Authors: G. Selander   J. Preuß Mattsson   F. Palombini
         Ericsson   Ericsson   Ericsson

Ephemeral Diffie-Hellman Over COSE (EDHOC)

Abstract

This document specifies Ephemeral Diffie-Hellman Over COSE (EDHOC),
a very compact and lightweight authenticated Diffie-Hellman key
exchange with ephemeral keys. EDHOC provides mutual authentication,
forward secrecy, and identity protection. EDHOC is intended for
usage in constrained scenarios and a main use case is to establish
an OSCORE security context. By reusing COSE for cryptography, CBOR
for encoding, and CoAP for transport, the additional code size can
be kept very low.

Status of This Memo

Copyright Notice

Section 4.e of the Trust Legal Provisions and are provided without
warranty as described in the Revised BSD License.

Table of Contents

1. Introduction

1.1. Motivation

Many Internet of Things (IoT) deployments require technologies which
are highly performant in constrained environments [RFC7228]. IoT
devices may be constrained in various ways, including memory,
storage, processing capacity, and power. The connectivity for these
settings may also exhibit constraints such as unreliable and lossy
channels, highly restricted bandwidth, and dynamic topology. The
IETF has acknowledged this problem by standardizing a range of
lightweight protocols and enablers designed for the IoT, including
the Constrained Application Protocol (CoAP, [RFC7252]), Concise
Binary Object Representation (CBOR, [RFC8949]), and Static Context
Header Compression (SCHC, [RFC8724]).

The need for special protocols targeting constrained IoT deployments
extends also to the security domain [I-D.ietf-lake-reqs]. Important

characteristics in constrained environments are the number of round
trips and protocol message sizes, which if kept low can contribute
to good performance by enabling transport over a small number of
radio frames, reducing latency due to fragmentation or duty cycles,
etc. Another important criteria is code size, which may be
prohibitive for certain deployments due to device capabilities or
network load during firmware update. Some IoT deployments also need
to support a variety of underlying transport technologies,
potentially even with a single connection.

Some security solutions for such settings exist already. CBOR Object
Signing and Encryption (COSE, [I-D.ietf-cose-rfc8152bis-struct])
specifies basic application-layer security services efficiently
encoded in CBOR. Another example is Object Security for Constrained
RESTful Environments (OSCORE, [RFC8613]) which is a lightweight
communication security extension to CoAP using CBOR and COSE. In
order to establish good quality cryptographic keys for security
protocols such as COSE and OSCORE, the two endpoints may run an
authenticated Diffie-Hellman key exchange protocol, from which
shared secret key material can be derived. Such a key exchange
protocol should also be lightweight; to prevent bad performance in
case of repeated use, e.g., due to device rebooting or frequent
rekeying for security reasons; or to avoid latencies in a network
formation setting with many devices authenticating at the same time.

This document specifies Ephemeral Diffie-Hellman Over COSE (EDHOC),
a lightweight authenticated key exchange protocol providing good
security properties including forward secrecy, identity protection,
and cipher suite negotiation. Authentication can be based on raw
public keys (RPK) or public key certificates and requires the
application to provide input on how to verify that endpoints are
trusted. This specification focuses on referencing instead of
transporting credentials to reduce message overhead. EDHOC does
currently not support pre-shared key (PSK) authentication as
authentication with static Diffie-Hellman public keys by reference
produces equally small message sizes but with much simpler key
distribution and identity protection.

EDHOC makes use of known protocol constructions, such as SIGMA
[SIGMA] and Extract-and-Expand [RFC5869]. EDHOC uses COSE for
cryptography and identification of credentials (including COSE_Key,
CWT, CCS, X.509, C509, see Section 3.5.3). COSE provides crypto
agility and enables the use of future algorithms and credentials
targeting IoT.

1.2. Use of EDHOC

EDHOC is designed for highly constrained settings making it
especially suitable for low-power wide area networks [RFC8376] such

as Cellular IoT, 6TiSCH, and LoRaWAN. A main objective for EDHOC is to be a lightweight authenticated key exchange for OSCORE, i.e., to provide authentication and session key establishment for IoT use cases such as those built on CoAP [RFC7252]. CoAP is a specialized web transfer protocol for use with constrained nodes and networks, providing a request/response interaction model between application endpoints. As such, EDHOC is targeting a large variety of use cases involving 'things' with embedded microcontrollers, sensors, and actuators.

A typical setting is when one of the endpoints is constrained or in a constrained network, and the other endpoint is a node on the Internet (such as a mobile phone) or at the edge of the constrained network (such as a gateway). Thing-to-thing interactions over constrained networks are also relevant since both endpoints would then benefit from the lightweight properties of the protocol. EDHOC could e.g., be run when a device connects for the first time, or to establish fresh keys which are not revealed by a later compromise of the long-term keys. Further security properties are described in Section 8.1.

EDHOC enables the reuse of the same lightweight primitives as OSCORE: CBOR for encoding, COSE for cryptography, and CoAP for transport. By reusing existing libraries, the additional code size can be kept very low. Note that, while CBOR and COSE primitives are built into the protocol messages, EDHOC is not bound to a particular transport. Transfer of EDHOC messages in CoAP payloads is detailed in Appendix A.3.

1.3. Message Size Examples

Compared to the DTLS 1.3 handshake [I-D.ietf-tls-dtls13] with ECDHE and connection ID, the number of bytes in EDHOC + CoAP can be less than 1/6 when RPK authentication is used, see [I-D.ietf-lwig-security-protocol-comparison]. Figure 1 shows examples of message sizes for EDHOC with different kinds of authentication keys and different COSE header parameters for identification: static Diffie-Hellman keys or signature keys, either in CBOR Web Token (CWT) / CWT Claims Set (CCS) [RFC8392] identified by a key identifier using 'kid' [I-D.ietf-cose-rfc8152bis-struct], or in X.509 certificates identified by a hash value using 'x5t' [I-D.ietf-cose-x509].

```
=======================================================
                  Static DH Keys      Signature Keys
                 --------------       --------------
                  kid       x5t        kid       x5t
        -----------------------------------------------
        message_1          37        37         37        37
        message_2          45        58        102       115
        message_3          19        33         77        90
        -----------------------------------------------
        Total             101       128        216       242
=======================================================
```

Figure 1: Example of message sizes in bytes.

1.4. Document Structure

The remainder of the document is organized as follows: Section 2
outlines EDHOC authenticated with digital signatures, Section 3
describes the protocol elements of EDHOC, including formatting of
the ephemeral public keys, Section 4 specifies the key derivation,
Section 5 specifies message processing for EDHOC authenticated with
signature keys or static Diffie-Hellman keys, Section 6 describes
the error messages, and Appendix A shows how to transfer EDHOC with
CoAP and establish an OSCORE security context.

1.5. Terminology and Requirements Language

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

Readers are expected to be familiar with the terms and concepts
described in CBOR [RFC8949], CBOR Sequences [RFC8742], COSE
structures and processing [I-D.ietf-cose-rfc8152bis-struct], COSE
algorithms [I-D.ietf-cose-rfc8152bis-algs], CWT and CWT Claims Set
[RFC8392], and CDDL [RFC8610]. The Concise Data Definition Language
(CDDL) is used to express CBOR data structures [RFC8949]. Examples
of CBOR and CDDL are provided in Appendix C.1. When referring to
CBOR, this specification always refers to Deterministically Encoded
CBOR as specified in Sections 4.2.1 and 4.2.2 of [RFC8949]. The
single output from authenticated encryption (including the
authentication tag) is called "ciphertext", following [RFC5116].

2. EDHOC Outline

EDHOC specifies different authentication methods of the Diffie-
Hellman key exchange: digital signatures and static Diffie-Hellman
keys. This section outlines the digital signature-based method.

Further details of protocol elements and other authentication
methods are provided in the remainder of this document.

SIGMA (SIGn-and-MAc) is a family of theoretical protocols with a
large number of variants [SIGMA]. Like IKEv2 [RFC7296] and (D)TLS
1.3 [RFC8446], EDHOC authenticated with digital signatures is built
on a variant of the SIGMA protocol which provides identity
protection of the initiator (SIGMA-I) against active attackers, and
like IKEv2 [RFC7296], EDHOC implements the MAC-then-Sign variant of
the SIGMA-I protocol shown in Figure 2.

```
Initiator                                                      Responder
|                              G_X                                    |
+-------------------------------------------------------------------->|
|                                                                     |
|      G_Y, Enc( ID_CRED_R, Sig( R; MAC( CRED_R, G_X, G_Y ) ) )       |
|<-------------------------------------------------------------------+
|                                                                     |
|        AEAD( ID_CRED_I, Sig( I; MAC( CRED_I, G_Y, G_X ) ) )         |
+-------------------------------------------------------------------->|
|                                                                     |
```
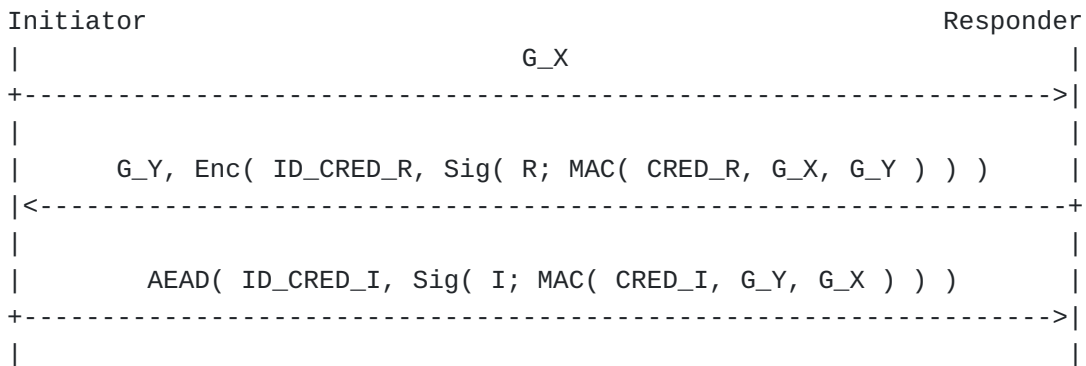
        Figure 2: MAC-then-Sign variant of the SIGMA-I protocol.

The parties exchanging messages are called Initiator (I) and
Responder (R). They exchange ephemeral public keys, compute a shared
secret, and derive symmetric application keys used to protect
application data.

  *G_X and G_Y are the ECDH ephemeral public keys of I and R,
   respectively.

  *CRED_I and CRED_R are the credentials containing the public
   authentication keys of I and R, respectively.

  *ID_CRED_I and ID_CRED_R are credential identifiers enabling the
   recipient party to retrieve the credential of I and R,
   respectively.

  *Sig(I; .) and Sig(R; .) denote signatures made with the private
   authentication key of I and R, respectively.

  *Enc(), AEAD(), and MAC() denotes encryption, authenticated
   encryption with additional data, and message authentication code
   using keys derived from the shared secret.

In order to create a "full-fledged" protocol some additional
protocol elements are needed. EDHOC adds:

  *Transcript hashes (hashes of message data) TH_2, TH_3, TH_4 used
   for key derivation and as additional authenticated data.

  *Computationally independent keys derived from the ECDH shared
   secret and used for authenticated encryption of different
   messages.

  *An optional fourth message giving explicit key confirmation to I
   in deployments where no protected application data is sent from R
   to I.

  *A key material exporter and a key update function with forward
   secrecy.

  *Verification of a common preferred cipher suite.

  *Method types and error handling.

  *Selection of connection identifiers C_I and C_R which may be used
   to identify established keys or protocol state.

  *Transport of external authorization data.

EDHOC is designed to encrypt and integrity protect as much
information as possible, and all symmetric keys are derived using as
much previous information as possible. EDHOC is furthermore designed
to be as compact and lightweight as possible, in terms of message
sizes, processing, and the ability to reuse already existing CBOR,
COSE, and CoAP libraries.

To simplify for implementors, the use of CBOR and COSE in EDHOC is
summarized in Appendix C. Test vectors including CBOR diagnostic
notation are provided in [I-D.selander-lake-traces].

3. Protocol Elements

3.1. General

The EDHOC protocol consists of three mandatory messages (message_1,
message_2, message_3) between Initiator and Responder, an optional
fourth message (message_4), and an error message. All EDHOC messages
are CBOR Sequences [RFC8742]. Figure 3 illustrates an EDHOC message
flow with the optional fourth message as well as the content of each
message. The protocol elements in the figure are introduced in
Section 3 and Section 5. Message formatting and processing is
specified in Section 5 and Section 6.

Application data may be protected using the agreed application algorithms (AEAD, hash) in the selected cipher suite (see Section 3.6) and the application can make use of the established connection identifiers C_I and C_R (see Section 3.3). EDHOC may be used with the media type application/edhoc defined in Section 9.

The Initiator can derive symmetric application keys after creating EDHOC message_3, see Section 4.3. Protected application data can therefore be sent in parallel or together with EDHOC message_3. EDHOC message_4 is typically not sent.

```
Initiator                                                   Responder
|                  METHOD, SUITES_I, G_X, C_I, EAD_1                 |
+------------------------------------------------------------------->|
|                            message_1                               |
|                                                                    |
|       G_Y, Enc( ID_CRED_R, Signature_or_MAC_2, EAD_2 ), C_R        |
|<------------------------------------------------------------------+
|                            message_2                               |
|                                                                    |
|          AEAD( ID_CRED_I, Signature_or_MAC_3, EAD_3 )              |
+------------------------------------------------------------------->|
|                            message_3                               |
|                                                                    |
|                         AEAD( EAD_4 )                              |
|<- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - +
|                            message_4                               |
```
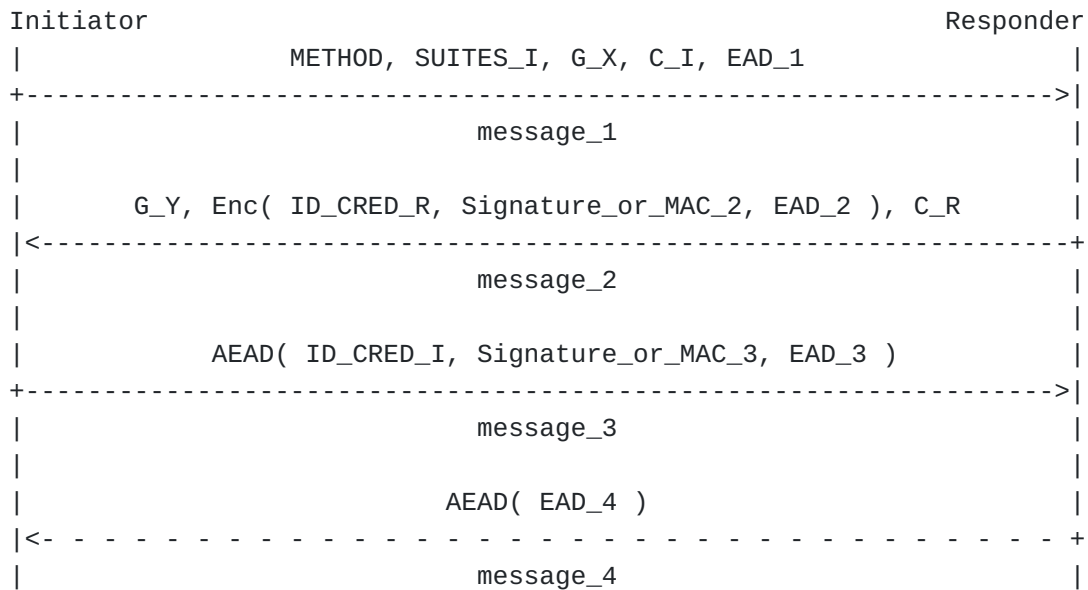
Figure 3: EDHOC Message Flow with the Optional Fourth Message

3.2. Method

The data item METHOD in message_1 (see Section 5.2.1), is an integer specifying the authentication method. EDHOC supports authentication with signature or static Diffie-Hellman keys, as defined in the four authentication methods: 0, 1, 2, and 3, see Figure 4. When using a static Diffie-Hellman key the authentication is provided by a Message Authentication Code (MAC) computed from an ephemeral-static ECDH shared secret which enables significant reductions in message sizes.

The Initiator and the Responder need to have agreed on a single method to be used for EDHOC, see Section 3.9.

```
+-------+------------------+------------------+------------------+
| Value | Initiator        | Responder        | Reference        |
+-------+------------------+------------------+------------------+
|     0 | Signature Key    | Signature Key    | [[this document]] |
|     1 | Signature Key    | Static DH Key    | [[this document]] |
|     2 | Static DH Key    | Signature Key    | [[this document]] |
|     3 | Static DH Key    | Static DH Key    | [[this document]] |
+-------+------------------+------------------+------------------+
```

Figure 4: Method Types

3.3. Connection Identifiers

EDHOC includes the selection of connection identifiers (C_I, C_R)
identifying a connection for which keys are agreed.

