Ephemeral Diffie-Hellman Over COSE (EDHOC)
draft-selander-ace-cose-ecdhe-11

Abstract

   This document specifies Ephemeral Diffie-Hellman Over COSE (EDHOC), a
   very compact, and lightweight authenticated Diffie-Hellman key
   exchange with ephemeral keys that can be used over any layer. EDHOC
   provides mutual authentication, perfect forward secrecy, and identity
   protection. EDHOC uses CBOR and COSE, allowing reuse of existing
   libraries.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

   Security at the application layer provides an attractive option for
   protecting Internet of Things (IoT) deployments, for example where
   transport layer security is not sufficient
   [I-D.hartke-core-e2e-security-reqs] or where the protocol needs to
   work on a variety of underlying protocols. IoT devices may be
   constrained in various ways, including memory, storage, processing
   capacity, and energy [RFC7228]. A method for protecting individual
   messages at the application layer suitable for constrained devices,
   is provided by CBOR Object Signing and Encryption (COSE) [RFC8152]),
   which builds on the Concise Binary Object Representation (CBOR)
   [I-D.ietf-cbor-7049bis].

   In order for a communication session to provide forward secrecy, the
   communicating parties can run an Elliptic Curve Diffie-Hellman (ECDH)
   key exchange protocol with ephemeral keys, from which shared key
   material can be derived. This document specifies Ephemeral Diffie-
   Hellman Over COSE (EDHOC), a lightweight key exchange protocol
   providing perfect forward secrecy and identity protection. EDHOC
   uses CBOR and COSE, allowing reuse of existing libraries.
   Authentication is based on credentials established out of band, e.g.
   from a trusted third party, such as an Authorization Server as
   specified by [I-D.ietf-ace-oauth-authz]. EDHOC supports
   authentication using pre-shared keys (PSK), raw public keys (RPK),
   and public key certificates. After successful completion of the
   EDHOC protocol, application keys and other application specific data
   can be derived using the EDHOC-Exporter interface. Note that this
   document focuses on authentication and key establishment: for
   integration with authorization of resource access, refer to
   [I-D.ietf-ace-oscore-profile].

   EDHOC is designed to work in highly constrained scenarios making it
   especially suitable for network technologies such as Cellular IoT,
   6TiSCH [I-D.ietf-6tisch-dtsecurity-zerotouch-join], and LoRaWAN
   [LoRa1][LoRa2]. Compared to the DTLS 1.3 handshake
   [I-D.ietf-tls-dtls13] with ECDH and connection ID, the number of
   bytes in EDHOC is less than 1/4 when PSK authentication is used and
   less than 1/3 when RPK authentication is used, see Appendix E.

The ECDH exchange and the key derivation follow [SIGMA], NIST SP-
800-56A [SP-800-56A], and HKDF [RFC5869]. CBOR
[I-D.ietf-cbor-7049bis] and COSE [RFC8152] are used to implement
these standards.

This document is organized as follows: Section 2 describes how EDHOC
builds on SIGMA-I, Section 3 specifies general properties of EDHOC,
including message flow, formatting of the ephemeral public keys, and
key derivation, Section 4 specifies EDHOC with asymmetric key
authentication, Section 5 specifies EDHOC with symmetric key
authentication, Section 6 specifies the EDHOC error message, and
Appendix B provides a wealth of test vectors to ease implementation
and ensure interoperability.

1.1. Rationale for EDHOC

EDHOC is optimized for small message overhead. The message size of a
key exchange protocol may have a large impact on the performance of
an IoT deployment. For example, in a network bootstrapping setting a
large number of devices turned on in a short period of time may
result in large latencies caused by parallel key exchanges.
Requirements on network formation time can in constrained
environments be translated into key exchange overhead.

Power consumption for wireless devices is highly dependent on message
transmission and reception. For devices that only send a few bytes
occasionally, the battery lifetime may be significantly reduced by a
heavy key exchange protocol. Moreover, a key exchange may need to be
executed more than once, e.g. due to a device losing power or
rebooting for other reason.

