoAP client is able to fragment the plain application data before any OSCORE processing, it cannot fragment the EDHOC + OSCORE request or the EDHOC message_3 added therein.

* If inner Block-wise is not used, hence SIZE_BODY <= LIMIT, the CoAP client must verify whether all the following conditions hold:

  - COND1: SIZE_EDHOC_M3 <= LIMIT

- COND2: (SIZE_BODY + SIZE_EDHOC_M3) <= LIMIT

* If inner Block-wise is used, the CoAP client must verify whether all the following conditions hold:

  - COND3: SIZE_EDHOC_M3 <= LIMIT

  - COND4: (SIZE_BLOCK + SIZE_EDHOC_M3) <= LIMIT

In either case, if not all the corresponding conditions hold, the CoAP client should not send the EDHOC + OSCORE request. Instead, the CoAP client can continue by switching to the purely sequential, original EDHOC workflow (see Appendix A.2.1 of [RFC9528]). That is, the CoAP client first sends EDHOC message_3 prepended by the EDHOC Connection Identifier C_R encoded as per Section 3.3 of [RFC9528] and then sends the OSCORE-protected CoAP request once the EDHOC execution is completed.

5.3. Effectively Using Block-Wise

In order to avoid further fragmentation at lower layers when sending an EDHOC + OSCORE request, the CoAP client has to use inner Block-wise if _any_ of the following conditions holds:

* COND5: SIZE_BODY > LIMIT

* COND6: (SIZE_BODY + SIZE_EDHOC_M3) > LIMIT

In particular, consistently with Section 5.2, the SIZE_BLOCK used has to be such that the following condition also holds:

* COND7: (SIZE_BLOCK + SIZE_EDHOC_M3) <= LIMIT

Note that the CoAP client might still use Block-wise due to reasons different from exceeding the size indicated by LIMIT.

The following shows the number of round trips for completing both the EDHOC execution and the first OSCORE-protected exchange, under the assumption that the exchange of EDHOC message_1 and EDHOC message_2 does not result in using Block-wise.

If _both_ the conditions COND5 and COND6 hold, the use of Block-wise results in the following number of round trips experienced by the CoAP client.

* If the original EDHOC execution workflow is used (see Appendix A.2.1 of [RFC9528]), the number of round trips RT_ORIG is equal to 1 + ceil(SIZE_EDHOC_M3 / SIZE_BLOCK) + ceil(SIZE_BODY / SIZE_BLOCK).

* If the optimized EDHOC execution workflow is used (see Section 3 of [RFC9668]), the number of round trips RT_COMB is equal to 1 + ceil(SIZE_BODY / SIZE_BLOCK).

It follows that RT_COMB < RT_ORIG, i.e., the optimized EDHOC execution workflow always yields a lower number of round trips.

Instead, the convenience of using the optimized EDHOC execution workflow becomes questionable if _both_ the following conditions hold:

* COND8: SIZE_BODY <= LIMIT

* COND9: (SIZE_BODY + SIZE_EDHOC_M3) > LIMIT

That is, since SIZE_BODY <= LIMIT, using Block-wise would not be required when using the original EDHOC execution workflow, provided that SIZE_EDHOC_M3 <= LIMIT still holds.

At the same time, using the combined workflow is in itself what actually triggers the use of Block-wise, since (SIZE_BODY + SIZE_EDHOC_M3) > LIMIT.

Therefore, the following round trips are experienced by the CoAP client.

* The original EDHOC execution workflow run without using Block-wise results in a number of round trips RT_ORIG equal to 3.

* The optimized EDHOC execution workflow run using Block-wise results in a number of round trips RT_COMB equal to 1 + ceil(SIZE_BODY / SIZE_BLOCK).

It follows that RT_COMB >= RT_ORIG, i.e., the optimized EDHOC execution workflow might still be not worse than the original EDHOC execution workflow in terms of round trips. This is the case only if the SIZE_BLOCK used is such that ceil(SIZE_BODY / SIZE_BLOCK) is equal to 2, i.e., the plain application data is fragmented into only 2 inner blocks, and thus the EDHOC + OSCORE request is followed by only one more request message transporting the last block of the body.