Connection identifiers may be used to correlate EDHOC messages and
facilitate the retrieval of protocol state during EDHOC protocol
execution (see Section 3.4) or in a subsequent application protocol,
e.g., OSCORE (see Section 3.3.2). The connection identifiers do not
have any cryptographic purpose in EDHOC.

Connection identifiers in EDHOC are byte strings or integers,
encoded in CBOR. One byte connection identifiers (the integers -24
to 23 and the empty CBOR byte string h'') are realistic in many
scenarios as most constrained devices only have a few connections.

3.3.1. Selection of Connection Identifiers

C_I and C_R are chosen by I and R, respectively. The Initiator
selects C_I and sends it in message_1 for the Responder to use as a
reference to the connection in communications with the Initiator.
The Responder selects C_R and sends in message_2 for the Initiator
to use as a reference to the connection in communications with the
Responder.

If connection identifiers are used by an application protocol for
which EDHOC establishes keys then the selected connection
identifiers SHALL adhere to the requirements for that protocol, see
Section 3.3.2 for an example.

3.3.2. Use of Connection Identifiers with OSCORE

For OSCORE, the choice of a connection identifier results in the
endpoint selecting its Recipient ID, see Section 3.1 of [RFC8613],
for which certain uniqueness requirements apply, see Section 3.3 of
[RFC8613]. Therefore, the Initiator and the Responder MUST NOT
select connection identifiers such that it results in same OSCORE
Recipient ID. Since the Recipient ID is a byte string and a EDHOC
connection identifier is either a CBOR byte string or a CBOR

integer, care must be taken when selecting the connection
identifiers and converting them to Recipient IDs. A mapping from
EDHOC connection identifier to OSCORE Recipient ID is specified in
[Appendix A.1](#).

3.4. Transport

Cryptographically, EDHOC does not put requirements on the lower
layers. EDHOC is not bound to a particular transport layer and can
even be used in environments without IP. The transport is
responsible, where necessary, to handle:

  *message loss,

  *message reordering,

  *message duplication,

  *fragmentation,

  *demultiplex EDHOC messages from other types of messages,

  *denial-of-service protection,

  *message correlation.

The Initiator and the Responder need to have agreed on a transport
to be used for EDHOC, see [Section 3.9](#).

3.4.1. Use of Connection Identifiers for EDHOC Message Correlation

The transport needs to support the correlation between EDHOC
messages and facilitate the retrieval of protocol state during EDHOC
protocol execution, including an indication of a message being
message_1. The correlation may reuse existing mechanisms in the
transport protocol. For example, the CoAP Token may be used to
correlate EDHOC messages in a CoAP response and an associated CoAP
request.

Connection identifiers may be used to correlate EDHOC messages and
facilitate the retrieval of protocol state during EDHOC protocol
execution. EDHOC transports that do not inherently provide
correlation across all messages of an exchange can send connection
identifiers along with EDHOC messages to gain that required
capability, e.g., by prepending the appropriate connection
identifier (when available from the EDHOC protocol) to the EDHOC
message. Transport of EDHOC in CoAP payloads is described in
[Appendix A.3](#), which also shows how to use connection identifiers and
message_1 indication with CoAP.

3.5. Authentication Parameters

EDHOC supports various settings for how the other endpoint's authentication (public) key is transported, identified, and trusted as described in this section.

The authentication key (see Section 3.5.2) is used in several parts of EDHOC:

1. as part of the authentication credential included in the integrity calculation

2. for verification of the Signature_or_MAC field in message_2 and message_3 (see Section 5.3.2 and Section 5.4.2)

3. in the key derivation (in case of a static Diffie-Hellman key, see Section 4).

The authentication credential (CRED_x) contains, in addition to the authentication key, also the authentication key algorithm and optionally other parameters such as identity, key usage, expiry, issuer, etc. (see Section 3.5.3). Identical authentication credentials need to be established in both endpoints to be able to verify integrity. For many settings it is not necessary to transport the authentication credential within EDHOC over constrained links, for example, it may be pre-provisioned or acquired out-of-band over less constrained links.

EDHOC relies on COSE for identification of authentication credentials (using ID_CRED_x, see Section 3.5.4) and supports all credential types for which COSE header parameters are defined (see Section 3.5.3).

The choice of authentication credential depends also on the trust model (see Section 3.5.1). For example, a certificate or CWT may rely on a trusted third party, whereas a CCS or a self-signed certificate/CWT may be used when trust in the public key can be achieved by other means, or in the case of trust-on-first-use.

The type of authentication key, authentication credential, and the way to identify the credential have a large impact on the message size. For example, the signature_or_MAC field is much smaller with a static DH key than with a signature key. A CCS is much smaller than a self-signed certificate/CWT, but if it is possible to reference the credential with a COSE header like 'kid', then that is typically much smaller than to transport a CCS.

3.5.1. Identities and trust anchors

Policies for what connections to allow are typically set based on the identity of the other party, and parties typically only allow connections from a specific identity or a small restricted set of identities. For example, in the case of a device connecting to a network, the network may only allow connections from devices which authenticate with certificates having a particular range of serial numbers and signed by a particular CA. On the other hand, the device may only be allowed to connect to a network which authenticates with a particular public key (information of which may be provisioned, e.g., out of band or in the external authorization data, see Section 3.8). The EDHOC implementation or the application must enforce information about the intended endpoint, and in particular whether it is a specific identity or a set of identities. Either EDHOC passes information about identity to the application for a decision, or EDHOC needs to have access to relevant information and makes the decision on its own.

EDHOC assumes the existence of mechanisms (certification authority, trusted third party, pre-provisioning, etc.) for specifying and distributing authentication credentials.

*When a Public Key Infrastructure (PKI) is used with certificates, the trust anchor is a Certification Authority (CA) certificate, and the identity is the subject whose unique name (e.g., a domain name, NAI, or EUI) is included in the endpoint's certificate. In order to run EDHOC each party needs at least one CA public key certificate, or just the public key, and a specific identity or set of identities it is allowed to communicate with. Only validated public-key certificates with an allowed subject name, as specified by the application, are to be accepted. EDHOC provides proof that the other party possesses the private authentication key corresponding to the public authentication key in its certificate. The certification path provides proof that the subject of the certificate owns the public key in the certificate.

*Similarly, when a PKI is used with CWTs, each party needs to have a trusted third party public key as trust anchor to verify the end-entity CWTs, and a specific identity or set of identities in the 'sub' (subject) claim of the CWT to determine if it is allowed to communicate with. The trusted third party public key can, e.g., be stored in a self-signed CWT or in a CCS.

*When PKI is not used (CCS, self-signed certificate/CWT), the trust anchor is the authentication key of the other party. In this case, the identity is typically directly associated to the authentication key of the other party. For example, the name of

the subject may be a canonical representation of the public key. Alternatively, if identities can be expressed in the form of unique subject names assigned to public keys, then a binding to identity can be achieved by including both public key and associated subject name in the protocol message computation: CRED_I or CRED_R may be a self-signed certificate/CWT or CCS containing the authentication key and the subject name, see Section 3.5.3. In order to run EDHOC, each endpoint needs a specific authentication key/unique associated subject name, or a set of public authentication keys/unique associated subject names, which it is allowed to communicate with. EDHOC provides the proof that the other party possesses the private authentication key corresponding to the public authentication key.

To prevent misbinding attacks in systems where an attacker can register public keys without proving knowledge of the private key, SIGMA [SIGMA] enforces a MAC to be calculated over the "identity". EDHOC follows SIGMA by calculating a MAC over the whole credential, which in case of an X.509 or C509 certificate includes the "subject" and "subjectAltName" fields, and in the case of CWT or CCS includes the "sub" claim. While the SIGMA paper only focuses on the identity, the same principle is true for other information such as policies associated to the public key.

3.5.2. Authentication Keys

The authentication key (i.e. the public key used for authentication) MUST be a signature key or static Diffie-Hellman key. The Initiator and the Responder MAY use different types of authentication keys, e.g., one uses a signature key and the other uses a static Diffie-Hellman key. The authentication key algorithm needs to be compatible with the method and the cipher suite. The authentication key algorithm needs to be compatible with the EDHOC key exchange algorithm when static Diffie-Hellman authentication is used, and compatible with the EDHOC signature algorithm when signature authentication is used.

Note that for most signature algorithms, the signature is determined by the signature algorithm and the authentication key algorithm together. When using static Diffie-Hellman keys the Initiator's and Responder's private authentication keys are called I and R, respectively, and the public authentication keys are called G_I and G_R, respectively.

For X.509 the authentication key is represented with a SubjectPublicKeyInfo field. For CWT and CCS, the authentication key is represented with a 'cnf' claim [RFC8747] containing a COSE_Key [I-D.ietf-cose-rfc8152bis-struct].

3.5.3. Authentication Credentials

The authentication credentials, CRED_I and CRED_R, contain the public authentication key of the Initiator and the Responder, respectively.

EDHOC relies on COSE for identification of authentication credentials (see Section 3.5.4) and supports all credential types for which COSE header parameters are defined including X.509 [RFC5280], C509 [I-D.ietf-cose-cbor-encoded-cert], CWT [RFC8392] and CWT Claims Set (CCS) [RFC8392]. When the identified credential is a chain or bag, CRED_x is just the end-entity X.509 or C509 certificate / CWT. In X.509 and C509 certificates, signature keys typically have key usage "digitalSignature" and Diffie-Hellman public keys typically have key usage "keyAgreement".

CRED_x needs to be defined such that it is identical when used by Initiator or Responder. The Initiator and Responder are expected to agree on a specific encoding of the credential, see Section 3.9. It is RECOMMENDED that the COSE 'kid' parameter, when used, refers to a specific encoding. The Initiator and Responder SHOULD use an available authentication credential (transported in EDHOC or otherwise provisioned) without re-encoding. If for some reason re-encoding of the authentication credential may occur, then a potential common encoding for CBOR based credentials is bytewise lexicographic order of their deterministic encodings as specified in Section 4.2.1 of [RFC8949].

  *When the authentication credential is an X.509 certificate, CRED_x SHALL be the end-entity DER encoded certificate, encoded as a bstr [I-D.ietf-cose-x509].

  *When the authentication credential is a C509 certificate, CRED_x SHALL be the end-entity C509Certificate [I-D.ietf-cose-cbor-encoded-cert]

  *When the authentication credential is a COSE_Key in a CWT, CRED_x SHALL be the untagged CWT.

  *When the authentication credential is a COSE_Key but not in a CWT, CRED_x SHALL be an untagged CCS.

    -Naked COSE_Keys are thus dressed as CCS when used in EDHOC, which is done by prefixing the COSE_Key with 0xA108A101.

An example of a CRED_x is shown below:

```
{                                                  /CCS/
  2 : "42-50-31-FF-EF-37-32-39",          /sub/
  8 : {                                            /cnf/
    1 : {                                          /COSE_Key/
      1 : 1,                                       /kty/
      2 : 0,                                       /kid/
     -1 : 4,                                       /crv/
     -2 : h'b1a3e89460e88d3a8d54211dc95f0b90    /x/
           3ff205eb71912d6db8f4af980d2db83a'
    }
  }
}
```

Figure 5: A CCS Containing an X25519 Static Diffie-Hellman Key and an
EUI-64 Identity.

3.5.4. Identification of Credentials

ID_CRED_R and ID_CRED_I are transported in message_2 and message_3,
respectively (see Section 5.3.2 and Section 5.4.2). They are used to
identify and optionally transport the authentication keys of the
Initiator and the Responder, respectively. ID_CRED_I and ID_CRED_R
do not have any cryptographic purpose in EDHOC since EDHOC integrity
protects the authentication credential. EDHOC relies on COSE for
identification of authentication credentials and supports all types
of COSE header parameters used to identify authentication
credentials including X.509, C509, CWT and CCS.

  *ID_CRED_R is intended to facilitate for the Initiator to retrieve
   the Responder's authentication key.

  *ID_CRED_I is intended to facilitate for the Responder to retrieve
   the Initiator's authentication key.

ID_CRED_I and ID_CRED_R are COSE header maps and contains one or
more COSE header parameter. ID_CRED_I and ID_CRED_R MAY contain
different header parameters. The header parameters typically provide
some information about the format of authentication credential.

Note that COSE header parameters in ID_CRED_x are used to identify
the sender's authentication credential. There is therefore no reason
to use the "-sender" header parameters, such as x5t-sender, defined
in Section 3 of [I-D.ietf-cose-x509]. Instead, the corresponding
parameter without "-sender", such as x5t, SHOULD be used.

Example: X.509 certificates can be identified by a hash value using
the 'x5t' parameter:

  *ID_CRED_x = { 34 : COSE_CertHash }, for x = I or R,

Example: CWT or CCS can be identified by a key identifier using the 'kid' parameter:

*ID_CRED_x = { 4 : key_id_x }, where key_id_x : kid, for x = I or R.

Note that 'kid' is extended to support int values to allow more one-byte identifiers (see Section 9.7 and Section 9.8) which may be useful in many scenarios since constrained devices only have a few keys. As stated in Section 3.1 of [I-D.ietf-cose-rfc8152bis-struct], applications MUST NOT assume that 'kid' values are unique and several keys associated with a 'kid' may need to be checked before the correct one is found. Applications might use additional information such as 'kid context' or lower layers to determine which key to try first. Applications should strive to make ID_CRED_x as unique as possible, since the recipient may otherwise have to try several keys.

See Appendix C.3 for more examples.

3.6. Cipher Suites

An EDHOC cipher suite consists of an ordered set of algorithms from the "COSE Algorithms" and "COSE Elliptic Curves" registries as well as the EDHOC MAC length. Algorithms need to be specified with enough parameters to make them completely determined. EDHOC is currently only specified for use with key exchange algorithms of type ECDH curves, but any Key Encapsulation Method (KEM), including Post-Quantum Cryptography (PQC) KEMs, can be used in method 0, see Section 8.4. Use of other types of key exchange algorithms to replace static DH authentication (method 1,2,3) would likely require a specification updating EDHOC with new methods.

EDHOC supports all signature algorithms defined by COSE, including PQC signature algorithms such as HSS-LMS. Just like in TLS 1.3 [RFC8446] and IKEv2 [RFC7296], a signature in COSE is determined by the signature algorithm and the authentication key algorithm together, see Section 3.5.2. The exact details of the authentication key algorithm depend on the type of authentication credential. COSE supports different formats for storing the public authentication keys including COSE_Key and X.509, which have different names and ways to represent the authentication key and the authentication key algorithm.

An EDHOC cipher suite consists of the following parameters:

*EDHOC AEAD algorithm

*EDHOC hash algorithm

*EDHOC MAC length in bytes (Static DH)

  *EDHOC key exchange algorithm (ECDH curve)

  *EDHOC signature algorithm

  *Application AEAD algorithm

  *Application hash algorithm

Each cipher suite is identified with a pre-defined int label.

EDHOC can be used with all algorithms and curves defined for COSE. Implementation can either use any combination of COSE algorithms and parameters to define their own private cipher suite, or use one of the pre-defined cipher suites. Private cipher suites can be identified with any of the four values -24, -23, -22, -21. The pre-defined cipher suites are listed in the IANA registry ([Section 9.2](#)) with initial content outlined here:

  Cipher suites 0-3, based on AES-CCM, are intended for constrained IoT where message overhead is a very important factor. Note that AES-CCM-16-64-128 and AES-CCM-16-64-128 are compatible with the IEEE CCM mode.

    -Cipher suites 1 and 3 use a larger tag length (128-bit) in EDHOC than in the Application AEAD algorithm (64-bit).

  *Cipher suites 4 and 5, based on ChaCha20, are intended for less constrained applications and only use 128-bit tag lengths.

  *Cipher suite 6, based on AES-GCM, is for general non-constrained applications. It uses high performance algorithms that are widely supported.

  *Cipher suites 24 and 25 are intended for high security applications such as government use and financial applications. These cipher suites do not share any algorithms. Cipher suite 24 is compatible with the CNSA suite [[CNSA](#)].

The different methods ([Section 3.2](#)) use the same cipher suites, but some algorithms are not used in some methods. The EDHOC signature algorithm is not used in methods without signature authentication.

The Initiator needs to have a list of cipher suites it supports in order of preference. The Responder needs to have a list of cipher suites it supports. SUITES_I contains cipher suites supported by the Initiator, formatted and processed as detailed in [Section 5.2.1](#) to secure the cipher suite negotiation. Examples of cipher suite negotiation are given in [Section 6.3.2](#).