EDHOC is adapted to primitives and protocols designed for the
Internet of Things: EDHOC is built on CBOR and COSE which enables
small message overhead and efficient parsing in constrained devices.
Since EDHOC is not bound to a particular transport layer, the
protocol messages can e.g. be carried as CoAP payload. By reusing
already existing IoT primitives in the device (CBOR, CoAP and COSE
encryption and signature formats) the additional code footprint can
be kept very low.

EDHOC is not bound to a particular communication security protocol
but works off-the-shelf with OSCORE [I-D.ietf-core-object-security]
providing the necessary input parameters with required properties.
Since EDHOC builds on the same IoT primitives and protocols as OSCORE
(CBOR, COAP, COSE encryption and signature formats) the device
footprint for EDHOC + OSCORE can be kept very low. The use of
compact native encoding formats reduces the need for a general
purpose compression algorithm with associated footprint.

1.2. 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.

The word "encryption" without qualification always refers to
authenticated encryption, in practice implemented with an
Authenticated Encryption with Additional Data (AEAD) algorithm, see
[RFC5116].

This document uses the Concise Data Definition Language (CDDL)
[I-D.ietf-cbor-cddl] to express CBOR data structures
[I-D.ietf-cbor-7049bis]. The use of the CDDL unwrap operator "~" is
extended to unwrapping of byte strings. It is the inverse of "bstr
.cbor" that wraps a data item in a bstr, i.e. ~ bstr .cbor T = T.
Examples of CBOR and CDDL are provided in Appendix A.1.

2. Background

SIGMA (SIGn-and-MAc) is a family of theoretical protocols with a
large number of variants [SIGMA]. Like IKEv2 and (D)TLS 1.3, EDHOC
is built on a variant of the SIGMA protocol which provide identity
protection of the initiator (SIGMA-I), and like (D)TLS 1.3, EDHOC
implements the SIGMA-I variant as Sign-then-MAC. The SIGMA-I
protocol using an authenticated encryption algorithm is shown in
Figure 1.

```
   Party U                                                    Party V
      |                             X_U                             |
     +------------------------------------------------------------>|
      |                                                             |
      |   X_V, AE( K_2; ID_CRED_V, Sig(V; CRED_V, X_U, X_V) )   |
      |<-----------------------------------------------------------+
      |                                                             |
      |     AE( K_3; ID_CRED_U, Sig(U; CRED_U, X_V, X_U) )      |
     +------------------------------------------------------------>|
      |                                                             |
```
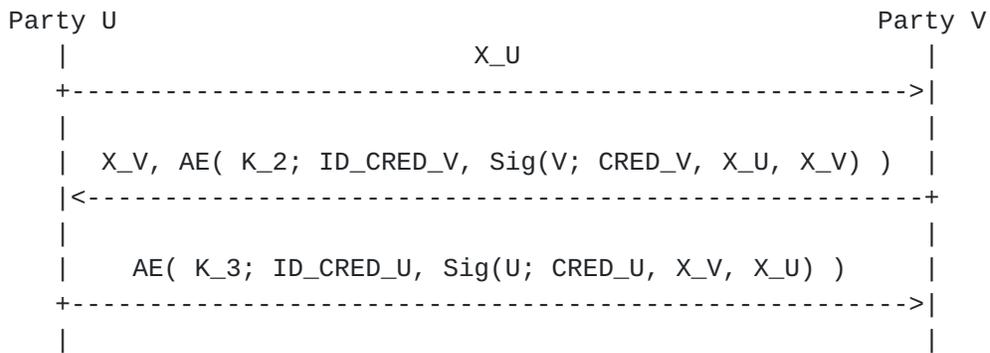
  Figure 1: Authenticated encryption variant of the SIGMA-I protocol.

The parties exchanging messages are called "U" and "V". They
exchange identities and ephemeral public keys, compute the shared
secret, and derive symmetric application keys.

o X_U and X_V are the ECDH ephemeral public keys of U and V,
   respectively.

o CRED_U and CRED_V are the credentials containing the public
   authentication keys of U and V, respectively.

o ID_CRED_U and ID_CRED_V are data enabling the recipient party to
   retrieve the credential of U and V, respectively

o Sig(U; .) and S(V; .) denote signatures made with the private
   authentication key of U and V, respectively.

o AE(K; P) denotes authenticated encryption of plaintext P using the
   key K derived from the shared secret. The authenticated
   encryption MUST NOT be replaced by plain encryption, see
   Section 8.