However, even in such a case, there would be no advantage in terms or round trips compared to the original workflow, while still requiring the CoAP client and the CoAP server to perform the processing due to using the EDHOC + OSCORE request and Block-wise transferring.

Therefore, if both the conditions COND8 and COND9 hold, the CoAP client should not send the EDHOC + OSCORE request. Instead, the CoAP client should continue by switching to the original EDHOC execution workflow. That is, the CoAP client first sends EDHOC message_3 prepended by the EDHOC Connection Identifier C_R encoded as per Section 3.3 of [RFC9528] and then sends the OSCORE-protected CoAP request once the EDHOC execution is completed.

6. Security Considerations

This document provides considerations for implementations of the EDHOC protocol. The security considerations compiled in Section 9 of [RFC9528] and in Section 7 of [RFC9668] apply. The compliance requirements for implementations that are listed in Section 8 of [RFC9528] also apply.

It is foreseeable that the EDHOC protocol will be extended (e.g., as to new cipher suites, new methods, and new types of authentication credentials) and that external security applications will be integrated into EDHOC by embedding the transport of their data in EDHOC EAD items. For implementations that support such extensions and external applications, the related security considerations and compliance requirements also apply.

6.1. Assessing the Correctness of Implementations

Tools relying on fuzz testing such as EDHOC-Fuzzer [EDHOC-Fuzzer] can help assess the correctness of implementations of the EDHOC protocol and of external security applications integrated into EDHOC.

Such tools help finding and amending implementation errors especially related to the following points:

* Non-conformance with the protocol specification (e.g., unintended deviations in performing the protocol steps), which can be a potential source of security vulnerabilities in addition to performance deficiencies.

* Presence of inappropriate states and state transitions in the modeling of the EDHOC execution, e.g., states that are impossible to reach and traverse or that are not part of the protocol specification (which is a particular case of non-conformance).

These states and transitions should be amended or removed, in order to reduce the memory footprint and code complexity and to simplify the implementation, thus reducing the risks of bugs and related security vulnerabilities.

7. IANA Considerations

This document has no actions for IANA.

8. References

8.1. Normative References

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

[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>.

[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>.

[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>.

[RFC9668] Palombini, F., Tiloca, M., Hglund, R., Hristozov, S., and G. Selander, "Using Ephemeral Diffie-Hellman Over COSE (EDHOC) with the Constrained Application Protocol (CoAP) and Object Security for Constrained RESTful Environments (OSCORE)", RFC 9668, DOI 10.17487/RFC9668, November 2024, <https://www.rfc-editor.org/rfc/rfc9668>.

8.2. Informative References

[EDHOC-Fuzzer] Sagonas, K. and T. Typaldos, "EDHOC-Fuzzer: An EDHOC Protocol State Fuzzer", ISSTA 2023: Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis , 13 July 2023, <https://dl.acm.org/doi/10.1145/3597926.3604922>.

[I-D.ietf-ace-edhoc-oscore-profile]
Selander, G., Mattsson, J. P., Tiloca, M., and R. Hglund, "Ephemeral Diffie-Hellman Over COSE (EDHOC) and Object Security for Constrained Environments (OSCORE) Profile for Authentication and Authorization for Constrained Environments (ACE)", Work in Progress, Internet-Draft, draft-ietf-ace-edhoc-oscore-profile-08, 7 July 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-ace-edhoc-oscore-profile-08>.

[I-D.ietf-ace-workflow-and-params]
Tiloca, M. and G. Selander, "Short Distribution Chain (SDC) Workflow and New OAuth Parameters for the Authentication and Authorization for Constrained Environments (ACE) Framework", Work in Progress, Internet-Draft, draft-ietf-ace-workflow-and-params-04, 3 March 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-ace-workflow-and-params-04>.

[I-D.ietf-core-oscore-key-limits]
Hglund, R. and M. Tiloca, "Key Usage Limits for OSCORE", Work in Progress, Internet-Draft, draft-ietf-core-oscore-key-limits-04, 8 January 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-core-oscore-key-limits-04>.