3.7. Ephemeral Public Keys

EDHOC always uses compact representation of elliptic curve points, see Appendix B. In COSE compact representation is achieved by formatting the ECDH ephemeral public keys as COSE_Keys of type EC2 or OKP according to Sections 7.1 and 7.2 of [I-D.ietf-cose-rfc8152bis-algs], but only including the 'x' parameter in G_X and G_Y. For Elliptic Curve Keys of type EC2, compact representation MAY be used also in the COSE_Key. If the COSE implementation requires an 'y' parameter, the value y = false SHALL be used. COSE always use compact output for Elliptic Curve Keys of type EC2.

3.8. External Authorization Data (EAD)

In order to reduce round trips and number of messages or to simplify processing, external security applications may be integrated into EDHOC by transporting authorization related data in the messages. One example is third-party identity and authorization information protected out of scope of EDHOC [I-D.selander-ace-ake-authz]. Another example is a certificate enrolment request or the resulting issued certificate.

EDHOC allows opaque external authorization data (EAD) to be sent in the EDHOC messages. External authorization data sent in message_1 (EAD_1) or message_2 (EAD_2) should be considered unprotected by EDHOC, see Section 8.5. External authorization data sent in message_3 (EAD_3) or message_4 (EAD_4) is protected between Initiator and Responder.

External authorization data is a CBOR sequence (see Appendix C.1) consisting of one or more (ead_label, ead_value) pairs as defined below:

```
ead = 1* (
  ead_label : int,
  ead_value : any,
)
```

Applications using external authorization data need to specify format, processing, and security considerations and register the (ead_label, ead_value) pair, see Section 9.5. The CDDL type of ead_value is determined by the int ead_label and MUST be specified.

The EAD fields of EDHOC are not intended for generic application data. Since data carried in EAD_1 and EAD_2 fields may not be protected, special considerations need to be made such that it does not violate security and privacy requirements of the service which uses this data. Moreover, the content in an EAD field may impact the security properties provided by EDHOC. Security applications making use of the EAD fields must perform the necessary security analysis.

3.9. Applicability Statement

EDHOC requires certain parameters to be agreed upon between Initiator and Responder. Some parameters can be agreed through the protocol execution (specifically cipher suite negotiation, see Section 3.6) but other parameters may need to be known out-of-band (e.g., which authentication method is used, see Section 3.2).

The purpose of the applicability statement is to describe the intended use of EDHOC to allow for the relevant processing and verifications to be made, including things like:

1. How the endpoint detects that an EDHOC message is received. This includes how EDHOC messages are transported, for example in the payload of a CoAP message with a certain Uri-Path or Content-Format; see Appendix A.3.

    *The method of transporting EDHOC messages may also describe data carried along with the messages that are needed for the transport to satisfy the requirements of Section 3.4, e.g., connection identifiers used with certain messages, see Appendix A.3.

2. Authentication method (METHOD; see Section 3.2).

3. Profile for authentication credentials (CRED_I, CRED_R; see Section 3.5.3), e.g., profile for certificate or CCS, including supported authentication key algorithms (subject public key algorithm in X.509 or C509 certificate).

4. Type used to identify authentication credentials (ID_CRED_I, ID_CRED_R; see Section 3.5.4).

5. Use and type of external authorization data (EAD_1, EAD_2, EAD_3, EAD_4; see Section 3.8).

6. Identifier used as identity of endpoint; see Section 3.5.1.

7. If message_4 shall be sent/expected, and if not, how to ensure a protected application message is sent from the Responder to the Initiator; see Section 5.5.

The applicability statement may also contain information about supported cipher suites. The procedure for selecting and verifying cipher suite is still performed as described in Section 5.2.1 and Section 6.3, but it may become simplified by this knowledge.

An example of an applicability statement is shown in Appendix D.

For some parameters, like METHOD, ID_CRED_x, type of EAD, the receiver is able to verify compliance with applicability statement, and if it needs to fail because of incompliance, to infer the reason why the protocol failed.

For other parameters, like CRED_x in the case that it is not transported, it may not be possible to verify that incompliance with applicability statement was the reason for failure: Integrity verification in message_2 or message_3 may fail not only because of wrong authentication credential. For example, in case the Initiator uses public key certificate by reference (i.e., not transported within the protocol) then both endpoints need to use an identical data structure as CRED_I or else the integrity verification will fail.

Note that it is not necessary for the endpoints to specify a single transport for the EDHOC messages. For example, a mix of CoAP and HTTP may be used along the path, and this may still allow correlation between messages.

The applicability statement may be dependent on the identity of the other endpoint, or other information carried in an EDHOC message, but it then applies only to the later phases of the protocol when such information is known. (The Initiator does not know identity of Responder before having verified message_2, and the Responder does not know identity of the Initiator before having verified message_3.)

Other conditions may be part of the applicability statement, such as target application or use (if there is more than one application/ use) to the extent that EDHOC can distinguish between them. In case multiple applicability statements are used, the receiver needs to be able to determine which is applicable for a given session, for example based on URI or external authorization data type.

4. Key Derivation

EDHOC uses Extract-and-Expand [RFC5869] with the EDHOC hash algorithm in the selected cipher suite to derive keys used in EDHOC and in the application. Extract is used to derive fixed-length uniformly pseudorandom keys (PRK) from ECDH shared secrets. Expand is used to derive additional output keying material (OKM) from the PRKs.

This section defines Extract, Expand and other key derivation functions based on these: Expand is used to define EDHOC-KDF and in turn EDHOC-Exporter, whereas Extract is used to define EDHOC-KeyUpdate.

4.1. Extract

The pseudorandom keys (PRKs) are derived using Extract.

PRK = Extract(salt, IKM)

where the input keying material (IKM) and salt are defined for each PRK below.

The definition of Extract depends on the EDHOC hash algorithm of the selected cipher suite:

  *if the EDHOC hash algorithm is SHA-2, then Extract(salt, IKM) = HKDF-Extract(salt, IKM) [RFC5869]

  *if the EDHOC hash algorithm is SHAKE128, then Extract(salt, IKM) = KMAC128(salt, IKM, 256, "")

  *if the EDHOC hash algorithm is SHAKE256, then Extract(salt, IKM) = KMAC256(salt, IKM, 512, "")

4.1.1. PRK_2e

PRK_2e is used to derive a keystream to encrypt message_2. PRK_2e is derived with the following input:

  *The salt SHALL be a zero-length byte string. Note that [RFC5869] specifies that if the salt is not provided, it is set to a string of zeros (see Section 2.2 of [RFC5869]). For implementation purposes, not providing the salt is the same as setting the salt to the zero-length byte string (0x).

  *The IKM SHALL be the ephemeral-ephemeral ECDH shared secret G_XY (calculated from G_X and Y or G_Y and X) as defined in Section 6.3.1 of [I-D.ietf-cose-rfc8152bis-algs]. The use of G_XY gives forward secrecy, in the sense that compromise of the private authentication keys does not compromise past session keys.

Example: Assuming the use of curve25519, the ECDH shared secret G_XY is the output of the X25519 function [RFC7748]:

G_XY = X25519(Y, G_X) = X25519(X, G_Y)

Example: Assuming the use of SHA-256 the extract phase of HKDF produces PRK_2e as follows:

PRK_2e = HMAC-SHA-256(salt, G_XY)

where salt = 0x (zero-length byte string).

4.1.2. PRK_3e2m

PRK_3e2m is used to produce a MAC in message_2 and to encrypt
message_3. PRK_3e2m is derived as follows:

If the Responder authenticates with a static Diffie-Hellman key,
then PRK_3e2m = Extract(PRK_2e, G_RX), where G_RX is the ECDH
shared secret calculated from G_R and X, or G_X and R, else PRK_3e2m
= PRK_2e.

4.1.3. PRK_4x3m

PRK_4x3m is used to produce a MAC in message_3, to encrypt
message_4, and to derive application specific data. PRK_4x3m is
derived as follows:

If the Initiator authenticates with a static Diffie-Hellman key,
then PRK_4x3m = Extract(PRK_3e2m, G_IY), where G_IY is the ECDH
shared secret calculated from G_I and Y, or G_Y and I, else PRK_4x3m
= PRK_3e2m.

4.2. Expand

The keys, IVs and MACs used in EDHOC are derived from the PRKs using
Expand, and instantiated with the EDHOC AEAD algorithm in the
selected cipher suite.

```
OKM = EDHOC-KDF( PRK, transcript_hash, label, context, length )
    = Expand( PRK, info, length )
```

where info is encoded as the CBOR sequence

```
info = (
  transcript_hash : bstr,
  label : tstr,
  context : bstr,
  length : uint,
)
```

where

*transcript_hash is a bstr set to one of the transcript hashes
 TH_2, TH_3, or TH_4 as defined in Sections 5.3.1, 5.4.1, and 4.3.

*label is a tstr set to the name of the derived key, IV or MAC;
 i.e., "KEYSTREAM_2", "MAC_2", "K_3", "IV_3", or "MAC_3".

*context is a bstr

*length is the length of output keying material (OKM) in bytes

The definition of Expand depends on the EDHOC hash algorithm of the selected cipher suite:

  *if the EDHOC hash algorithm is SHA-2, then Expand(PRK, info, length) = HKDF-Expand(PRK, info, length) [RFC5869]

  *if the EDHOC hash algorithm is SHAKE128, then Expand(PRK, info, length) = KMAC128(PRK, info, L, "")

  *if the EDHOC hash algorithm is SHAKE256, then Expand(PRK, info, length) = KMAC256(PRK, info, L, "")

where L = 8*length, the output length in bits.

The keys, IVs and MACs are derived as follows:

  *KEYSTREAM_2 is derived using the transcript hash TH_2 and the pseudorandom key PRK_2e.

  *MAC_2 is derived using the transcript hash TH_2 and the pseudorandom key PRK_3e2m.

  *K_3 and IV_3 are derived using the transcript hash TH_3 and the pseudorandom key PRK_3e2m. IVs are only used if the EDHOC AEAD algorithm uses IVs.

  *MAC_3 is derived using the transcript hash TH_3 and the pseudorandom key PRK_4x3m.

KEYSTREAM_2, K_3, and IV_3 use an empty CBOR byte string h'' as context. MAC_2 and MAC_3 use context as defined in Section 5.3.2 and Section 5.4.2, respectively.

4.3. EDHOC-Exporter

Application keys and other application specific data can be derived using the EDHOC-Exporter interface defined as:

EDHOC-Exporter(label, context, length)
  = EDHOC-KDF(PRK_4x3m, TH_4, label, context, length)

where label is a registered tstr from the EDHOC Exporter Label registry (Section 9.1), context is a bstr defined by the application, and length is a uint defined by the application. The (label, context) pair must be unique, i.e., a (label, context) MUST NOT be used for two different purposes. However an application can re-derive the same key several times as long as it is done in a secure way. For example, in most encryption algorithms the same key kan be reused with different nonces. The context can for example be the empty (zero-length) sequence or a single CBOR byte string.

The transcript hash TH_4 is a CBOR encoded bstr and the input to the hash function is a CBOR Sequence.

```
TH_4 = H( TH_3, CIPHERTEXT_3 )
```

where H() is the hash function in the selected cipher suite. Examples of use of the EDHOC-Exporter are given in [Section 5.5.2](#) and [Appendix A](#).

  * K_4 and IV_4 are derived with the EDHOC-Exporter using the empty CBOR byte string h'' as context, and labels "EDHOC_K_4" and "EDHOC_IV_4", respectively. IVs are only used if the EDHOC AEAD algorithm uses IVs.

4.4. EDHOC-KeyUpdate

To provide forward secrecy in an even more efficient way than re-running EDHOC, EDHOC provides the function EDHOC-KeyUpdate. When EDHOC-KeyUpdate is called the old PRK_4x3m is deleted and the new PRK_4x3m is calculated as a "hash" of the old key using the Extract function as illustrated by the following pseudocode:

```
EDHOC-KeyUpdate( nonce ):
   PRK_4x3m = Extract( nonce, PRK_4x3m )
```

The EDHOC-KeyUpdate takes a nonce as input to guarantee that there are no short cycles. The Initiator and the Responder need to agree on the nonce, which can e.g., be a counter or a random number. While the KeyUpdate method provides forward secrecy it does not give as strong security properties as re-running EDHOC, see [Section 8](#).

5. Message Formatting and Processing

This section specifies formatting of the messages and processing steps. Error messages are specified in [Section 6](#). Annotated traces of EDHOC protocol runs are provided in [[I-D.selander-lake-traces](#)].

An EDHOC message is encoded as a sequence of CBOR data items (CBOR Sequence, [[RFC8742](#)]). Additional optimizations are made to reduce message overhead.

While EDHOC uses the COSE_Key, COSE_Sign1, and COSE_Encrypt0 structures, only a subset of the parameters is included in the EDHOC messages, see [Appendix C.3](#). The unprotected COSE header in COSE_Sign1, and COSE_Encrypt0 (not included in the EDHOC message) MAY contain parameters (e.g., 'alg').

5.1. Message Processing Outline

This section outlines the message processing of EDHOC.

For each new/ongoing session, the endpoints are assumed to keep an associated protocol state containing identifiers, keying material, etc. used for subsequent processing of protocol related data. The protocol state is assumed to be associated to an applicability statement (Section 3.9) which provides the context for how messages are transported, identified, and processed.

EDHOC messages SHALL be processed according to the current protocol state. The following steps are expected to be performed at reception of an EDHOC message:

1. Detect that an EDHOC message has been received, for example by means of port number, URI, or media type (Section 3.9).

2. Retrieve the protocol state according to the message correlation provided by the transport, see Section 3.4. If there is no protocol state, in the case of message_1, a new protocol state is created. The Responder endpoint needs to make use of available Denial-of-Service mitigation (Section 8.6).

3. If the message received is an error message, then process according to Section 6, else process as the expected next message according to the protocol state.

If the processing fails for some reason then, typically, an error message is sent, the protocol is discontinued, and the protocol state erased. Further details are provided in the following subsections and in Section 6.

Different instances of the same message MUST NOT be processed in one session. Note that processing will fail if the same message appears a second time for EDHOC processing because the state of the protocol has moved on and now expects something else. This assumes that message duplication due to re-transmissions is handled by the transport protocol, see Section 3.4. The case when the transport does not support message deduplication is addressed in Appendix E.

5.2. EDHOC Message 1

5.2.1. Formatting of Message 1

message_1 SHALL be a CBOR Sequence (see Appendix C.1) as defined below

```
message_1 = (
  METHOD : int,
  SUITES_I : suites,
  G_X : bstr,
  C_I : bstr / int,
  ? EAD_1 : ead,
)

suites = [ 2* int ] / int
```

where:

*METHOD - authentication method, see Section 3.2.

*SUITES_I - array of cipher suites which the Initiator supports in
 order of preference, starting with the most preferred and ending
 with the cipher suite selected for this session. If the most
 preferred cipher suite is selected then SUITES_I is encoded as
 that cipher suite, i.e., as an int. The processing steps are
 detailed below and in Section 6.3.

*G_X - the ephemeral public key of the Initiator

*C_I - variable length connection identifier

*EAD_1 - unprotected external authorization data, see Section 3.8.

5.2.2. Initiator Processing of Message 1

The Initiator SHALL compose message_1 as follows:

*SUITES_I contains a list of supported cipher suites, in order of
 preference, truncated after the cipher suite selected for this
 session.

  -The Initiator MUST select its most preferred cipher suite,
   conditioned on what it can assume to be supported by the
   Responder.

  -The selected cipher suite MAY be changed between sessions,
   e.g., based on previous error messages (see next bullet), but
   all cipher suites which are more preferred than the selected
   cipher suite in the list MUST be included in SUITES_I.