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

o Explicit connection identifiers C_U, C_V chosen by U and V,
   respectively, enabling the recipient to find the protocol state.

o An Authenticated Encryption with Additional Data (AEAD) algorithm
   is used.

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

o Verification of a common preferred cipher suite (AEAD algorithm,
   ECDH algorithm, ECDH curve, signature algorithm):

   * U lists supported cipher suites in order of preference

   * V verifies that the selected cipher suite is the first
      supported cipher suite

o Message types and error handling.

o Transport of opaque application defined 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 and
COSE libraries. EDHOC does not put any requirement on the lower

   layers and can therefore also be used e.g. in environments without
   IP.

   To simplify implementation, the use of CBOR and COSE in EDHOC is
   summarized in Appendix A.

3. EDHOC Overview

   EDHOC consists of three messages (message_1, message_2, message_3)
   that maps directly to the three messages in SIGMA-I, plus an EDHOC
   error message. All EDHOC messages consists of a sequence of CBOR
   encoded data items, where the first data item of message_1 is an int
   specifying the message type (MSG_TYPE). The messages may be viewed
   as a CBOR encoding of an indefinite-length array without the first
   and last byte, see Appendix A.1.

   While EDHOC uses the COSE_Key, COSE_Sign1, and COSE_Encrypt0
   structures, only a subset of the parameters are included in the EDHOC
   messages. After creating EDHOC message_3, Party U can derive
   symmetric application keys, and application protected data can
   therefore be sent in parallel with EDHOC message_3. The application
   may protect data using the algorithms (AEAD, HKDF, etc.) in the
   selected cipher suite and the connection identifiers (C_U, C_V).
   EDHOC may be used with the media type application/edhoc defined in
   Section 7.

```
     Party U                                               Party V
        |                                                     |
        | ----------------- EDHOC message_1 ----------------> |
        |                                                     |
        | <---------------- EDHOC message_2 ----------------- |
        |                                                     |
        | ----------------- EDHOC message_3 ----------------> |
        |                                                     |
        | <----------- Application Protected Data ----------> |
        |                                                     |
```
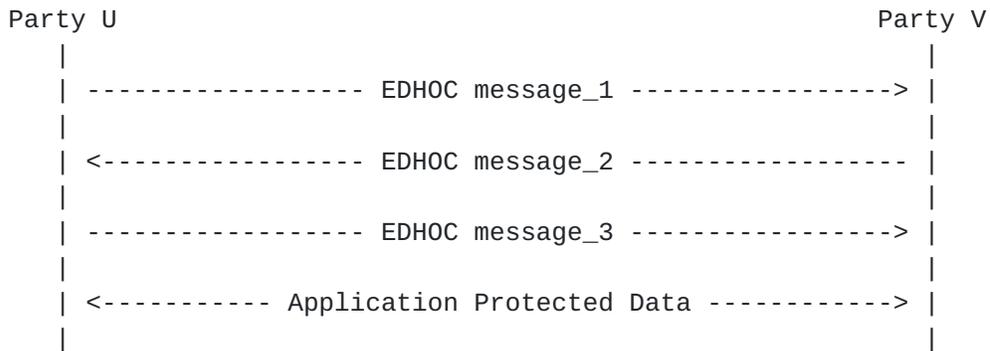
                    Figure 2: EDHOC message flow

   The EDHOC message exchange may be authenticated using pre-shared keys
   (PSK), raw public keys (RPK), or public key certificates. EDHOC
   assumes the existence of mechanisms (certification authority, manual
   distribution, etc.) for binding identities with authentication keys
   (public or pre-shared). When a public key infrastructure is used,
   the identity is included in the certificate and bound to the
   authentication key by trust in the certification authority. When the
   credential is manually distributed (PSK, RPK, self-signed
   certificate), the identity and authentication key is distributed out-

of-band and bound together by trust in the distribution method.
EDHOC with symmetric key authentication is very similar to EDHOC with
asymmetric key authentication, the difference being that information
is only MACed, not signed.