[I-D.ietf-core-oscore-key-update]
Hglund, R. and M. Tiloca, "Key Update for OSCORE (KUDOS)", Work in Progress, Internet-Draft, draft-ietf-core-oscore-key-update-11, 7 July 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-core-oscore-key-update-11>.

[I-D.ietf-cose-cbor-encoded-cert]
Mattsson, J. P., Selander, G., Raza, S., Hglund, J., and M. Furuhed, "CBOR Encoded X.509 Certificates (C509 Certificates)", Work in Progress, Internet-Draft, draft-ietf-cose-cbor-encoded-cert-14, 23 June 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-cose-cbor-encoded-cert-14>.

[I-D.ietf-lake-authz]
Selander, G., Mattsson, J. P., Vuini, M., Fedrecheski, G., and M. Richardson, "Lightweight Authorization using Ephemeral Diffie-Hellman Over COSE (ELA)", Work in Progress, Internet-Draft, draft-ietf-lake-authz-04, 3 March 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-lake-authz-04>.

[RFC2986] Nystrom, M. and B. Kaliski, "PKCS #10: Certification Request Syntax Specification Version 1.7", RFC 2986, DOI 10.17487/RFC2986, November 2000, <https://www.rfc-editor.org/rfc/rfc2986>.

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <https://www.rfc-editor.org/rfc/rfc5280>.

[RFC6960] Santesson, S., Myers, M., Ankney, R., Malpani, A., Galperin, S., and C. Adams, "X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP", RFC 6960, DOI 10.17487/RFC6960, June 2013, <https://www.rfc-editor.org/rfc/rfc6960>.

[RFC8392] Jones, M., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "CBOR Web Token (CWT)", RFC 8392, DOI 10.17487/RFC8392, May 2018, <https://www.rfc-editor.org/rfc/rfc8392>.

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

Appendix A. Foreseen Advanced Processing of Incoming EDHOC message_1

As mentioned in Section 4.1, future developments in EDHOC and in related external security applications might rely on an EAD item in EDHOC message_1 that specifies the authentication credential CRED associated with the Initiator (by value or by reference), as wrapped in a cryptographically protected "envelope".

In order to handle such a case, the processing of an incoming EDHOC message_1 as described in Section 4.1 is extended with additional steps performed by the SPO.

Such an extended side processing shares similarities with that of an incoming EDHOC message_2 or message_3 (see Section 4.3). In particular, similarly to what is compiled in Section 4.3.2 and Section 4.3.3, it consists of the following steps.

* (0) The SPO checks the presence of an EAD item that specifies the authentication credential CRED associated with the Initiator (by value or by reference).

If no such EAD item is found, the SPO moves to Step 12. Otherwise, the SPO moves to Step 1.

* (1) The SPO determines CRED based on an EAD item retrieved at Step 0.

  The EAD item may specify CRED by value or by reference, including a URI or other external reference where CRED can be retrieved from.

  If CRED is already installed, the SPO moves to Step 2. Otherwise, the SPO moves to Step 3.

* (2) The SPO determines if the stored CRED is currently valid, e.g., by verifying that CRED has not expired and has not been revoked.

  Performing such a validation may require the SPO to first process an EAD item included in message_1.

  In case CRED is determined to be valid, the SPO moves to Step 9. Otherwise, the SPO moves to Step 11.

* (3) The SPO attempts to retrieve CRED via an EAD item considered at Step 1. Then, the SPO moves to Step 4.

* (4) If the retrieval of CRED has succeeded, the SPO moves to Step 5. Otherwise, the SPO moves to Step 11.

* (5) If the enforced trust policy for new authentication credentials is "NO-LEARNING" and P does not admit any exceptions that are acceptable to enforce for message_1 (see Section 3.2), the SPO moves to Step 11. Otherwise, the SPO moves to Step 6.

* (6) If this step has been reached, the peer P is not already storing the retrieved CRED and, at the same time, it enforces either the trust policy "LEARNING" or the trust policy "NO-LEARNING" while also enforcing an exception acceptable for message_1 (see Section 3.2).

  Consistently, the SPO determines if CRED is currently valid, e.g., by verifying that CRED has not expired and has not been revoked. Then, the SPO moves to Step 7.

  Validating CRED may require the SPO to first process an EAD item included in message_1.