  -If the Initiator previously received from the Responder an
   error message with error code 2 (see Section 6.3) indicating
   cipher suites supported by the Responder, then the Initiator
   SHOULD select the most preferred supported cipher suite among
   those (note that error messages are not authenticated and may
   be forged).
```

-The supported cipher suites and the order of preference MUST
 NOT be changed based on previous error messages.

*Generate an ephemeral ECDH key pair using the curve in the
 selected cipher suite and format it as a COSE_Key. Let G_X be the
 'x' parameter of the COSE_Key.

*Choose a connection identifier C_I and store it for the length of
 the protocol.

*Encode message_1 as a sequence of CBOR encoded data items as
 specified in Section 5.2.1

### 5.2.3.  Responder Processing of Message 1

The Responder SHALL process message_1 as follows:

*Decode message_1 (see Appendix C.1).

*Verify that the selected cipher suite is supported and that no
 prior cipher suite in SUITES_I is supported.

*Pass EAD_1 to the security application.

If any processing step fails, the Responder SHOULD send an EDHOC
error message back, formatted as defined in Section 6, and the
session MUST be discontinued. Sending error messages is essential
for debugging but MAY e.g., be skipped due to denial-of-service
reasons, see Section 8.6. If an error message is sent, the session
MUST be discontinued.

### 5.3.  EDHOC Message 2

### 5.3.1.  Formatting of Message 2

message_2 SHALL be a CBOR Sequence (see Appendix C.1) as defined
below

```
message_2 = (
  G_Y_CIPHERTEXT_2 : bstr,
  C_R : bstr / int,
)
```

where:

*G_Y_CIPHERTEXT_2 - the concatenation of G_Y, the ephemeral public
 key of the Responder, and CIPHERTEXT_2

*C_R - variable length connection identifier

5.3.2. Responder Processing of Message 2

The Responder SHALL compose message_2 as follows:

* Generate an ephemeral ECDH key pair using the curve in the selected cipher suite and format it as a COSE_Key. Let G_Y be the 'x' parameter of the COSE_Key.

* Choose a connection identifier C_R and store it for the length of the protocol.

* Compute the transcript hash TH_2 = H(H(message_1), G_Y, C_R) where H() is the hash function in the selected cipher suite. The transcript hash TH_2 is a CBOR encoded bstr and the input to the hash function is a CBOR Sequence. Note that H(message_1) can be computed and cached already in the processing of message_1.

* Compute MAC_2 = EDHOC-KDF(PRK_3e2m, TH_2, "MAC_2", << ID_CRED_R, CRED_R, ? EAD_2 >>, mac_length_2). If the Responder authenticates with a static Diffie-Hellman key (method equals 1 or 3), then mac_length_2 is the EDHOC MAC length given by the selected cipher suite. If the Responder authenticates with a signature key (method equals 0 or 2), then mac_length_2 is equal to the output size of the EDHOC hash algorithm given by the selected cipher suite.

  - ID_CRED_R - identifier to facilitate retrieval of CRED_R, see Section 3.5.4

  - CRED_R - CBOR item containing the credential of the Responder, see Section 3.5.3

  - EAD_2 - unprotected external authorization data, see Section 3.8

* If the Responder authenticates with a static Diffie-Hellman key (method equals 1 or 3), then Signature_or_MAC_2 is MAC_2. If the Responder authenticates with a signature key (method equals 0 or 2), then Signature_or_MAC_2 is the 'signature' field of a COSE_Sign1 object as defined in Section 4.4 of [I-D.ietf-cose-rfc8152bis-struct] using the signature algorithm of the selected cipher suite, the private authentication key of the Responder, and the following parameters as input (see Appendix C.3):

  - protected = << ID_CRED_R >>

  - external_aad = << TH_2, CRED_R, ? EAD_2 >>

  - payload = MAC_2

*CIPHERTEXT_2 is calculated by using the Expand function as a
    binary additive stream cipher.

      -plaintext = (ID_CRED_R / bstr / int, Signature_or_MAC_2, ?
       EAD_2)

         oIf ID_CRED_R contains a single 'kid' parameter, i.e.,
          ID_CRED_R = { 4 : kid_R }, then only the byte string or
          integer kid_R is conveyed in the plaintext encoded
          accordingly as bstr or int.

      -Compute KEYSTREAM_2 = EDHOC-KDF(PRK_2e, TH_2, "KEYSTREAM_2",
       h'', plaintext_length), where plaintext_length is the length
       of the plaintext.

      -CIPHERTEXT_2 = plaintext XOR KEYSTREAM_2

   *Encode message_2 as a sequence of CBOR encoded data items as
    specified in Section 5.3.1.

5.3.3. Initiator Processing of Message 2

   The Initiator SHALL process message_2 as follows:

   *Decode message_2 (see Appendix C.1).

   *Retrieve the protocol state using the message correlation
    provided by the transport (e.g., the CoAP Token, the 5-tuple, or
    the prepended C_I, see Appendix A.3).

   *Decrypt CIPHERTEXT_2, see Section 5.3.2.

   *Pass EAD_2 to the security application.

   *Verify that the identity of the Responder is an allowed identity
    for this connection, see Section 3.5.1.

   *Verify Signature_or_MAC_2 using the algorithm in the selected
    cipher suite. The verification process depends on the method, see
    Section 5.3.2.

   If any processing step fails, the Initiator SHOULD send an EDHOC
   error message back, formatted as defined in Section 6. Sending error
   messages is essential for debugging but MAY e.g., be skipped if a
   session cannot be found or due to denial-of-service reasons, see
   Section 8.6. If an error message is sent, the session MUST be
   discontinued.

5.4. EDHOC Message 3

5.4.1. Formatting of Message 3

message_3 SHALL be a CBOR Sequence (see [Appendix C.1](#)) as defined below

```
message_3 = (
  CIPHERTEXT_3 : bstr,
)
```

5.4.2. Initiator Processing of Message 3

The Initiator SHALL compose message_3 as follows:

*Compute the transcript hash TH_3 = H(TH_2, CIPHERTEXT_2) where H() is the hash function in the selected cipher suite. The transcript hash TH_3 is a CBOR encoded bstr and the input to the hash function is a CBOR Sequence. Note that H(TH_2, CIPHERTEXT_2) can be computed and cached already in the processing of message_2.

*Compute MAC_3 = EDHOC-KDF(PRK_4x3m, TH_3, "MAC_3", << ID_CRED_I, CRED_I, ? EAD_3 >>, mac_length_3). If the Initiator authenticates with a static Diffie-Hellman key (method equals 2 or 3), then mac_length_3 is the EDHOC MAC length given by the selected cipher suite. If the Initiator authenticates with a signature key (method equals 0 or 1), then mac_length_3 is equal to the output size of the EDHOC hash algorithm given by the selected cipher suite.

  -ID_CRED_I - identifier to facilitate retrieval of CRED_I, see [Section 3.5.4](#)

  -CRED_I - CBOR item containing the credential of the Initiator, see [Section 3.5.3](#)

  -EAD_3 - protected external authorization data, see [Section 3.8](#)

*If the Initiator authenticates with a static Diffie-Hellman key (method equals 2 or 3), then Signature_or_MAC_3 is MAC_3. If the Initiator authenticates with a signature key (method equals 0 or 1), then Signature_or_MAC_3 is the 'signature' field of a COSE_Sign1 object as defined in Section 4.4 of [[I-D.ietf-cose-rfc8152bis-struct](#)] using the signature algorithm of the selected cipher suite, the private authentication key of the Initiator, and the following parameters as input (see [Appendix C.3](#)):

  -protected = << ID_CRED_I >>

```
    -external_aad = << TH_3, CRED_I, ? EAD_3 >>

    -payload = MAC_3
```

  *Compute a COSE_Encrypt0 object as defined in Section 5.3 of [I-
   D.ietf-cose-rfc8152bis-struct], with the EDHOC AEAD algorithm of
   the selected cipher suite, using the encryption key K_3, the
   initialization vector IV_3, the plaintext P, and the following
   parameters as input (see Appendix C.3):

```
    -protected = h''

    -external_aad = TH_3
```

  where

```
    -K_3 = EDHOC-KDF( PRK_3e2m, TH_3, "K_3", h'', key_length )

       okey_length - length of the encryption key of the EDHOC AEAD
        algorithm

    -IV_3 = EDHOC-KDF( PRK_3e2m, TH_3, "IV_3", h'', iv_length )

       oiv_length - length of the intialization vector of the EDHOC
        AEAD algorithm

    -P = ( ID_CRED_I / bstr / int, Signature_or_MAC_3, ? EAD_3 )

       oIf ID_CRED_I contains a single 'kid' parameter, i.e.,
        ID_CRED_I = { 4 : kid_I }, only the byte string or integer
        kid_I is conveyed in the plaintext encoded accordingly as
        bstr or int.
```

   CIPHERTEXT_3 is the 'ciphertext' of COSE_Encrypt0.

  *Encode message_3 as a CBOR data item as specified in Section
   5.4.1.

Pass the connection identifiers (C_I, C_R) and the application
algorithms in the selected cipher suite to the application. The
application can now derive application keys using the EDHOC-Exporter
interface, see Section 4.3.

After sending message_3, the Initiator is assured that no other
party than the Responder can compute the key PRK_4x3m (implicit key
authentication). The Initiator can securely derive application keys
and send protected application data. However, the Initiator does not
know that the Responder has actually computed the key PRK_4x3m and
therefore the Initiator SHOULD NOT permanently store the keying
material PRK_4x3m and TH_4, or derived application keys, until the

Initiator is assured that the Responder has actually computed the key PRK_4x3m (explicit key confirmation). This is similar to waiting for acknowledgement (ACK) in a transport protocol. Explicit key confirmation is e.g., assured when the Initiator has verified an OSCORE message or message_4 from the Responder.

5.4.3. Responder Processing of Message 3

The Responder SHALL process message_3 as follows:

  *Decode message_3 (see Appendix C.1).

  *Retrieve the protocol state using the message correlation provided by the transport (e.g., the CoAP Token, the 5-tuple, or the prepended C_I, see Appendix A.3).

  *Decrypt and verify the COSE_Encrypt0 as defined in Section 5.3 of [I-D.ietf-cose-rfc8152bis-struct], with the EDHOC AEAD algorithm in the selected cipher suite, and the parameters defined in Section 5.4.2.

  *Pass EAD_3 to the security application.

  *Verify that the identity of the Initiator is an allowed identity for this connection, see Section 3.5.1.

  *Verify Signature_or_MAC_3 using the algorithm in the selected cipher suite. The verification process depends on the method, see Section 5.4.2.

  *Pass the connection identifiers (C_I, C_R), and the application algorithms in the selected cipher suite to the security application. The application can now derive application keys using the EDHOC-Exporter interface.

If any processing step fails, the Responder SHOULD send an EDHOC error message back, formatted as defined in Section 6. Sending error messages is essential for debugging but MAY e.g., be skipped if a session cannot be found or due to denial-of-service reasons, see Section 8.6. If an error message is sent, the session MUST be discontinued.

After verifying message_3, the Responder is assured that the Initiator has calculated the key PRK_4x3m (explicit key confirmation) and that no other party than the Responder can compute the key. The Responder can securely send protected application data and store the keying material PRK_4x3m and TH_4.

5.5. EDHOC Message 4

This section specifies message_4 which is OPTIONAL to support. Key
confirmation is normally provided by sending an application message
from the Responder to the Initiator protected with a key derived
with the EDHOC-Exporter, e.g., using OSCORE (see Appendix A). In
deployments where no protected application message is sent from the
Responder to the Initiator, the Responder MUST send message_4. Two
examples of such deployments:

1. When EDHOC is only used for authentication and no application
   data is sent.

2. When application data is only sent from the Initiator to the
   Responder.

Further considerations about when to use message_4 are provided in
Section 3.9 and Section 8.1.

5.5.1. Formatting of Message 4

message_4 SHALL be a CBOR Sequence (see Appendix C.1) as defined
below

```
message_4 = (
  CIPHERTEXT_4 : bstr,
)
```

5.5.2. Responder Processing of Message 4

The Responder SHALL compose message_4 as follows:

  *Compute a COSE_Encrypt0 as defined in Section 5.3 of [I-D.ietf-
   cose-rfc8152bis-struct], with the EDHOC AEAD algorithm of the
   selected cipher suite, using the encryption key K_4, the
   initialization vector IV_4, the plaintext P, and the following
   parameters as input (see Appendix C.3):

     -protected = h''

     -external_aad = TH_4

   where

     -K_4 = EDHOC-Exporter("EDHOC_K_4", h'', key_length)

        okey_length - length of the encryption key of the EDHOC AEAD
         algorithm

```
      -IV_4 = EDHOC-Exporter( "EDHOC_IV_4", h'', iv_length )

        oiv_length - length of the intialization vector of the EDHOC
         AEAD algorithm

      -P = ( ? EAD_4 )

        oEAD_4 - protected external authorization data, see Section
         3.8.
```

   CIPHERTEXT_4 is the 'ciphertext' of COSE_Encrypt0.

  *Encode message_4 as a CBOR data item as specified in Section
   5.5.1.

5.5.3. Initiator Processing of Message 4

   The Initiator SHALL process message_4 as follows:

  *Decode message_4 (see Appendix C.1).

  *Retrieve the protocol state using the message correlation
   provided by the transport (e.g., the CoAP Token, the 5-tuple, or
   the prepended C_I, see Appendix A.3).

  *Decrypt and verify the COSE_Encrypt0 as defined in Section 5.3 of
   [I-D.ietf-cose-rfc8152bis-struct], with the EDHOC AEAD algorithm
   in the selected cipher suite, and the parameters defined in
   Section 5.5.2.

  *Pass EAD_4 to the security application.

   If any processing step fails, the Responder SHOULD send an EDHOC
   error message back, formatted as defined in Section 6. Sending error
   messages is essential for debugging but MAY e.g., be skipped if a
   session cannot be found or due to denial-of-service reasons, see
   Section 8.6. If an error message is sent, the session MUST be
   discontinued.

6. Error Handling

   This section defines the format for error messages.

   An EDHOC error message can be sent by either endpoint as a reply to
   any non-error EDHOC message. How errors at the EDHOC layer are
   transported depends on lower layers, which need to enable error
   messages to be sent and processed as intended.

   Errors in EDHOC are fatal. After sending an error message, the
   sender MUST discontinue the protocol. The receiver SHOULD treat an

error message as an indication that the other party likely has
discontinued the protocol. But as the error message is not
authenticated, a received error message might also have been sent by
an attacker and the receiver MAY therefore try to continue the
protocol.

error SHALL be a CBOR Sequence (see [Appendix C.1](#)) as defined below

```
error = (
  ERR_CODE : int,
  ERR_INFO : any,
)
```

Figure 6: EDHOC Error Message

where:

*ERR_CODE - error code encoded as an integer. The value 0 is used
for success, all other values (negative or positive) indicate
errors.

*ERR_INFO - error information. Content and encoding depend on
error code.

The remainder of this section specifies the currently defined error
codes, see [Figure 7](#). Additional error codes and corresponding error
information may be specified.

+----------+---------------+--+
| ERR_CODE | ERR_INFO Type | Description |
+==========+===============+==+
| 0 | any | Success |
+----------+---------------+--+
| 1 | tstr | Unspecified |
+----------+---------------+--+
| 2 | suites | Wrong selected cipher suite |
+----------+---------------+--+

Figure 7: Error Codes and Error Information

6.1. Success

Error code 0 MAY be used internally in an application to indicate
success, e.g., in log files. ERR_INFO can contain any type of CBOR
item. Error code 0 MUST NOT be used as part of the EDHOC message
exchange flow.

6.2. Unspecified

Error code 1 is used for errors that do not have a specific error code defined. ERR_INFO MUST be a text string containing a human-readable diagnostic message written in English. The diagnostic text message is mainly intended for software engineers that during debugging need to interpret it in the context of the EDHOC specification. The diagnostic message SHOULD be provided to the calling application where it SHOULD be logged.

6.3. Wrong Selected Cipher Suite

Error code 2 MUST only be used in a response to message_1 in case the cipher suite selected by the Initiator is not supported by the Responder, or if the Responder supports a cipher suite more preferred by the Initiator than the selected cipher suite, see Section 5.2.3. ERR_INFO is in this case denoted SUITES_R and is of type suites, see Section 5.2.1. If the Responder does not support the selected cipher suite, then SUITES_R MUST include one or more supported cipher suites. If the Responder supports a cipher suite in SUITES_I other than the selected cipher suite (independently of if the selected cipher suite is supported or not) then SUITES_R MUST include the supported cipher suite in SUITES_I which is most preferred by the Initiator. SUITES_R MAY include a single cipher suite, i.e., be encoded as an int. If the Responder does not support any cipher suite in SUITES_I, then it SHOULD include all its supported cipher suites in SUITES_R in any order.

6.3.1. Cipher Suite Negotiation

After receiving SUITES_R, the Initiator can determine which cipher suite to select (if any) for the next EDHOC run with the Responder.

If the Initiator intends to contact the Responder in the future, the Initiator SHOULD remember which selected cipher suite to use until the next message_1 has been sent, otherwise the Initiator and Responder will likely run into an infinite loop where the Initiator selects its most preferred and the Responder sends an error with supported cipher suites. After a successful run of EDHOC, the Initiator MAY remember the selected cipher suite to use in future EDHOC sessions. Note that if the Initiator or Responder is updated with new cipher suite policies, any cached information may be outdated.

6.3.2. Examples

Assume that the Initiator supports the five cipher suites 5, 6, 7, 8, and 9 in decreasing order of preference. Figures 8 and 9 show examples of how the Initiator can format SUITES_I and how SUITES_R

is used by Responders to give the Initiator information about the
cipher suites that the Responder supports.

In the first example (Figure 8), the Responder supports cipher suite
6 but not the initially selected cipher suite 5.