EDHOC allows opaque application data (UAD and PAD) to be sent in the
EDHOC messages. Unprotected Application Data (UAD_1, UAD_2) may be
sent in message_1 and message_2, while Protected Application Data
(PAD_3) may be send in message_3.

3.1. Cipher Suites

EDHOC cipher suites consists of a set of COSE algorithms: an AEAD
algorithm, an ECDH algorithm, an ECDH curve (including HKDF
algorithm), and a signature algorithm. The signature algorithm is
not used when EDHOC is authenticated with symmetric keys. Each
cipher suite is associated with an integer value.

1. AES-CCM-64-64-128, ECDH-SS + HKDF-256, X25519, and Ed25519

2. AES-CCM-64-64-128, ECDH-SS + HKDF-256, P-256, and ES256

3. Application defined.

4. Application defined.

3.2. Ephemeral Public Keys

The ECDH ephemeral public keys are formatted as a COSE_Key of type
EC2 or OKP according to Sections 13.1 and 13.2 of [RFC8152], but only
a subset of the parameters are included in the EDHOC messages. For
Elliptic Curve Keys of type EC2, compact representation as per
[RFC6090] MAY be used also in the COSE_Key. If the COSE
implementation requires an y-coordinate, any of the possible values
of the y-coordinate can be used, see Appendix C of [RFC6090]. COSE
[RFC8152] always use compact output for Elliptic Curve Keys of type
EC2.

3.3. Key Derivation

Key and IV derivation SHALL be performed as specified in Section 11
of [RFC8152] with the following input:

o The KDF SHALL be the HKDF [RFC5869] in the in the selected cipher
   suite (CIPHER_SUITE_U).

o The secret (Section 11.1 of [RFC8152]) SHALL be the ECDH shared
   secret as defined in Section 12.4.1 of [RFC8152].

   o The salt (Section 11.1 of [RFC8152]) SHALL be the PSK when EDHOC
      is authenticated with symmetric keys, and the empty byte string
      when EDHOC is authenticated with asymmetric keys. 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 empty byte string.

   o The fields in the context information COSE_KDF_Context
      (Section 11.2 of [RFC8152]) SHALL have the following values:

      * AlgorithmID is an int or tstr, see below

      * PartyUInfo = PartyVInfo = (null, null, null)

      * keyDataLength is a uint, see below

      * protected SHALL be a zero length bstr

      * other is a bstr and SHALL be aad_2, aad_3, or exchange_hash;
         see below

      * SuppPrivInfo is omitted

   where exchange_hash, in non-CDDL notation, is:

   exchange_hash = H(bstr .cborseq [aad_3, CIPHERTEXT_3])

   where H() is the hash function in the HKDF, which takes a CBOR byte
   string (bstr) as input and produces a CBOR byte string as output.
   The use of '.cborseq' is exemplified in Appendix A.1.

   We define EDHOC-Key-Derivation to be the function which produces the
   output as described in [RFC5869] and [RFC8152] depending on the
   variable input AlgorithmID, keyDataLength, and other:

   output = EDHOC-Key-Derivation(AlgorithmID, keyDataLength, other)

   For message_i the key, called K_i, SHALL be derived using other =
   aad_i, where i = 2 or 3. The key SHALL be derived using AlgorithmID
   set to the integer value of the AEAD in the selected cipher suite
   (CIPHER_SUITE_U), and keyDataLength equal to the key length of the
   AEAD.

   If the AEAD algorithm uses an IV, then IV_i for message_i SHALL be
   derived using other = aad_i, where i = 2 or 3. The IV SHALL be
   derived using AlgorithmID = "IV-GENERATION" as specified in

Section 12.1.2. of [RFC8152], and keyDataLength equal to the IV
length of the AEAD.

3.3.1. EDHOC-Exporter Interface

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

EDHOC-Exporter(label, length) = EDHOC-Key-Derivation(label, 8 *
length, exchange_hash)

The output of the EDHOC-Exporter function SHALL be derived using
other = exchange_hash, AlgorithmID = label, and keyDataLength = 8 *
length, where label is a tstr defined by the application and length
is a uint defined by the application. The label SHALL be different
for each different exporter value. An example use of the EDHOC-
Exporter is given in Appendix D.2).