* (7) If CRED has been determined valid, the SPO moves to Step 8. Otherwise, the SPO moves to Step 11.

* (8) The SPO stores CRED as a valid and trusted authentication credential associated with the other peer, together with corresponding authentication credential identifiers (see Section 3). Then, the SPO moves to Step 9.

* (9) The SPO checks if CRED is fine to use in the context of the ongoing EDHOC session, also depending on the specific identity of the other peer (see Sections 3.5 and D.2 of [RFC9528]).

  If this is the case, the SPO moves to Step 10. Otherwise, the SPO moves to Step 11.

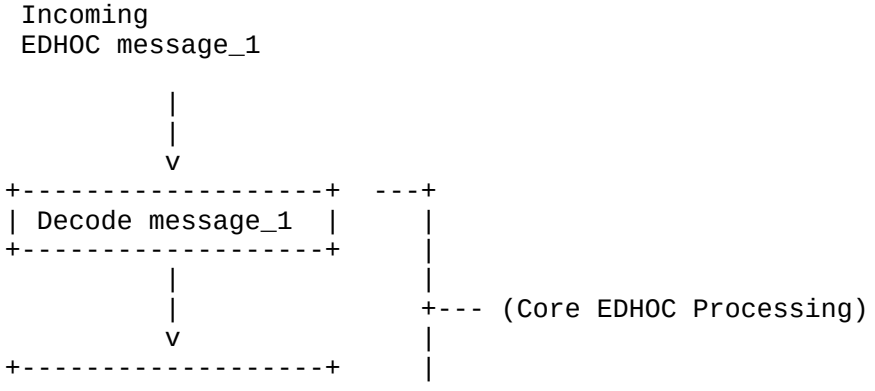
* (10) P uses CRED as authentication credential associated with the other peer in the ongoing EDHOC session. Then, the SPO moves to Step 12.

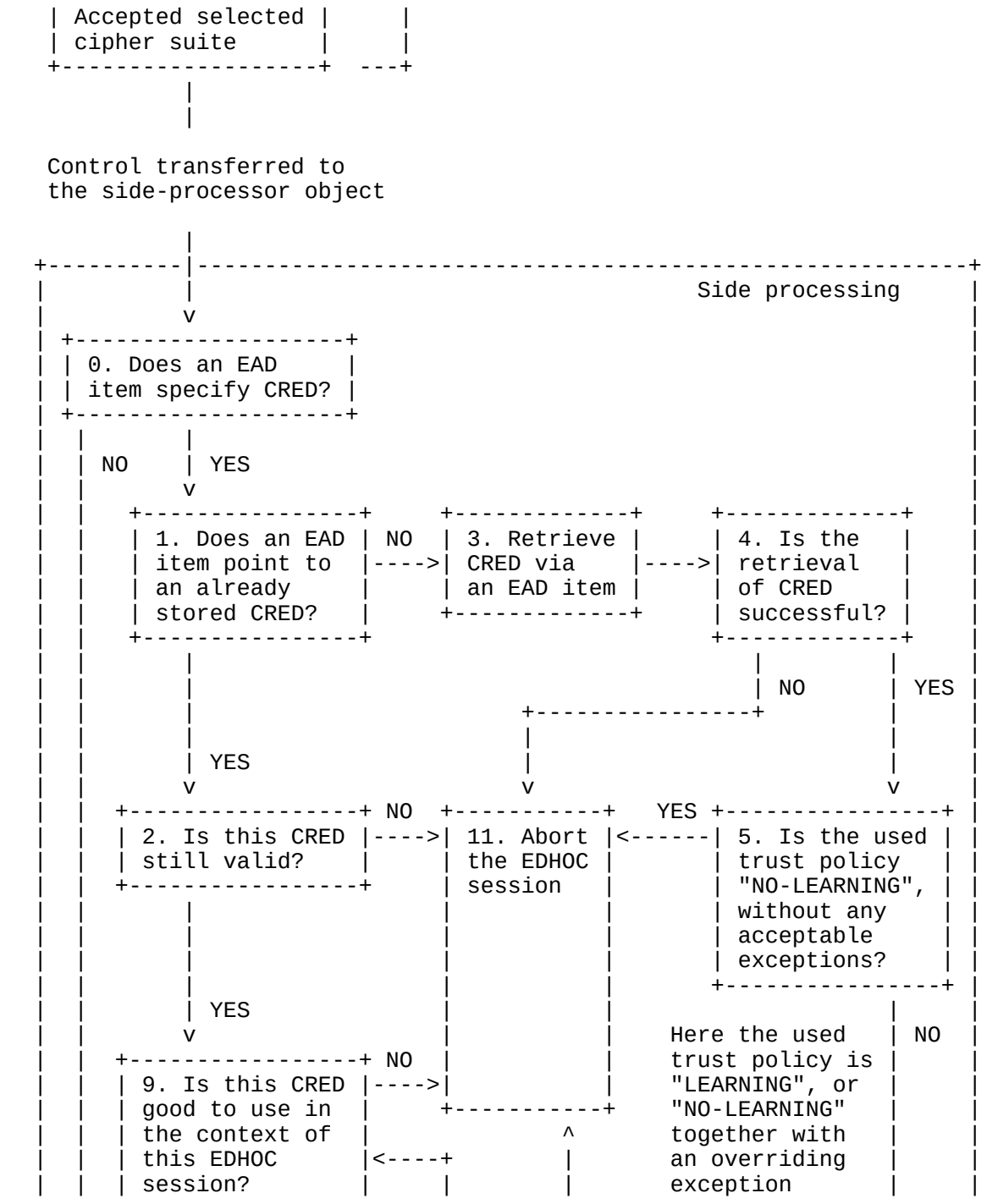
* (11) The SPO has not found a valid authentication credential associated with the other peer that can be used in the ongoing EDHOC session. Therefore, the EDHOC session with the other peer is aborted.