```
Initiator                                                       Responder
|                METHOD, SUITES_I = 5, G_X, C_I, EAD_1                  |
+--------------------------------------------------------------------->|
|                             message_1                                |
|                                                                      |
|                    ERR_CODE = 2, SUITES_R = 6                        |
|<--------------------------------------------------------------------+
|                              error                                   |
|                                                                      |
|             METHOD, SUITES_I = [5, 6], G_X, C_I, EAD_1               |
+--------------------------------------------------------------------->|
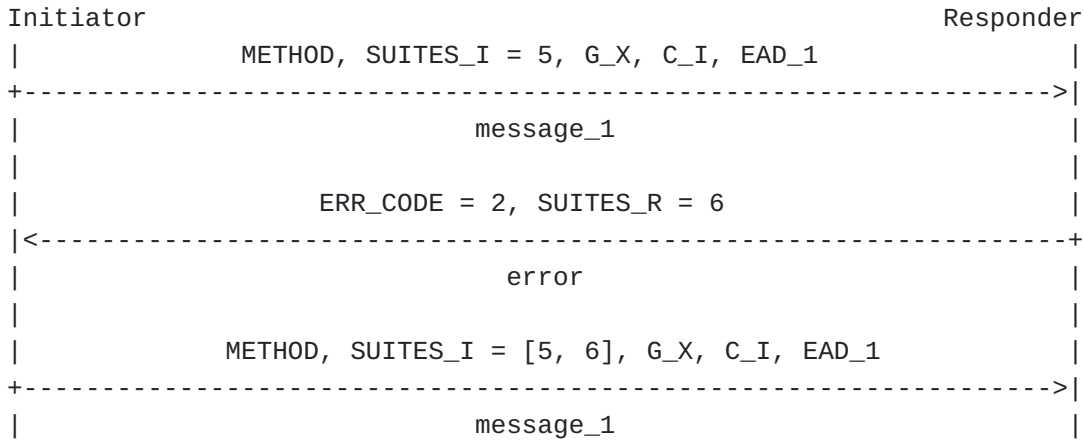|                             message_1                                |
```

Figure 8: Example of Responder supporting suite 6 but not suite 5.

In the second example (Figure 9), the Responder supports cipher
suites 8 and 9 but not the more preferred (by the Initiator) cipher
suites 5, 6 or 7. To illustrate the negotiation mechanics we let the
Initiator first make a guess that the Responder supports suite 6 but
not suite 5. Since the Responder supports neither 5 nor 6, it
responds with SUITES_R containing the supported suites, after which
the Initiator selects its most preferred supported suite. The order
of cipher suites in SUITES_R does not matter. (If the Responder had
supported suite 5, it would have included it in SUITES_R of the
response, and it would in that case have become the selected suite
in the second message_1.)

```
Initiator                                                       Responder
|             METHOD, SUITES_I = [5, 6], G_X, C_I, EAD_1               |
+--------------------------------------------------------------------->|
|                             message_1                                |
|                                                                      |
|                  ERR_CODE = 2, SUITES_R = [9, 8]                    |
|<--------------------------------------------------------------------+
|                              error                                   |
|                                                                      |
|          METHOD, SUITES_I = [5, 6, 7, 8], G_X, C_I, EAD_1           |
+--------------------------------------------------------------------->|
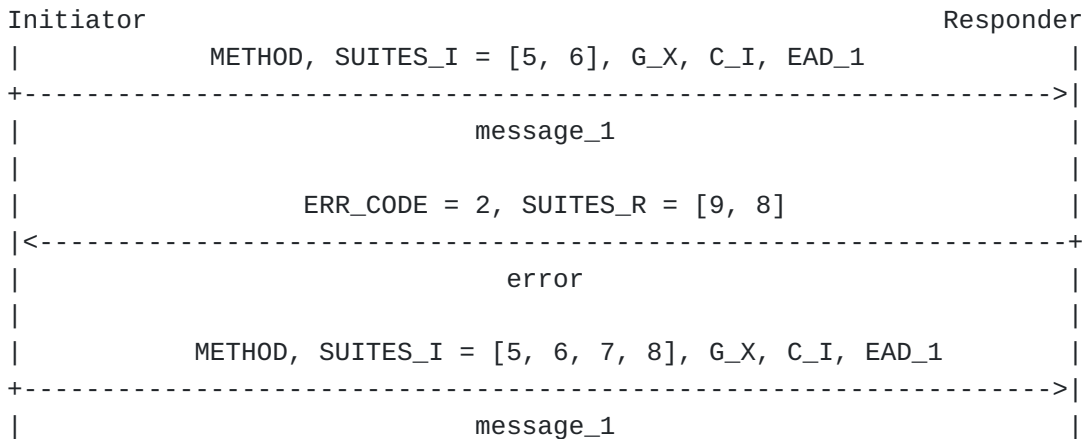|                             message_1                                |
```

Figure 9: Example of Responder supporting suites 8 and 9 but not 5, 6
                                or 7.

Note that the Initiator's list of supported cipher suites and order
of preference is fixed (see [Section 5.2.1](#) and [Section 5.2.2](#)).
Furthermore, the Responder shall only accept message_1 if the
selected cipher suite is the first cipher suite in SUITES_I that the
Responder supports (see [Section 5.2.3](#)). Following this procedure
ensures that the selected cipher suite is the most preferred (by the
Initiator) cipher suite supported by both parties.

If the selected cipher suite is not the first cipher suite which the
Responder supports in SUITES_I received in message_1, then Responder
MUST discontinue the protocol, see [Section 5.2.3](#). If SUITES_I in
message_1 is manipulated, then the integrity verification of
message_2 containing the transcript hash TH_2 will fail and the
Initiator will discontinue the protocol.

7. Mandatory-to-Implement Compliance Requirements

An implementation may support only Initiator or only Responder.

An implementation may support only a single method. None of the
methods are mandatory-to-implement.

Implementations MUST support 'kid' parameters of type int. None of
the other COSE header parameters are mandatory-to-implement.

An implementation may support only a single credential type (CCS,
CWT, X.509, C509). None of the credential types are mandatory-to-
implement.

Implementations MUST support the EDHOC-Exporter. Implementations
SHOULD support EDHOC-KeyUpdate.

Implementations MAY support message_4. Error codes 1 and 2 MUST be
supported.

Implementations MAY support EAD.

For many constrained IoT devices it is problematic to support more
than one cipher suite. Existing devices can be expected to support
either ECDSA or EdDSA. To enable as much interoperability as we can
reasonably achieve, less constrained devices SHOULD implement both
cipher suite 0 (AES-CCM-16-64-128, SHA-256, 8, X25519, EdDSA, AES-
CCM-16-64-128, SHA-256) and cipher suite 2 (AES-CCM-16-64-128,
SHA-256, 8, P-256, ES256, AES-CCM-16-64-128, SHA-256). Constrained
endpoints SHOULD implement cipher suite 0 or cipher suite 2.
Implementations only need to implement the algorithms needed for
their supported methods.

8. Security Considerations

8.1. Security Properties

EDHOC inherits its security properties from the theoretical SIGMA-I protocol [SIGMA]. Using the terminology from [SIGMA], EDHOC provides forward secrecy, mutual authentication with aliveness, consistency, and peer awareness. As described in [SIGMA], peer awareness is provided to the Responder, but not to the Initiator.

As described in [SIGMA], different levels of identity protection is provided to the Initiator and the Responder. EDHOC protects the credential identifier of the Initiator against active attacks and the credential identifier of the Responder against passive attacks. The roles should be assigned to protect the most sensitive identity/ identifier, typically that which is not possible to infer from routing information in the lower layers.

Compared to [SIGMA], EDHOC adds an explicit method type and expands the message authentication coverage to additional elements such as algorithms, external authorization data, and previous messages. This protects against an attacker replaying messages or injecting messages from another session.

EDHOC also adds selection of connection identifiers and downgrade protected negotiation of cryptographic parameters, i.e., an attacker cannot affect the negotiated parameters. A single session of EDHOC does not include negotiation of cipher suites, but it enables the Responder to verify that the selected cipher suite is the most preferred cipher suite by the Initiator which is supported by both the Initiator and the Responder.

As required by [RFC7258], IETF protocols need to mitigate pervasive monitoring when possible. EDHOC therefore only supports methods with ephemeral Diffie-Hellman and provides a KeyUpdate function for lightweight application protocol rekeying with forward secrecy, in the sense that compromise of the private authentication keys does not compromise past session keys, and compromise of a session key does not compromise past session keys.

While the KeyUpdate method can be used to meet cryptographic limits and provide partial protection against key leakage, it provides significantly weaker security properties than re-running EDHOC with ephemeral Diffie-Hellman. Even with frequent use of KeyUpdate, compromise of one session key compromises all future session keys, and an attacker therefore only needs to perform static key exfiltration [RFC7624]. Frequently re-running EDHOC with ephemeral Diffie-Hellman forces attackers to perform dynamic key exfiltration instead of static key exfiltration [RFC7624]. In the dynamic case,

the attacker must have continuous interactions with the collaborator, which is more complicated and has a higher risk profile than the static case.

To limit the effect of breaches, it is important to limit the use of symmetrical group keys for bootstrapping. EDHOC therefore strives to make the additional cost of using raw public keys and self-signed certificates as small as possible. Raw public keys and self-signed certificates are not a replacement for a public key infrastructure but SHOULD be used instead of symmetrical group keys for bootstrapping.

Compromise of the long-term keys (private signature or static DH keys) does not compromise the security of completed EDHOC exchanges. Compromising the private authentication keys of one party lets an active attacker impersonate that compromised party in EDHOC exchanges with other parties but does not let the attacker impersonate other parties in EDHOC exchanges with the compromised party. Compromise of the long-term keys does not enable a passive attacker to compromise future session keys. Compromise of the HDKF input parameters (ECDH shared secret) leads to compromise of all session keys derived from that compromised shared secret. Compromise of one session key does not compromise other session keys. Compromise of PRK_4x3m leads to compromise of all exported keying material derived after the last invocation of the EDHOC-KeyUpdate function.

EDHOC provides a minimum of 64-bit security against online brute force attacks and a minimum of 128-bit security against offline brute force attacks. This is in line with IPsec, TLS, and COSE. To break 64-bit security against online brute force an attacker would on average have to send 4.3 billion messages per second for 68 years, which is infeasible in constrained IoT radio technologies.

After sending message_3, the Initiator is assured that no other party than the Responder can compute the key PRK_4x3m (implicit key authentication). The Initiator does however not know that the Responder has actually computed the key PRK_4x3m. While the Initiator can securely send protected application data, the Initiator SHOULD NOT permanently store the keying material PRK_4x3m and TH_4 until the Initiator is assured that the Responder has actually computed the key PRK_4x3m (explicit key confirmation). Explicit key confirmation is e.g., assured when the Initiator has verified an OSCORE message or message_4 from the Responder. After verifying message_3, the Responder is assured that the Initiator has calculated the key PRK_4x3m (explicit key confirmation) and that no other party than the Responder can compute the key. The Responder can securely send protected application data and store the keying material PRK_4x3m and TH_4.

Key compromise impersonation (KCI): In EDHOC authenticated with signature keys, EDHOC provides KCI protection against an attacker having access to the long-term key or the ephemeral secret key. With static Diffie-Hellman key authentication, KCI protection would be provided against an attacker having access to the long-term Diffie-Hellman key, but not to an attacker having access to the ephemeral secret key. Note that the term KCI has typically been used for compromise of long-term keys, and that an attacker with access to the ephemeral secret key can only attack that specific session.

Repudiation: In EDHOC authenticated with signature keys, the Initiator could theoretically prove that the Responder performed a run of the protocol by presenting the private ephemeral key, and vice versa. Note that storing the private ephemeral keys violates the protocol requirements. With static Diffie-Hellman key authentication, both parties can always deny having participated in the protocol.

Two earlier versions of EDHOC have been formally analyzed [Norrman20] [Bruni18] and the specification has been updated based on the analysis.

8.2. Cryptographic Considerations

The SIGMA protocol requires that the encryption of message_3 provides confidentiality against active attackers and EDHOC message_4 relies on the use of authenticated encryption. Hence the message authenticating functionality of the authenticated encryption in EDHOC is critical: authenticated encryption MUST NOT be replaced by plain encryption only, even if authentication is provided at another level or through a different mechanism.

To reduce message overhead EDHOC does not use explicit nonces and instead rely on the ephemeral public keys to provide randomness to each session. A good amount of randomness is important for the key generation, to provide liveness, and to protect against interleaving attacks. For this reason, the ephemeral keys MUST NOT be used in more than one EDHOC message, and both parties SHALL generate fresh random ephemeral key pairs. Note that an ephemeral key may be used to calculate several ECDH shared secrets. When static Diffie-Hellman authentication is used the same ephemeral key is used in both ephemeral-ephemeral and ephemeral-static ECDH.

As discussed in [SIGMA], the encryption of message_2 does only need to protect against passive attacker as active attackers can always get the Responders identity by sending their own message_1. EDHOC uses the Expand function (typically HKDF-Expand) as a binary additive stream cipher. HKDF-Expand provides better confidentiality than AES-CTR but is not often used as it is slow on long messages,

and most applications require both IND-CCA confidentiality as well as integrity protection. For the encryption of message_2, any speed difference is negligible, IND-CCA does not increase security, and integrity is provided by the inner MAC (and signature depending on method).

Requirement for how to securely generate, validate, and process the ephemeral public keys depend on the elliptic curve. For X25519 and X448, the requirements are defined in [RFC7748]. For secp256r1, secp384r1, and secp521r1, the requirements are defined in Section 5 of [SP-800-56A]. For secp256r1, secp384r1, and secp521r1, at least partial public-key validation MUST be done.

8.3. Cipher Suites and Cryptographic Algorithms

When using private cipher suite or registering new cipher suites, the choice of key length used in the different algorithms needs to be harmonized, so that a sufficient security level is maintained for certificates, EDHOC, and the protection of application data. The Initiator and the Responder should enforce a minimum security level.

The hash algorithms SHA-1 and SHA-256/64 (SHA-256 truncated to 64-bits) SHALL NOT be supported for use in EDHOC except for certificate identification with x5t and c5t. Note that secp256k1 is only defined for use with ECDSA and not for ECDH. Note that some COSE algorithms are marked as not recommended in the COSE IANA registry.

8.4. Post-Quantum Considerations

As of the publication of this specification, it is unclear when or even if a quantum computer of sufficient size and power to exploit public key cryptography will exist. Deployments that need to consider risks decades into the future should transition to Post-Quantum Cryptography (PQC) in the not-too-distant future. Many other systems should take a slower wait-and-see approach where PQC is phased in when the quantum threat is more imminent. Current PQC algorithms have limitations compared to Elliptic Curve Cryptography (ECC) and the data sizes would be problematic in many constrained IoT systems.

Symmetric algorithms used in EDHOC such as SHA-256 and AES-CCM-16-64-128 are practically secure against even large quantum computers. EDHOC supports all signature algorithms defined by COSE, including PQC signature algorithms such as HSS-LMS. EDHOC is currently only specified for use with key exchange algorithms of type ECDH curves, but any Key Encapsulation Method (KEM), including PQC KEMs, can be used in method 0. While the key exchange in method 0 is specified with terms of the Diffie-Hellman protocol, the key exchange adheres to a KEM interface: G_X is then the public key of

the Initiator, G_Y is the encapsulation, and G_XY is the shared secret. Use of PQC KEMs to replace static DH authentication would likely require a specification updating EDHOC with new methods.

8.5. Unprotected Data

The Initiator and the Responder must make sure that unprotected data and metadata do not reveal any sensitive information. This also applies for encrypted data sent to an unauthenticated party. In particular, it applies to EAD_1, ID_CRED_R, EAD_2, and error messages. Using the same EAD_1 in several EDHOC sessions allows passive eavesdroppers to correlate the different sessions. Another consideration is that the list of supported cipher suites may potentially be used to identify the application.

The Initiator and the Responder must also make sure that unauthenticated data does not trigger any harmful actions. In particular, this applies to EAD_1 and error messages.

8.6. Denial-of-Service

As CoAP provides Denial-of-Service protection in the form of the Echo option [RFC9175], EDHOC itself does not provide countermeasures against Denial-of-Service attacks. By sending a number of new or replayed message_1 an attacker may cause the Responder to allocate state, perform cryptographic operations, and amplify messages. To mitigate such attacks, an implementation SHOULD rely on lower layer mechanisms such as the Echo option in CoAP that forces the initiator to demonstrate reachability at its apparent network address.

An attacker can also send faked message_2, message_3, message_4, or error in an attempt to trick the receiving party to send an error message and discontinue the session. EDHOC implementations MAY evaluate if a received message is likely to have been forged by an attacker and ignore it without sending an error message or discontinuing the session.

8.7. Implementation Considerations

The availability of a secure random number generator is essential for the security of EDHOC. If no true random number generator is available, a truly random seed MUST be provided from an external source and used with a cryptographically secure pseudorandom number generator. As each pseudorandom number must only be used once, an implementation needs to get a new truly random seed after reboot, or continuously store state in nonvolatile memory, see ([RFC8613], Appendix B.1.1) for issues and solution approaches for writing to nonvolatile memory. Intentionally or unintentionally weak or predictable pseudorandom number generators can be abused or exploited for malicious purposes. [RFC8937] describes a way for

security protocol implementations to augment their (pseudo)random number generators using a long-term private key and a deterministic signature function. This improves randomness from broken or otherwise subverted random number generators. The same idea can be used with other secrets and functions such as a Diffie-Hellman function or a symmetric secret and a PRF like HMAC or KMAC. It is RECOMMENDED to not trust a single source of randomness and to not put unaugmented random numbers on the wire.

If ECDSA is supported, "deterministic ECDSA" as specified in [RFC6979] MAY be used. Pure deterministic elliptic-curve signatures such as deterministic ECDSA and EdDSA have gained popularity over randomized ECDSA as their security do not depend on a source of high-quality randomness. Recent research has however found that implementations of these signature algorithms may be vulnerable to certain side-channel and fault injection attacks due to their determinism. See e.g., Section 1 of [I-D.mattsson-cfrg-det-sigs-with-noise] for a list of attack papers. As suggested in Section 6.1.2 of [I-D.ietf-cose-rfc8152bis-algs] this can be addressed by combining randomness and determinism.

All private keys, symmetric keys, and IVs MUST be secret. Implementations should provide countermeasures to side-channel attacks such as timing attacks. Intermediate computed values such as ephemeral ECDH keys and ECDH shared secrets MUST be deleted after key derivation is completed.

The Initiator and the Responder are responsible for verifying the integrity of certificates. The selection of trusted CAs should be done very carefully and certificate revocation should be supported. The private authentication keys MUST be kept secret, only the Responder SHALL have access to the Responder's private authentication key and only the Initiator SHALL have access to the Initiator's private authentication key.

The Initiator and the Responder are allowed to select the connection identifiers C_I and C_R, respectively, for the other party to use in the ongoing EDHOC protocol as well as in a subsequent application protocol (e.g., OSCORE [RFC8613]). The choice of connection identifier is not security critical in EDHOC but intended to simplify the retrieval of the right security context in combination with using short identifiers. If the wrong connection identifier of the other party is used in a protocol message it will result in the receiving party not being able to retrieve a security context (which will terminate the protocol) or retrieve the wrong security context (which also terminates the protocol as the message cannot be verified).

If two nodes unintentionally initiate two simultaneous EDHOC message
exchanges with each other even if they only want to complete a
single EDHOC message exchange, they MAY terminate the exchange with
the lexicographically smallest G_X. If the two G_X values are equal,
the received message_1 MUST be discarded to mitigate reflection
attacks. Note that in the case of two simultaneous EDHOC exchanges
where the nodes only complete one and where the nodes have different
preferred cipher suites, an attacker can affect which of the two
nodes' preferred cipher suites will be used by blocking the other
exchange.

If supported by the device, it is RECOMMENDED that at least the
long-term private keys are stored in a Trusted Execution Environment
(TEE) and that sensitive operations using these keys are performed
inside the TEE. To achieve even higher security it is RECOMMENDED
that additional operations such as ephemeral key generation, all
computations of shared secrets, and storage of the PRK keys can be
done inside the TEE. The use of a TEE enforces that code within that
environment cannot be tampered with, and that any data used by such
code cannot be read or tampered with by code outside that
environment.

9. IANA Considerations

9.1. EDHOC Exporter Label Registry

IANA has created a new registry titled "EDHOC Exporter Label" under
the new group name "Ephemeral Diffie-Hellman Over COSE (EDHOC)". The
registration procedure is "Expert Review". The columns of the
registry are Label, Description, and Reference. All columns are text
strings where Label consists only of the printable ASCII characters
0x21 - 0x7e. Labels beginning with "PRIVATE" MAY be used for private
use without registration. All other label values MUST be registered.
The initial contents of the registry are:

Label: EDHOC_K_4
Description: Key used to protect EDHOC message_4
Reference: [[this document]]

Label: EDHOC_IV_4
Description: IV used to protect EDHOC message_4
Reference: [[this document]]

Label: OSCORE_Master_Secret
Description: Derived OSCORE Master Secret
Reference: [[this document]]

Label: OSCORE_Master_Salt
Description: Derived OSCORE Master Salt
Reference: [[this document]]

9.2. EDHOC Cipher Suites Registry

IANA has created a new registry titled "EDHOC Cipher Suites" under
the new group name "Ephemeral Diffie-Hellman Over COSE (EDHOC)". The
registration procedure is "Expert Review". The columns of the
registry are Value, Array, Description, and Reference, where Value
is an integer and the other columns are text strings. The initial
contents of the registry are:

```
Value: -24
Algorithms: N/A
Desc: Reserved for Private Use
Reference: [[this document]]

Value: -23
Algorithms: N/A
Desc: Reserved for Private Use
Reference: [[this document]]

Value: -22
Algorithms: N/A
Desc: Reserved for Private Use
Reference: [[this document]]

Value: -21
Algorithms: N/A
Desc: Reserved for Private Use
Reference: [[this document]]

Value: 0
Array: 10, -16, 8, 4, -8, 10, -16
Desc: AES-CCM-16-64-128, SHA-256, 8, X25519, EdDSA,
      AES-CCM-16-64-128, SHA-256
Reference: [[this document]]

Value: 1
Array: 30, -16, 16, 4, -8, 10, -16
Desc: AES-CCM-16-128-128, SHA-256, 16, X25519, EdDSA,
      AES-CCM-16-64-128, SHA-256
Reference: [[this document]]

Value: 2
Array: 10, -16, 8, 1, -7, 10, -16
Desc: AES-CCM-16-64-128, SHA-256, 8, P-256, ES256,
      AES-CCM-16-64-128, SHA-256
Reference: [[this document]]
```

```
Value: 3
Array: 30, -16, 16, 1, -7, 10, -16
Desc: AES-CCM-16-128-128, SHA-256, 16, P-256, ES256,
      AES-CCM-16-64-128, SHA-256
Reference: [[this document]]


Value: 4
Array: 24, -16, 16, 4, -8, 24, -16
Desc: ChaCha20/Poly1305, SHA-256, 16, X25519, EdDSA,
      ChaCha20/Poly1305, SHA-256
Reference: [[this document]]


Value: 5
Array: 24, -16, 16, 1, -7, 24, -16
Desc: ChaCha20/Poly1305, SHA-256, 16, P-256, ES256,
      ChaCha20/Poly1305, SHA-256
Reference: [[this document]]


Value: 6
Array: 1, -16, 16, 4, -7, 1, -16
Desc: A128GCM, SHA-256, 16, X25519, ES256,
      A128GCM, SHA-256
Reference: [[this document]]


Value: 24
Array: 3, -43, 16, 2, -35, 3, -43
Desc: A256GCM, SHA-384, 16, P-384, ES384,
      A256GCM, SHA-384
Reference: [[this document]]