4. EDHOC Authenticated with Asymmetric Keys

4.1. Overview

EDHOC supports authentication with raw public keys (RPK) and public
key certificates with the requirements that:

o Party U SHALL be able to retrieve Party V's public authentication
   key using ID_CRED_V,

o Party V SHALL be able to retrieve Party U's public authentication
   key using ID_CRED_U,

where ID_CRED_x, for x = U or V, is encoded in a COSE map, see
Appendix A.2. In the following we give some examples of possible
COSE map labels.

Raw public keys are most optimally stored as COSE_Key objects and
identified with a 'kid' value (see [RFC8152]):

o kid : ID_CRED_x, for x = U or V.

Public key certificates can be identified in different ways, for
example (see [I-D.schaad-cose-x509]):

o by a hash value;

   * x5t : ID_CRED_x, for x = U or V,

o by a URL;

        * x5u : ID_CRED_x, for x = U or V,

   o by a certificate chain;

        * x5chain : ID_CRED_x, for x = U or V,

   o or by a bag of certificates.

        * x5bag : ID_CRED_x, for x = U or V.

   In the latter two examples, ID_CRED_U and ID_CRED_V contains the
   actual credential used for authentication. ID_CRED_U and ID_CRED_V
   do not need to uniquely identify the public authentication key, but
   doing so is recommended as the recipient may otherwise have to try
   several public keys. ID_CRED_U and ID_CRED_V are transported in the
   ciphertext, see Section 4.3.2 and Section 4.4.2.

   The actual credentials CRED_U and CRED_V (e.g. a COSE_Key or a single
   X.509 certificate) are signed by party U and V, respectively, see
   Section 4.4.1 and Section 4.3.1. Party U and Party V MAY use
   different type of credentials, e.g. one uses RPK and the other uses
   certificate. Party U and Party V MAY use different signature
   algorithms.

   EDHOC with asymmetric key authentication is illustrated in Figure 3.

```
Party U                                                         Party V
|           C_U, CIPHER_SUITEs_U, CIPHER_SUITE_U, X_U, UAD_1         |
+------------------------------------------------------------------->|
|                            message_1                              |
|                                                                   |
|  C_U, C_V, X_V, AE(K_2; ID_CRED_V, Sig(V; CRED_V, aad_2), UAD_2)  |
|<------------------------------------------------------------------+
|                            message_2                              |
|                                                                   |
|      C_V, AE(K_3; ID_CRED_U, Sig(U; CRED_U, aad_3), PAD_3)        |
+------------------------------------------------------------------->|
|                            message_3                              |
```
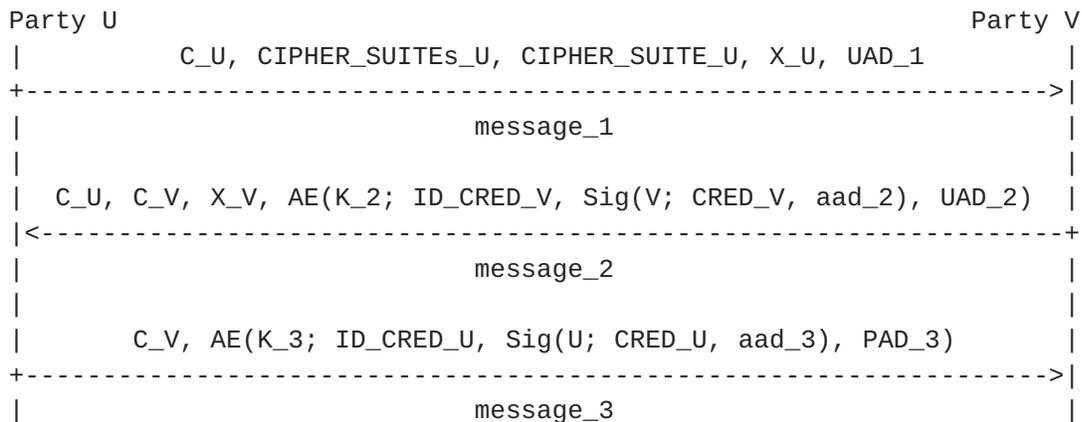
      Figure 3: Overview of EDHOC with asymmetric key authentication.