* (12) The SPO processes any EAD item included in message_1 that has not been already processed.

  Once all such EAD items have been processed, the SPO transfers control back to EDHOC. When doing so, the SPO also provides EDHOC with any produced EAD items to include in the EAD field of the next outgoing EDHOC message.

The flowchart in Figure 6 shows the different steps taken for the advanced processing of an incoming EDHOC message_1 defined above.




Accepted selected cipher suite
Control transferred to the side-processor object
Side processing
0. Does an EAD item specify CRED?
NO
YES
1. Does an EAD item point to an already stored CRED?
3. Retrieve CRED via an EAD item
4. Is the retrieval of CRED successful?
NO
YES
2. Is this CRED still valid?
11. Abort the EDHOC session
5. Is the used trust policy "NO-LEARNING", without any acceptable exceptions?
NO
YES
9. Is this CRED good to use in the context of this EDHOC session?
NO
YES
Here the used trust policy is "LEARNING", or "NO-LEARNING" together with an overriding exception

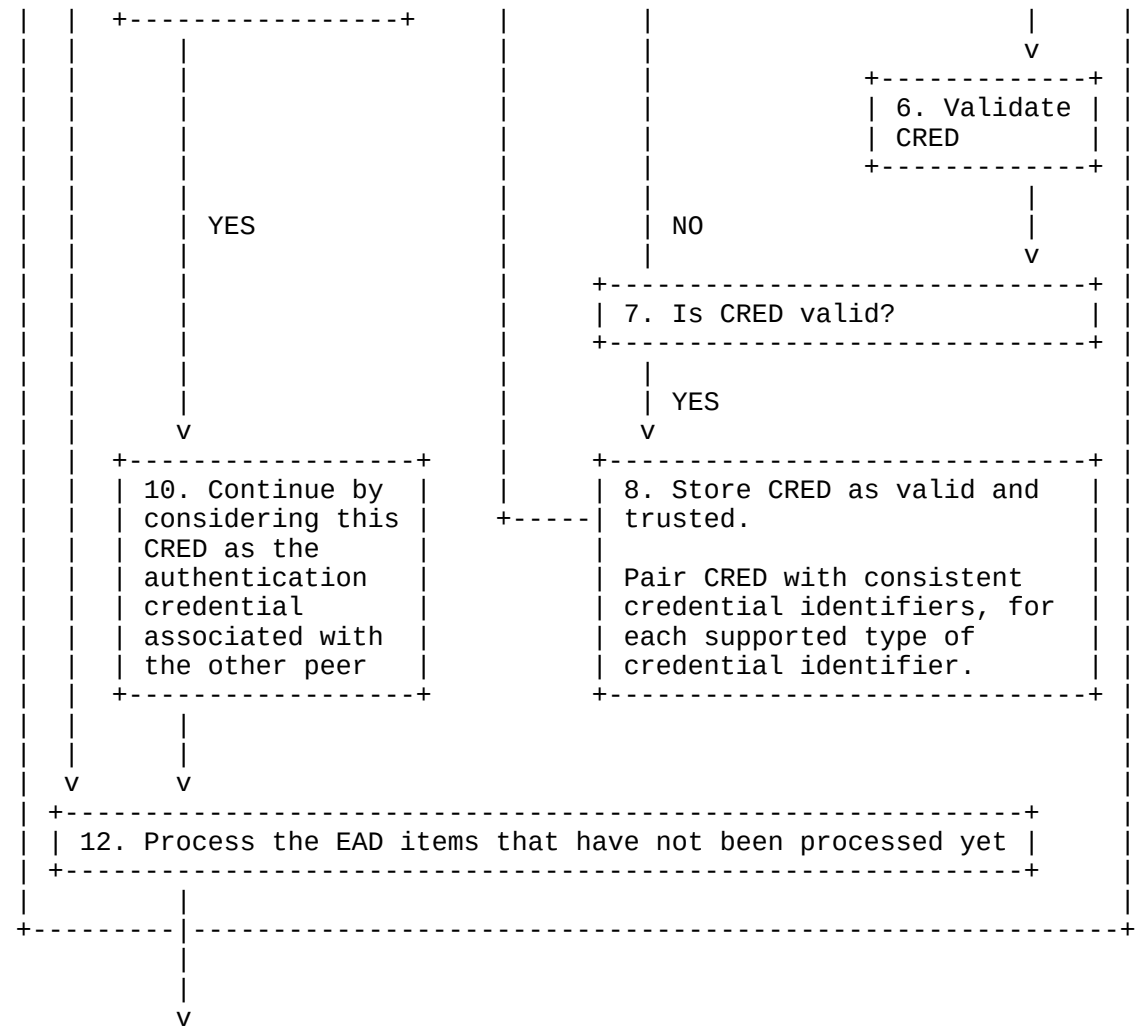


Figure 6: Processing Steps for EDHOC message_1.

Appendix B. Document Updates

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

B.1. Version -03 to -04

* Clarified and re-positioned exceptions to NO-LEARNING policy.
* Added security considerations.
* Appendix on foreseen advanced processing of incoming EDHOC message_1.
* Clarifications and editorial improvements.