Value: 25
Array: 24, -45, 16, 5, -8, 24, -45
Desc: ChaCha20/Poly1305, SHAKE256, 16, X448, EdDSA,
      ChaCha20/Poly1305, SHAKE256
Reference: [[this document]]
```

9.3. EDHOC Method Type Registry

   IANA has created a new registry entitled "EDHOC Method Type" under
   the new group name "Ephemeral Diffie-Hellman Over COSE (EDHOC)". The
   registration procedure is "Expert Review". The columns of the
   registry are Value, Description, and Reference, where Value is an
   integer and the other columns are text strings. The initial contents
   of the registry are shown in Figure 4.

9.4. EDHOC Error Codes Registry

   IANA has created a new registry entitled "EDHOC Error Codes" under
   the new group name "Ephemeral Diffie-Hellman Over COSE (EDHOC)". The
   registration procedure is "Expert Review". The columns of the

registry are ERR_CODE, ERR_INFO Type and Description, where ERR_CODE
is an integer, ERR_INFO is a CDDL defined type, and Description is a
text string. The initial contents of the registry are shown in
Figure 7.

9.5. EDHOC External Authorization Data Registry

IANA has created a new registry entitled "EDHOC External
Authorization Data" under the new group name "Ephemeral Diffie-
Hellman Over COSE (EDHOC)". The registration procedure is "Expert
Review". The columns of the registry are Label, Description, Value
Type, and Reference, where Label is an integer and the other columns
are text strings.

9.6. COSE Header Parameters Registry

IANA has registered the following entries in the "COSE Header
Parameters" registry under the group name "CBOR Object Signing and
Encryption (COSE)". The value of the 'kcwt' header parameter is a
COSE Web Token (CWT) [RFC8392], and the value of the 'kccs' header
parameter is an CWT Claims Set (CCS), see Section 1.5. The CWT/CCS
must contain a COSE_Key in a 'cnf' claim [RFC8747]. The Value
Registry for this item is empty and omitted from the table below.

Name	Label	Value Type	Description
=======	=======	===============	============================
kcwt	TBD1	COSE_Messages	A CBOR Web Token (CWT) containing a COSE_Key in a 'cnf' claim
kccs	TBD2	map / #6(map)	A CWT Claims Set (CCS) containing a COSE_Key in a 'cnf' claim

9.7. COSE Header Parameters Registry

IANA has extended the Value Type of 'kid' in the "COSE Header
Parameters" registry under the group name "CBOR Object Signing and
Encryption (COSE)" to also allow the Value Type int. The resulting
Value Type is bstr / int. The Value Registry for this item is empty
and omitted from the table below.

Name	Label	Value Type	Description	Reference
kid	4	bstr / int	Key identifier	[[This document]]

9.8. COSE Key Common Parameters Registry

IANA has extended the Value Type of 'kid' in the "COSE Key Common Parameters" registry under the group name "CBOR Object Signing and Encryption (COSE)" to also allow the Value Type int. The resulting Value Type is bstr / int. The Value Registry for this item is empty and omitted from the table below.

+------+-------+------------+---------------+------------------+
| Name | Label | Value Type | Description | Reference |
+------+-------+------------+---------------+------------------+
kid	2	bstr / int	Key identifi-	[[This document]]
			cation value -	
			match to kid	
			in message	
+------+-------+------------+---------------+------------------+

9.9. CWT Confirmation Methods Registry

IANA has extended the Value Type of 'kid' in the "CWT Confirmation Methods" registry under the group name "CBOR Web Token (CWT) Claims" to also allow the Value Type int. The incorrect term binary string has been corrected to bstr. The resulting Value Type is bstr / int. The new updated content for the 'kid' method is shown in the list below.

   *Confirmation Method Name: kid

   *Confirmation Method Description: Key Identifier

   *JWT Confirmation Method Name: kid

   *Confirmation Key: 3

   *Confirmation Value Type(s): bstr / int

   *Change Controller: IESG

   *Specification Document(s): Section 3.4 of RFC 8747 [[This
    document]]

9.10. The Well-Known URI Registry

IANA has added the well-known URI "edhoc" to the "Well-Known URIs" registry under the group name "Well-Known URIs".

   *URI suffix: edhoc

   *Change controller: IETF

*Specification document(s): [[this document]]

   *Related information: None

9.11. Media Types Registry

   IANA has added the media type "application/edhoc" to the "Media
   Types" registry.

   *Type name: application

   *Subtype name: edhoc

   *Required parameters: N/A

   *Optional parameters: N/A

   *Encoding considerations: binary

   *Security considerations: See Section 7 of this document.

   *Interoperability considerations: N/A

   *Published specification: [[this document]] (this document)

   *Applications that use this media type: To be identified

   *Fragment identifier considerations: N/A

   *Additional information:

      -Magic number(s): N/A

      -File extension(s): N/A

      -Macintosh file type code(s): N/A

   *Person & email address to contact for further information: See
    "Authors' Addresses" section.

   *Intended usage: COMMON

   *Restrictions on usage: N/A

   *Author: See "Authors' Addresses" section.

   *Change Controller: IESG

9.12. CoAP Content-Formats Registry

IANA has added the media type "application/edhoc" to the "CoAP Content-Formats" registry under the group name "Constrained RESTful Environments (CoRE) Parameters".

  *Media Type: application/edhoc

  *Encoding:

  *ID: TBD42

  *Reference: [[this document]]

9.13. Resource Type (rt=) Link Target Attribute Values Registry

IANA has added the resource type "core.edhoc" to the "Resource Type (rt=) Link Target Attribute Values" registry under the group name "Constrained RESTful Environments (CoRE) Parameters".

  *Value: "core.edhoc"

  *Description: EDHOC resource.

  *Reference: [[this document]]

Client applications can use this resource type to discover a server's resource for EDHOC, where to send a request for executing the EDHOC protocol.

9.14. Expert Review Instructions

The IANA Registries established in this document is defined as "Expert Review". This section gives some general guidelines for what the experts should be looking for, but they are being designated as experts for a reason so they should be given substantial latitude.

Expert reviewers should take into consideration the following points:

  *Clarity and correctness of registrations. Experts are expected to check the clarity of purpose and use of the requested entries. Expert needs to make sure the values of algorithms are taken from the right registry, when that is required. Expert should consider requesting an opinion on the correctness of registered parameters from relevant IETF working groups. Encodings that do not meet these objective of clarity and completeness should not be registered.

*Experts should take into account the expected usage of fields when approving point assignment. The length of the encoded value should be weighed against how many code points of that length are left, the size of device it will be used on, and the number of code points left that encode to that size.

*Specifications are recommended. When specifications are not provided, the description provided needs to have sufficient information to verify the points above.

10. References

10.1. Normative References

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

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

[I-D.ietf-cose-x509] Schaad, J., "CBOR Object Signing and Encryption
          (COSE): Header parameters for carrying and referencing X.
          509 certificates", Work in Progress, Internet-Draft,
          draft-ietf-cose-x509-08, 14 December 2020, <https://
          www.ietf.org/internet-drafts/draft-ietf-cose-
          x509-08.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>.

[RFC5116] McGrew, D., "An Interface and Algorithms for
          Authenticated Encryption", RFC 5116, DOI 10.17487/
          RFC5116, January 2008, <https://www.rfc-editor.org/info/
          rfc5116>.

[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/info/rfc5280>.

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

[RFC6090] McGrew, D., Igoe, K., and M. Salter, "Fundamental
          Elliptic Curve Cryptography Algorithms", RFC 6090, DOI
          10.17487/RFC6090, February 2011, <https://www.rfc-
          editor.org/info/rfc6090>.

[RFC6979] Pornin, T., "Deterministic Usage of the Digital Signature
          Algorithm (DSA) and Elliptic Curve Digital Signature
          Algorithm (ECDSA)", RFC 6979, DOI 10.17487/RFC6979,
          August 2013, <https://www.rfc-editor.org/info/rfc6979>.

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

[RFC7624] Barnes, R., Schneier, B., Jennings, C., Hardie, T.,
          Trammell, B., Huitema, C., and D. Borkmann,
          "Confidentiality in the Face of Pervasive Surveillance: A
          Threat Model and Problem Statement", RFC 7624, DOI
          10.17487/RFC7624, August 2015, <https://www.rfc-
          editor.org/info/rfc7624>.

[RFC7748] Langley, A., Hamburg, M., and S. Turner, "Elliptic Curves
          for Security", RFC 7748, DOI 10.17487/RFC7748, January
          2016, <https://www.rfc-editor.org/info/rfc7748>.

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

[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>.

[RFC8376] Farrell, S., Ed., "Low-Power Wide Area Network (LPWAN)
          Overview", RFC 8376, DOI 10.17487/RFC8376, May 2018,
          <https://www.rfc-editor.org/info/rfc8376>.

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

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

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

[RFC8724]   Minaburo, A., Toutain, L., Gomez, C., Barthel, D., and
            JC. Zuniga, "SCHC: Generic Framework for Static Context
            Header Compression and Fragmentation", RFC 8724, DOI
            10.17487/RFC8724, April 2020, <https://www.rfc-
            editor.org/info/rfc8724>.

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

[RFC8747]   Jones, M., Seitz, L., Selander, G., Erdtman, S., and H.
            Tschofenig, "Proof-of-Possession Key Semantics for CBOR
            Web Tokens (CWTs)", RFC 8747, DOI 10.17487/RFC8747, March
            2020, <https://www.rfc-editor.org/info/rfc8747>.

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

[RFC9175]   Amsüss, C., Preuß Mattsson, J., and G. Selander,
            "Constrained Application Protocol (CoAP): Echo, Request-
            Tag, and Token Processing", RFC 9175, DOI 10.17487/
            RFC9175, February 2022, <https://www.rfc-editor.org/info/
            rfc9175>.

10.2. Informative References

[Bruni18]   Bruni, A., Sahl Jørgensen, T., Grønbech Petersen, T.,
            and C. Schürmann, "Formal Verification of Ephemeral
            Diffie-Hellman Over COSE (EDHOC)", November 2018,
            <https://www.springerprofessional.de/en/formal-
            verification-of-ephemeral-diffie-hellman-over-cose-edhoc/
            16284348>.

[CborMe]   Bormann, C., "CBOR Playground", May 2018, <http://
            cbor.me/>.

[CNSA]   (Placeholder), ., "Commercial National Security Algorithm
           Suite", August 2015, <https://apps.nsa.gov/iaarchive/
           programs/iad-initiatives/cnsa-suite.cfm>.

[I-D.ietf-core-oscore-edhoc]
           Palombini, F., Tiloca, M., Hoeglund, R., Hristozov, S.,
           and G. Selander, "Profiling EDHOC for CoAP and OSCORE",
           Work in Progress, Internet-Draft, draft-ietf-core-oscore-
           edhoc-03, 7 March 2022, <https://www.ietf.org/archive/id/
           draft-ietf-core-oscore-edhoc-03.txt>.

[I-D.ietf-core-resource-directory] Amsüss, C., Shelby, Z., Koster,
           M., Bormann, C., and P. V. D. Stok, "CoRE Resource
           Directory", Work in Progress, Internet-Draft, draft-ietf-
           core-resource-directory-28, 7 March 2021, <https://
           www.ietf.org/archive/id/draft-ietf-core-resource-
           directory-28.txt>.

[I-D.ietf-cose-cbor-encoded-cert]
           Mattsson, J. P., Selander, G., Raza, S., Höglund, J.,
           and M. Furuhed, "CBOR Encoded X.509 Certificates (C509
           Certificates)", Work in Progress, Internet-Draft, draft-
           ietf-cose-cbor-encoded-cert-03, 10 January 2022,
           <https://www.ietf.org/archive/id/draft-ietf-cose-cbor-
           encoded-cert-03.txt>.

[I-D.ietf-lake-reqs] Vucinic, M., Selander, G., Mattsson, J. P.,
           and D. Garcia-Carrillo, "Requirements for a Lightweight
           AKE for OSCORE", Work in Progress, Internet-Draft, draft-
           ietf-lake-reqs-04, 8 June 2020, <https://www.ietf.org/
           archive/id/draft-ietf-lake-reqs-04.txt>.

[I-D.ietf-lwig-security-protocol-comparison] Mattsson, J. P.,
           Palombini, F., and M. Vucinic, "Comparison of CoAP
           Security Protocols", Work in Progress, Internet-Draft,
           draft-ietf-lwig-security-protocol-comparison-05, 2
           November 2020, <https://www.ietf.org/archive/id/draft-
           ietf-lwig-security-protocol-comparison-05.txt>.

[I-D.ietf-tls-dtls13] Rescorla, E., Tschofenig, H., and N. Modadugu,
           "The Datagram Transport Layer Security (DTLS) Protocol
           Version 1.3", Work in Progress, Internet-Draft, draft-
           ietf-tls-dtls13-43, 30 April 2021, <https://www.ietf.org/
           internet-drafts/draft-ietf-tls-dtls13-43.txt>.

[I-D.mattsson-cfrg-det-sigs-with-noise] Mattsson, J. P., Thormarker,
           E., and S. Ruohomaa, "Deterministic ECDSA and EdDSA
           Signatures with Additional Randomness", Work in Progress,
           Internet-Draft, draft-mattsson-cfrg-det-sigs-with-

noise-04, 15 February 2022, <https://www.ietf.org/
archive/id/draft-mattsson-cfrg-det-sigs-with-
noise-04.txt>.

[I-D.selander-ace-ake-authz]
         Selander, G., Mattsson, J. P., Vučinić, M., Richardson,
         M., and A. Schellenbaum, "Lightweight Authorization for
         Authenticated Key Exchange.", Work in Progress, Internet-
         Draft, draft-selander-ace-ake-authz-04, 22 October 2021,
         <https://www.ietf.org/archive/id/draft-selander-ace-ake-
         authz-04.txt>.

[I-D.selander-lake-traces] Selander, G. and J. P. Mattsson, "Traces
         of EDHOC", Work in Progress, Internet-Draft, draft-
         selander-lake-traces-02, 20 October 2021, <https://
         www.ietf.org/archive/id/draft-selander-lake-
         traces-02.txt>.

[Norrman20] Norrman, K., Sundararajan, V., and A. Bruni, "Formal
         Analysis of EDHOC Key Establishment for Constrained IoT
         Devices", September 2020, <https://arxiv.org/abs/
         2007.11427>.

[RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for
         Constrained-Node Networks", RFC 7228, DOI 10.17487/
         RFC7228, May 2014, <https://www.rfc-editor.org/info/
         rfc7228>.

[RFC7258] Farrell, S. and H. Tschofenig, "Pervasive Monitoring Is
         an Attack", BCP 188, RFC 7258, DOI 10.17487/RFC7258, May
         2014, <https://www.rfc-editor.org/info/rfc7258>.

[RFC7296] Kaufman, C., Hoffman, P., Nir, Y., Eronen, P., and T.
         Kivinen, "Internet Key Exchange Protocol Version 2
         (IKEv2)", STD 79, RFC 7296, DOI 10.17487/RFC7296, October
         2014, <https://www.rfc-editor.org/info/rfc7296>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS)
         Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446,
         August 2018, <https://www.rfc-editor.org/info/rfc8446>.

[RFC8937] Cremers, C., Garratt, L., Smyshlyaev, S., Sullivan, N.,
         and C. Wood, "Randomness Improvements for Security

Protocols", RFC 8937, DOI 10.17487/RFC8937, October 2020,
<https://www.rfc-editor.org/info/rfc8937>.

[SECG]   "Standards for Efficient Cryptography 1 (SEC 1)", May
             2009, <https://www.secg.org/sec1-v2.pdf>.

[SIGMA]   Krawczyk, H., "SIGMA - The 'SIGn-and-MAc' Approach to
             Authenticated Diffie-Hellman and Its Use in the IKE-
             Protocols (Long version)", June 2003, <http://
             webee.technion.ac.il/~hugo/sigma-pdf.pdf>.

[SP-800-56A] Barker, E., Chen, L., Roginsky, A., Vassilev, A., and
             R. Davis, "Recommendation for Pair-Wise Key-Establishment
             Schemes Using Discrete Logarithm Cryptography", NIST
             Special Publication 800-56A Revision 3, April 2018,
             <https://doi.org/10.6028/NIST.SP.800-56Ar3>.

Appendix A. Use with OSCORE and Transfer over CoAP

This appendix describes how to select EDHOC connection identifiers
and derive an OSCORE security context when OSCORE is used with
EDHOC, and how to transfer EDHOC messages over CoAP.

A.1. Selecting EDHOC Connection Identifier

This section specifies a rule for converting from EDHOC connection
identifier to OSCORE Sender/Recipient ID. (An identifier is Sender
ID or Recipient ID depending on from which endpoint is the point of
view, see Section 3.1 of [RFC8613].)

  *If the EDHOC connection identifier is numeric, i.e., encoded as a
   CBOR integer on the wire, it is converted to a (naturally byte-
   string shaped) OSCORE Sender/Recipient ID equal to its CBOR
   encoded form.

For example, a numeric C_R equal to 10 (0x0A in CBOR encoding) is
converted to a (typically client) Sender ID equal to 0x0A, while a
numeric C_I equal to -12 (0x2B in CBOR encoding) is converted to a
(typically client) Sender ID equal to 0x2B.

  *If the EDHOC connection identifier is byte-valued, hence encoded
   as a CBOR byte string on the wire, it is converted to an OSCORE
   Sender/Recipient ID equal to the byte string.

For example, a byte-string valued C_R equal to 0xFF (0x41FF in CBOR
encoding) is converted to a (typically client) Sender ID equal to
0xFF.

Two EDHOC connection identifiers are called "equivalent" if and only
if, by applying the conversion above, they both result in the same

OSCORE Sender/Recipient ID. For example, the two EDHOC connection identifiers with CBOR encoding 0x0A (numeric) and 0x410A (byte-valued) are equivalent since they both result in the same OSCORE Sender/Recipient ID 0x0A.

When EDHOC is used to establish an OSCORE security context, the connection identifiers C_I and C_R MUST NOT be equivalent. Furthermore, in case of multiple OSCORE security contexts with potentially different endpoints, to facilitate retrieval of the correct OSCORE security context, an endpoint SHOULD select an EDHOC connection identifier that when converted to OSCORE Recipient ID does not coincide with its other Recipient IDs.

A.2. Deriving the OSCORE Security Context

This section specifies how to use EDHOC output to derive the OSCORE security context.

After successful processing of EDHOC message_3, Client and Server derive Security Context parameters for OSCORE as follows (see Section 3.2 of [RFC8613]):

   *The Master Secret and Master Salt are derived by using the EDHOC-Exporter interface, see Section 4.3.

The EDHOC Exporter Labels for deriving the OSCORE Master Secret and the OSCORE Master Salt, are "OSCORE_Master_Secret" and "OSCORE_Master_Salt", respectively.

The context parameter is h'' (0x40), the empty CBOR byte string.

By default, key_length is the key length (in bytes) of the application AEAD Algorithm of the selected cipher suite for the EDHOC session. Also by default, salt_length has value 8. The Initiator and Responder MAY agree out-of-band on a longer key_length than the default and on a different salt_length.

```
Master Secret = EDHOC-Exporter("OSCORE_Master_Secret", h'', key_length)
Master Salt   = EDHOC-Exporter("OSCORE_Master_Salt", h'', salt_length)
```

   *The AEAD Algorithm is the application AEAD algorithm of the selected cipher suite for the EDHOC session.

   *The HKDF Algorithm is the one based on the application hash algorithm of the selected cipher suite for the EDHOC session. For example, if SHA-256 is the application hash algorithm of the selected cipher suite, HKDF SHA-256 is used as HKDF Algorithm in the OSCORE Security Context.

*In case the Client is Initiator and the Server is Responder, the
       Client's OSCORE Sender ID and the Server's OSCORE Sender ID are
       determined from the EDHOC connection identifiers C_R and C_I for
       the EDHOC session, respectively, by applying the conversion in
       Appendix A.1. The reverse applies in case the Client is the
       Responder and the Server is the Initiator.

   Client and Server use the parameters above to establish an OSCORE
   Security Context, as per Section 3.2.1 of [RFC8613].

   From then on, Client and Server retrieve the OSCORE protocol state
   using the Recipient ID, and optionally other transport information
   such as the 5-tuple.

A.3. Transferring EDHOC over CoAP

   This section specifies one instance for how EDHOC can be transferred
   as an exchange of CoAP [RFC7252] messages. CoAP provides a reliable
   transport that can preserve packet ordering and handle message
   duplication. CoAP can also perform fragmentation and protect against
   denial-of-service attacks. The underlying CoAP transport should be
   used in reliable mode, in particular when fragmentation is used, to
   avoid, e.g., situations with hanging endpoints waiting for each
   other.

   By default, the CoAP client is the Initiator and the CoAP server is
   the Responder, but the roles SHOULD be chosen to protect the most
   sensitive identity, see Section 8. According to this specification,
   EDHOC is transferred in POST requests and 2.04 (Changed) responses
   to the Uri-Path: "/.well-known/edhoc". An application may define its
   own path that can be discovered, e.g., using resource directory [I-
   D.ietf-core-resource-directory].

   By default, the message flow is as follows: EDHOC message_1 is sent
   in the payload of a POST request from the client to the server's
   resource for EDHOC. EDHOC message_2 or the EDHOC error message is
   sent from the server to the client in the payload of a 2.04
   (Changed) response. EDHOC message_3 or the EDHOC error message is
   sent from the client to the server's resource in the payload of a
   POST request. If needed, an EDHOC error message is sent from the
   server to the client in the payload of a 2.04 (Changed) response.
   Alternatively, if EDHOC message_4 is used, it is sent from the
   server to the client in the payload of a 2.04 (Changed) response
   analogously to message_2.

   In order to correlate a message received from a client to a message
   previously sent by the server, messages sent by the client are
   prepended with the CBOR serialization of the connection identifier
   which the server has chosen. This applies independently of if the

CoAP server is Responder or Initiator. For the default case when the
server is Responder, the prepended connection identifier is C_R, and
C_I if the server is Initiator. If message_1 is sent to the server,
the CBOR simple value "true" (0xf5) is sent in its place (given that
the server has not selected C_R yet).

These identifiers are encoded in CBOR and thus self-delimiting. They
are sent in front of the actual EDHOC message, and only the part of
the body following the identifier is used for EDHOC processing.

Consequently, the application/edhoc media type does not apply to
these messages; their media type is unnamed.

An example of a successful EDHOC exchange using CoAP is shown in
Figure 10. In this case the CoAP Token enables correlation on the
Initiator side, and the prepended C_R enables correlation on the
Responder (server) side.

```
        Client     Server
          |          |
         +--------->| Header: POST (Code=0.02)
          |   POST   | Uri-Path: "/.well-known/edhoc"
          |          | Payload: true, EDHOC message_1
          |          |
         |<---------+ Header: 2.04 Changed
          |   2.04   | Content-Format: application/edhoc
          |          | Payload: EDHOC message_2
          |          |
         +--------->| Header: POST (Code=0.02)
          |   POST   | Uri-Path: "/.well-known/edhoc"
          |          | Payload: C_R, EDHOC message_3
          |          |
         |<---------+ Header: 2.04 Changed
          |   2.04   |
          |          |
```

Figure 10: Transferring EDHOC in CoAP when the Initiator is CoAP Client

The exchange in Figure 10 protects the client identity against
active attackers and the server identity against passive attackers.

An alternative exchange that protects the server identity against
active attackers and the client identity against passive attackers
is shown in Figure 11. In this case the CoAP Token enables the
Responder to correlate message_2 and message_3, and the prepended
C_I enables correlation on the Initiator (server) side. If EDHOC
message_4 is used, C_I is prepended, and it is transported with CoAP
in the payload of a POST request with a 2.04 (Changed) response.