4.2. EDHOC Message 1

4.2.1. Formatting of Message 1

   message_1 SHALL be a sequence of CBOR data items (see Appendix A.1)
   as defined below

```
message_1 = (
  MSG_TYPE : int,
  C_U : bstr,
  CIPHER_SUITEs_U : suites,
  CIPHER_SUITE_U : uint,
  X_U : bstr,
  ? UAD_1 : bstr,
)
```

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

where:

o MSG_TYPE = 1

o C_U - variable length connection identifier

o CIPHER_SUITEs_U - cipher suites which Party U supports, in order
   of decreasing preference. If a single cipher suite is conveyed,
   an int is used, if multiple cipher suites are conveyed, an array
   of ints is used.

o CIPHER_SUITE_U - a single chosen cipher suite from CIPHER_SUITEs_U
   (zero-based index, i.e. 0 for the first or only, 1 for the second,
   etc.)

o X_U - the x-coordinate of the ephemeral public key of Party U

o UAD_1 - bstr containing unprotected opaque application data

4.2.2. Party U Processing of Message 1

Party U SHALL compose message_1 as follows:

o The supported cipher suites and the order of preference MUST NOT
   be changed based on previous error messages. However, the list
   CIPHER_SUITEs_U sent to Party V MAY be truncated such that cipher
   suites which are the least preferred are omitted. The amount of
   truncation 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 least preferred cipher suite in the list
   MUST be included in the list.

o Determine the cipher suite CIPHER_SUITE_U to use with Party V in
   message_1. If Party U previously received from Party V an error
   message to message_1 with diagnostic payload identifying a cipher
   suite that U supports, then U SHALL use that cipher suite.
   Otherwise the first cipher suite in CIPHER_SUITEs_U MUST be used.

o Generate an ephemeral ECDH key pair as specified in Section 5 of
   [SP-800-56A] using the curve in the cipher suite CIPHER_SUITE_U.
   Let X_U be the x-coordinate of the ephemeral public key.

o Choose a connection identifier C_U and store it for the length of
   the protocol. Party U MUST be able to retrieve the protocol state
   using the connection identifier C_U and optionally other
   information such as the 5-tuple. The connection identifier MAY be
   used with protocols for which EDHOC establishes application keys,
   in which case C_U SHALL be different from the concurrently used
   identifiers of that protocol.

o Format message_1 as the sequence of CBOR data items specified in
   Section 4.2.1 and encode it to a byte string (see Appendix A.1).

4.2.3. Party V Processing of Message 1

Party V SHALL process message_1 as follows:

o Decode message_1 (see Appendix A.1).

o Verify that the cipher suite CIPHER_SUITE_U is supported and that
   no prior cipher suites in CIPHER_SUITEs_U are supported.

o Validate that there is a solution to the curve definition for the
   given x-coordinate X_U.

o Pass UAD_1 to the application.

If any verification step fails, Party V MUST send an EDHOC error
message back, formatted as defined in Section 6, and the protocol
MUST be discontinued. If V does not support the cipher suite
CIPHER_SUITE_U, then CIPHER_SUITEs_V MUST include one or more
supported cipher suites. If V does not support the cipher suite
CIPHER_SUITE_U, but supports another cipher suite in CIPHER_SUITEs_U,
then CIPHER_SUITEs_V MUST include the first supported cipher suite in
CIPHER_SUITEs_U.

4.3. EDHOC Message 2

4.3.1. Formatting of Message 2

message_2 SHALL be a sequence of CBOR data items (see Appendix A.1)
as defined below

```
message_2 = (
  data_2,
  CIPHERTEXT_2 : bstr,
)

data_2 = (
  C_U : bstr / nil,
  C_V : bstr,
  X_V : bstr,
)

aad_2 : bstr
```

where aad_2, in non-CDDL notation, is:

```
aad_2 = H( bstr .cborseq [ message_1, data_2 ] )
```

where:

o C_V - variable length connection identifier

o X_V - the x-coordinate of the ephemeral public key of Party V

o H() - the hash function in the HKDF, which takes a CBOR byte
   string (bstr) as input and produces a CBOR byte string as output.
   The use of '.cborseq' is exemplified in Appendix A.1.