B.2. Version -02 to -03

* Consistent use of "trust policy" instead of "trust model".
* More modular presentation of trust policies and their enforcement.
* Alignment with use of EDHOC in the EDHOC and OSCORE profile of ACE.
* Note on follow-up actions for the application after EDHOC completion.
* Removed moot section on special handling when using the EDHOC and OSCORE profile of ACE.
* Consistency checks of authentication credentials from ID_CRED and EAD items.
* Updated reference.
* Clarifications and editorial improvements.

B.3. Version -01 to -02

* Improved content on the EDHOC and OSCORE profile of ACE.
* Admit situation-specific exceptions to the "NO-LEARNING" policy.
* Using the EDHOC and OSCORE profile of ACE with the "NO-LEARNING" policy.
* Revised guidelines on using EDHOC with CoAP and Block-wise.

* Editorial improvements.

B.4. Version -00 to -01

* Added considerations on trust policies when using the EDHOC and OSCORE profile of the ACE framework.

* Placeholder section on special processing when using the EDHOC and OSCORE profile of the ACE framework.

* Added considerations on using EDHOC with CoAP and Block-wise.

* Editorial improvements.

Acknowledgments

The author sincerely thanks Christian Amsss, Geovane Fedrecheski, Rikard Hglund, John Preu Mattsson, 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 project CYPRESS.

Author’s Address

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