```
         Client     Server
           |          |
           +--------->| Header: POST (Code=0.02)
           |   POST   | Uri-Path: "/.well-known/edhoc"
           |          |
           |<---------+ Header: 2.04 Changed
           |   2.04   | Content-Format: application/edhoc
           |          | Payload: EDHOC message_1
           |          |
           +--------->| Header: POST (Code=0.02)
           |   POST   | Uri-Path: "/.well-known/edhoc"
           |          | Payload: C_I, EDHOC message_2
           |          |
           |<---------+ Header: 2.04 Changed
           |   2.04   | Content-Format: application/edhoc
           |          | Payload: EDHOC message_3
           |          |
```

Figure 11: Transferring EDHOC in CoAP when the Initiator is CoAP Server

To protect against denial-of-service attacks, the CoAP server MAY respond to the first POST request with a 4.01 (Unauthorized) containing an Echo option [RFC9175]. This forces the initiator to demonstrate its reachability at its apparent network address. If message fragmentation is needed, the EDHOC messages may be fragmented using the CoAP Block-Wise Transfer mechanism [RFC7959].

EDHOC does not restrict how error messages are transported with CoAP, as long as the appropriate error message can to be transported in response to a message that failed (see Section 6). EDHOC error messages transported with CoAP are carried in the payload.

A.3.1. Transferring EDHOC and OSCORE over CoAP

When using EDHOC over CoAP for establishing an OSCORE Security Context, EDHOC error messages sent as CoAP responses MUST be sent in the payload of error responses, i.e., they MUST specify a CoAP error response code. In particular, it is RECOMMENDED that such error responses have response code either 4.00 (Bad Request) in case of client error (e.g., due to a malformed EDHOC message), or 5.00 (Internal Server Error) in case of server error (e.g., due to failure in deriving EDHOC key material). The Content-Format of the error response MUST be set to application/edhoc.

A method for combining EDHOC and OSCORE protocols in two round-trips is specified in [I-D.ietf-core-oscore-edhoc].

Appendix B. Compact Representation

As described in Section 4.2 of [RFC6090] the x-coordinate of an elliptic curve public key is a suitable representative for the entire point whenever scalar multiplication is used as a one-way function. One example is ECDH with compact output, where only the x-coordinate of the computed value is used as the shared secret.

This section defines a format for compact representation based on the Elliptic-Curve-Point-to-Octet-String Conversion defined in Section 2.3.3 of [SECG]. Using the notation from [SECG], the output is an octet string of length ceil((log2 q) / 8). See [SECG] for a definition of q, M, X, xp, and ~yp. The steps in Section 2.3.3 of [SECG] are replaced by:

1. Convert the field element xp to an octet string X of length ceil((log2 q) / 8) octets using the conversion routine specified in Section 2.3.5 of [SECG].

2. Output M = X

The encoding of the point at infinity is not supported. Compact representation does not change any requirements on validation. If a y-coordinate is required for validation or compatibily with APIs the value ~yp SHALL be set to zero. For such use, the compact representation can be transformed into the SECG point compressed format by prepending it with the single byte 0x02 (i.e., M = 0x02 || X).

Using compact representation have some security benefits. An implementation does not need to check that the point is not the point at infinity (the identity element). Similarly, as not even the sign of the y-coordinate is encoded, compact representation trivially avoids so called "benign malleability" attacks where an attacker changes the sign, see [SECG].

Appendix C. Use of CBOR, CDDL and COSE in EDHOC

This Appendix is intended to simplify for implementors not familiar with CBOR [RFC8949], CDDL [RFC8610], COSE [I-D.ietf-cose-rfc8152bis-struct], and HKDF [RFC5869].

C.1. CBOR and CDDL

The Concise Binary Object Representation (CBOR) [RFC8949] is a data format designed for small code size and small message size. CBOR builds on the JSON data model but extends it by e.g., encoding binary data directly without base64 conversion. In addition to the binary CBOR encoding, CBOR also has a diagnostic notation that is readable and editable by humans. The Concise Data Definition

Language (CDDL) [RFC8610] provides a way to express structures for protocol messages and APIs that use CBOR. [RFC8610] also extends the diagnostic notation.

CBOR data items are encoded to or decoded from byte strings using a type-length-value encoding scheme, where the three highest order bits of the initial byte contain information about the major type. CBOR supports several different types of data items, in addition to integers (int, uint), simple values, byte strings (bstr), and text strings (tstr), CBOR also supports arrays [] of data items, maps {} of pairs of data items, and sequences [RFC8742] of data items. Some examples are given below.

The EDHOC specification sometimes use CDDL names in CBOR dignostic notation as in e.g., << ID_CRED_R, ? EAD_2 >>. This means that EAD_2 is optional and that ID_CRED_R and EAD_2 should be substituted with their values before evaluation. I.e., if ID_CRED_R = { 4 : h'' } and EAD_2 is omitted then << ID_CRED_R, ? EAD_2 >> = << { 4 : h'' } >>, which encodes to 0x43a10440.

For a complete specification and more examples, see [RFC8949] and [RFC8610]. We recommend implementors to get used to CBOR by using the CBOR playground [CborMe].

```
Diagnostic          Encoded                 Type
-----------------------------------------------------------------
1                   0x01                    unsigned integer
24                  0x1818                  unsigned integer
-24                 0x37                    negative integer
-25                 0x3818                  negative integer
true                0xf5                    simple value
h''                 0x40                    byte string
h'12cd'             0x4212cd                byte string
'12cd'              0x4431326364            byte string
"12cd"              0x6431326364            text string
{ 4 : h'cd' }       0xa10441cd              map
<< 1, 2, true >>    0x430102f5              byte string
[ 1, 2, true ]      0x830102f5              array
( 1, 2, true )      0x0102f5                sequence
1, 2, true          0x0102f5                sequence
-----------------------------------------------------------------
```

C.2. CDDL Definitions

This sections compiles the CDDL definitions for ease of reference.

```
suites = [ 2* int ] / int

ead = 1* (
  ead_label : int,
  ead_value : any,
)

message_1 = (
  METHOD : int,
  SUITES_I : suites,
  G_X : bstr,
  C_I : bstr / int,
  ? EAD_1 : ead,
)

message_2 = (
  G_Y_CIPHERTEXT_2 : bstr,
  C_R : bstr / int,
)

message_3 = (
  CIPHERTEXT_3 : bstr,
)

message_4 = (
  CIPHERTEXT_4 : bstr,
)

error = (
  ERR_CODE : int,
  ERR_INFO : any,
)

info = (
  transcript_hash : bstr,
  label : tstr,
  context : bstr,
  length : uint,
)
```

C.3. COSE

CBOR Object Signing and Encryption (COSE) [I-D.ietf-cose-rfc8152bis-struct] describes how to create and process signatures, message authentication codes, and encryption using CBOR. COSE builds on JOSE, but is adapted to allow more efficient processing in

constrained devices. EDHOC makes use of COSE_Key, COSE_Encrypt0, and COSE_Sign1 objects in the message processing:

  *ECDH ephemeral public keys of type EC2 or OKP in message_1 and message_2 consist of the COSE_Key parameter named 'x', see Section 7.1 and 7.2 of [I-D.ietf-cose-rfc8152bis-algs]

  *The ciphertexts in message_3 and message_4 consist of a subset of the single recipient encrypted data object COSE_Encrypt0, which is described in Sections 5.2-5.3 of [I-D.ietf-cose-rfc8152bis-struct]. The ciphertext is computed over the plaintext and associated data, using an encryption key and an initialization vector. The associated data is an Enc_structure consisting of protected headers and externally supplied data (external_aad). COSE constructs the input to the AEAD [RFC5116] for message_i (i = 3 or 4, see Section 5.4 and Section 5.5, respectively) as follows:

    -Secret key K = K_i

    -Nonce N = IV_i

    -Plaintext P for message_i

    -Associated Data A = ["Encrypt0", h'', TH_i]

  *Signatures in message_2 of method 0 and 2, and in message_3 of method 0 and 1, consist of a subset of the single signer data object COSE_Sign1, which is described in Sections 4.2-4.4 of [I-D.ietf-cose-rfc8152bis-struct]. The signature is computed over a Sig_structure containing payload, protected headers and externally supplied data (external_aad) using a private signature key and verified using the corresponding public signature key. For COSE_Sign1, the message to be signed is:

    ["Signature1", protected, external_aad, payload]

  where protected, external_aad and payload are specified in Section 5.3 and Section 5.4.

Different header parameters to identify X.509 or C509 certificates by reference are defined in [I-D.ietf-cose-x509] and [I-D.ietf-cose-cbor-encoded-cert]:

  *by a hash value with the 'x5t' or 'c5t' parameters, respectively:

    -ID_CRED_x = { 34 : COSE_CertHash }, for x = I or R,

    -ID_CRED_x = { TBD3 : COSE_CertHash }, for x = I or R;

        *or by a URI with the 'x5u' or 'c5u' parameters, respectively:

          -ID_CRED_x = { 35 : uri }, for x = I or R,

          -ID_CRED_x = { TBD4 : uri }, for x = I or R.

   When ID_CRED_x does not contain the actual credential, it may be
   very short, e.g., if the endpoints have agreed to use a key
   identifier parameter 'kid':

     *ID_CRED_x = { 4 : key_id_x }, where key_id_x : kid, for x = I or
      R.

   Note that a COSE header map can contain several header parameters,
   for example { x5u, x5t } or { kid, kid_context }.

   ID_CRED_x MAY also identify the authentication credential by value.
   For example, a certificate chain can be transported in ID_CRED_x
   with COSE header parameter c5c or x5chain, defined in [I-D.ietf-
   cose-cbor-encoded-cert] and [I-D.ietf-cose-x509] and credentials of
   type CWT and CCS can be transported with the COSE header parameters
   registered in Section 9.6.

Appendix D. Applicability Template

   This appendix contains a rudimentary example of an applicability
   statement, see Section 3.9.

   For use of EDHOC in the XX protocol, the following assumptions are
   made:

   1. Transfer in CoAP as specified in Appendix A.3 with requests
      expected by the CoAP server (= Responder) at /app1-edh, no
      Content-Format needed.

   2. METHOD = 1 (I uses signature key, R uses static DH key.)

   3. CRED_I is an IEEE 802.1AR IDevID encoded as a C509 certificate
      of type 0 [I-D.ietf-cose-cbor-encoded-cert].

      *R acquires CRED_I out-of-band, indicated in EAD_1.

      *ID_CRED_I = {4: h''} is a 'kid' with value empty CBOR byte
       string.

   4. CRED_R is a CCS of type OKP as specified in Section 3.5.3.

      *The CBOR map has parameters 1 (kty), -1 (crv), and -2 (x-
       coordinate).

*ID_CRED_R is {TBD2 : CCS}. Editor's note: TBD2 is the COSE
    header parameter value of 'kccs', see Section 9.6

5. External authorization data is defined and processed as
   specified in [I-D.selander-ace-ake-authz].

6. EUI-64 used as identity of endpoint.

7. No use of message_4: the application sends protected messages
   from R to I.

Appendix E. EDHOC Message Deduplication

EDHOC by default assumes that message duplication is handled by the
transport, in this section exemplified with CoAP.

Deduplication of CoAP messages is described in Section 4.5 of
[RFC7252]. This handles the case when the same Confirmable (CON)
message is received multiple times due to missing acknowledgement on
CoAP messaging layer. The recommended processing in [RFC7252] is
that the duplicate message is acknowledged (ACK), but the received
message is only processed once by the CoAP stack.

Message deduplication is resource demanding and therefore not
supported in all CoAP implementations. Since EDHOC is targeting
constrained environments, it is desirable that EDHOC can optionally
support transport layers which does not handle message duplication.
Special care is needed to avoid issues with duplicate messages, see
Section 5.1.

The guiding principle here is similar to the deduplication
processing on CoAP messaging layer: a received duplicate EDHOC
message SHALL NOT result in a response consisting of another
instance of the next EDHOC message. The result MAY be that a
duplicate EDHOC response is sent, provided it is still relevant with
respect the current protocol state. In any case, the received
message MUST NOT be processed more than once in the same EDHOC
session. This is called "EDHOC message deduplication".

An EDHOC implementation MAY store the previously sent EDHOC message
to be able to resend it. An EDHOC implementation MAY keep the
protocol state to be able to recreate the previously sent EDHOC
message and resend it. The previous message or protocol state MUST
NOT be kept longer than what is required for retransmission, for
example, in the case of CoAP transport, no longer than the
EXCHANGE_LIFETIME (see Section 4.8.2 of [RFC7252]).

Note that the requirements in [Section 5.1](#) still apply because duplicate messages are not processed by the EDHOC state machine:

  *EDHOC messages SHALL be processed according to the current protocol state.

  *Different instances of the same message MUST NOT be processed in one session.

Appendix F. Transports Not Natively Providing Correlation

Protocols that do not natively provide full correlation between a series of messages can send the C_I and C_R identifiers along as needed.

The transport over CoAP ([Appendix A.3](#)) can serve as a blueprint for other server-client protocols: The client prepends the C_x which the server selected (or, for message 1, the CBOR simple value 'true' which is not a valid C_x) to any request message it sends. The server does not send any such indicator, as responses are matched to request by the client-server protocol design.

Protocols that do not provide any correlation at all can prescribe prepending of the peer's chosen C_x to all messages.

Appendix G. Change Log

RFC Editor: Please remove this appendix.

  *From -11 to -12:

      -Clarified applicability to KEMs

      -Clarified use of COSE header parameters

      -Updates on MTI

      -Updated security considerations

      -New section on PQC

      -Removed duplicate definition of cipher suites

      -Explanations of use of COSE moved to Appendix C.3

      -Updated internal references

  *From -10 to -11:

      -Restructured section on authentication parameters

-Changed UCCS to CCS

-Changed names and description of COSE header parameters for
 CWT/CCS

-Changed several of the KDF and Exporter labels

-Removed edhoc_aead_id from info (already in transcript_hash)

-Added MTI section

-EAD: changed CDDL names and added value type to registry

-Updated Figures 1, 2, and 3

-Some correction and clarifications

-Added core.edhoc to CoRE Resource Type registry

*From -09 to -10:

-SUITES_I simplified to only contain the selected and more
 preferred suites

-Info is a CBOR sequence and context is a bstr

-Added kid to UCCS example

-Separate header parameters for CWT and UCCS

-CWT Confirmation Method kid extended to bstr / int

*From -08 to -09:

-G_Y and CIPHERTEXT_2 are now included in one CBOR bstr

-MAC_2 and MAC_3 are now generated with EDHOC-KDF

-Info field "context" is now general and explicit in EDHOC-KDF

-Restructured Section 4, Key Derivation

-Added EDHOC MAC length to cipher suite for use with static DH

-More details on the use of CWT and UCCS

-Restructured and clarified Section 3.5, Authentication
 Parameters

-Replaced 'kid2' with extension of 'kid'

-EAD encoding now supports multiple ead types in one message

-Clarified EAD type

-Updated message sizes

-Replaced "perfect forward secrecy" with "forward secrecy"

-Updated security considerations

-Replaced prepended 'null' with 'true' in the CoAP transport of
 message_1

-Updated CDDL definitions

-Expanded on the use of COSE

*From -07 to -08:

-Prepended C_x moved from the EDHOC protocol itself to the
 transport mapping

-METHOD_CORR renamed to METHOD, corr removed

-Removed bstr_identifier and use bstr / int instead; C_x can
 now be int without any implied bstr semantics

-Defined COSE header parameter 'kid2' with value type bstr /
 int for use with ID_CRED_x

-Updated message sizes

-New cipher suites with AES-GCM and ChaCha20 / Poly1305

-Changed from one- to two-byte identifier of CNSA compliant
 suite

-Separate sections on transport and connection id with further
 sub-structure

-Moved back key derivation for OSCORE from draft-ietf-core-
 oscore-edhoc

-OSCORE and CoAP specific processing moved to new appendix

-Message 4 section moved to message processing section

*From -06 to -07:

-Changed transcript hash definition for TH_2 and TH_3

-Removed "EDHOC signature algorithm curve" from cipher suite

-New IANA registry "EDHOC Exporter Label"

-New application defined parameter "context" in EDHOC-Exporter

-Changed normative language for failure from MUST to SHOULD
 send error

-Made error codes non-negative and 0 for success

-Added detail on success error code

-Aligned terminology "protocol instance" -> "session"

-New appendix on compact EC point representation

-Added detail on use of ephemeral public keys

-Moved key derivation for OSCORE to draft-ietf-core-oscore-
 edhoc

-Additional security considerations

-Renamed "Auxililary Data" as "External Authorization Data"

-Added encrypted EAD_4 to message_4

*From -05 to -06:

-New section 5.2 "Message Processing Outline"

-Optional inital byte C_1 = null in message_1

-New format of error messages, table of error codes, IANA
 registry

-Change of recommendation transport of error in CoAP

-Merge of content in 3.7 and appendix C into new section 3.7
 "Applicability Statement"

-Requiring use of deterministic CBOR

-New section on message deduplication

-New appendix containin all CDDL definitions

-New appendix with change log

-Removed section "Other Documents Referencing EDHOC"

-Clarifications based on review comments

*From -04 to -05:

-EDHOC-Rekey-FS -> EDHOC-KeyUpdate

-Clarification of cipher suite negotiation

-Updated security considerations

-Updated test vectors

-Updated applicability statement template

*From -03 to -04:

-Restructure of section 1

-Added references to C509 Certificates

-Change in CIPHERTEXT_2 -> plaintext XOR KEYSTREAM_2 (test
 vector not updated)

-"K_2e", "IV_2e" -> KEYSTREAM_2

-Specified optional message 4

-EDHOC-Exporter-FS -> EDHOC-Rekey-FS

-Less constrained devices SHOULD implement both suite 0 and 2

-Clarification of error message

-Added exporter interface test vector

*From -02 to -03:

-Rearrangements of section 3 and beginning of section 4

-Key derivation new section 4

-Cipher suites 4 and 5 added

-EDHOC-EXPORTER-FS - generate a new PRK_4x3m from an old one

-Change in CIPHERTEXT_2 -> COSE_Encrypt0 without tag (no change
 to test vector)

-Clarification of error message

-New appendix C applicability statement

*From -01 to -02:

     -New section 1.2 Use of EDHOC

     -Clarification of identities

     -New section 4.3 clarifying bstr_identifier

     -Updated security considerations

     -Updated text on cipher suite negotiation and key confirmation

     -Test vector for static DH

  *From -00 to -01:

     -Removed PSK method

     -Removed references to certificate by value

Acknowledgments

Authors' Addresses

Göran Selander
Ericsson AB
SE-164 80 Stockholm
Sweden

Email: goran.selander@ericsson.com

John Preuß Mattsson
Ericsson AB
SE-164 80 Stockholm

Sweden

Email: john.mattsson@ericsson.com

Francesca Palombini
Ericsson AB
SE-164 80 Stockholm
Sweden

Email: francesca.palombini@ericsson.com