4.3.2. Party V Processing of Message 2

Party V SHALL compose message_2 as follows:

o Generate an ephemeral ECDH key pair as specified in Section 5 of
   [SP-800-56A] using the curve in the cipher suite CIPHER_SUITE_U.
   Let X_V be the x-coordinate of the ephemeral public key.

o Choose a connection identifier C_V and store it for the length of
   the protocol. Party V MUST be able to retrieve the protocol state
   using the connection identifier C_V and optionally other
   information such as the 5-tuple. The connection identifier MAY be
   used with protocols for which EDHOC establishes application keys,
   in which case C_V SHALL be different from the concurrently used
   identifiers of that protocol. To reduce message overhead, party V
   can set the message field C_U in message_2 to null (still storing
   the actual value of C_U) if there is an external correlation
   mechanism (e.g. the Token in CoAP) that enables Party U to
   correlate message_1 and message_2.

o Compute COSE_Sign1 as defined in Section 4.4 of [RFC8152], using
   the signature algorithm in the cipher suite CIPHER_SUITE_U, the
   private authentication key of Party V, and the following
   parameters (further clarifications in Appendix A.2.2).   The
   unprotected header MAY contain parameters (e.g. 'alg').

   * protected = bstr .cbor { abc : ID_CRED_V }

   * payload = CRED_V

   * external_aad = aad_2

   * abc - any COSE map label that can identify a public
      authentication key, see Section 4.1

   * ID_CRED_V - bstr enabling the retrieval of the public
      authentication key of Party V, see Section 4.1

   * CRED_V - bstr credential containing the public authentication
      key of Party V, see Section 4.1

   Note that only 'protected' and 'signature' of the COSE_Sign1
   object are used in message_2, see next bullet.

o Compute COSE_Encrypt0 as defined in Section 5.3 of [RFC8152], with
   the AEAD algorithm in the cipher suite CIPHER_SUITE_U, K_2, IV_2,
   and the following parameters (further clarifications in
   Appendix A.2.2).   The protected header SHALL be empty.   The
   unprotected header MAY contain parameters (e.g. 'alg').

   * plaintext = bstr .cborseq [~protected, signature, ? UAD_2]

   * external_aad = aad_2

   * UAD_2 = bstr containing opaque unprotected application data

   Note that protected and signature in the plaintext are taken from
   the COSE_Sign1 object, and that that only 'ciphertext' of the
   COSE_Encrypt0 object are used in message_2, see next bullet.

o Format message_2 as the sequence of CBOR data items specified in
   Section 4.3.1 and encode it to a byte string (see Appendix A.1).
   CIPHERTEXT_2 is the COSE_Encrypt0 ciphertext.

4.3.3. Party U Processing of Message 2

Party U SHALL process message_2 as follows:

o Decode message_2 (see Appendix A.1).

o Retrieve the protocol state using the connection identifier C_U
   and optionally other information such as the 5-tuple.

o Validate that there is a solution to the curve definition for the
   given x-coordinate X_V.

o Decrypt and verify COSE_Encrypt0 as defined in Section 5.3 of
   [RFC8152], with the AEAD algorithm in the cipher suite
   CIPHER_SUITE_U, K_2, and IV_2.

o Verify COSE_Sign1 as defined in Section 4.4 of [RFC8152], using
   the signature algorithm in the cipher suite CIPHER_SUITE_U and the
   public authentication key of Party V.

If any verification step fails, Party U MUST send an EDHOC error
message back, formatted as defined in Section 6, and the protocol
MUST be discontinued.

4.4. EDHOC Message 3

4.4.1. Formatting of Message 3

message_3 SHALL be a sequence of CBOR data items (see Appendix A.1)
as defined below

```
message_3 = (
  data_3,
  CIPHERTEXT_3 : bstr,
)

data_3 = (
  C_V : bstr,
)

aad_3 : bstr
```

where aad_3, in non-CDDL notation, is:

```
aad_3 = H( bstr .cborseq [ aad_2, CIPHERTEXT_2, data_3 ] )
```

4.4.2. Party U Processing of Message 3

Party U SHALL compose message_3 as follows:

o Compute COSE_Sign1 as defined in Section 4.4 of [RFC8152], using
   the signature algorithm in the cipher suite CIPHER_SUITE_U, the
   private authentication key of Party U, and the following
   parameters. The unprotected header MAY contain parameters (e.g.
   'alg').

   * protected = bstr .cbor { abc : ID_CRED_U }

   * payload = CRED_U

   * external_aad = aad_3

   * abc - any COSE map label that can identify a public
      authentication key, see Section 4.1

   * ID_CRED_U - bstr enabling the retrieval of the public
      authentication key of Party U, see Section 4.1

   * CRED_U - bstr credential containing the public authentication
      key of Party U, see Section 4.1

   Note that only 'protected' and 'signature' of the COSE_Sign1
   object are used in message_3, see next bullet.

o Compute COSE_Encrypt0 as defined in Section 5.3 of [RFC8152], with
   the AEAD algorithm in the cipher suite CIPHER_SUITE_U, K_3, and
   IV_3 and the following parameters. The protected header SHALL be
   empty. The unprotected header MAY contain parameters (e.g.
   'alg').

   * plaintext = bstr .cborseq [~protected, signature, ? PAD_3]

   * external_aad = aad_2

   * PAD_3 = bstr containing opaque protected application data

   Note that protected and signature in the plaintext are taken from
   the COSE_Sign1 object, and that only 'ciphertext' of the
   COSE_Encrypt0 object are used in message_3, see next bullet.

o Format message_3 as the sequence of CBOR data items specified in
   Section 4.4.1 and encode it to a byte string (see Appendix A.1).
   CIPHERTEXT_3 is the COSE_Encrypt0 ciphertext.

o Pass the connection identifiers (C_U, C_V) and the negotiated
   cipher suite CIPHER_SUITE_U to the application. The application
   can now derive application keys using the EDHOC-Exporter
   interface.

4.4.3. Party V Processing of Message 3

Party V SHALL process message_3 as follows:

o Decode message_3 (see Appendix A.1).

o Retrieve the protocol state using the connection identifier C_V
   and optionally other information such as the 5-tuple.

o Decrypt and verify COSE_Encrypt0 as defined in Section 5.3 of
   [RFC8152], with the AEAD algorithm in the cipher suite
   CIPHER_SUITE_U, K_3, and IV_3.

o Verify COSE_Sign1 as defined in Section 4.4 of [RFC8152], using
   the signature algorithm in the cipher suite CIPHER_SUITE_U and the
   public authentication key of Party U.

If any verification step fails, Party V MUST send an EDHOC error
message back, formatted as defined in Section 6, and the protocol
MUST be discontinued.

o Pass PAD_3, the connection identifiers (C_U, C_V), and the
   negotiated cipher suite CIPHER_SUITE_U to the application. The
   application can now derive application keys using the EDHOC-
   Exporter interface.

5. EDHOC Authenticated with Symmetric Keys

5.1. Overview

EDHOC supports authentication with pre-shared keys. Party U and V
are assumed to have a pre-shared key (PSK) with a good amount of
randomness and the requirement that:

o Party V SHALL be able to retrieve the PSK using KID.

KID may optionally contain information about how to retrieve the PSK.
KID does not need to uniquely identify the PSK, but doing so is
recommended as the recipient may otherwise have to try several PSKs.

EDHOC with symmetric key authentication is illustrated in Figure 4.

```
   Party U                                              Party V
      |       C_U, CIPHER_SUITEs_U, CIPHER_SUITE_U, X_U, KID, UAD_1      |
      +----------------------------------------------------------------->|
      |                           message_1                              |
      |                                                                  |
      |                   C_U, C_V, X_V, AE(K_2; UAD_2)                  |
      |<-----------------------------------------------------------------+
      |                           message_2                              |
      |                                                                  |
      |                       C_V, AE(K_3; PAD_3)                        |
      +----------------------------------------------------------------->|
      |                           message_3                              |
```
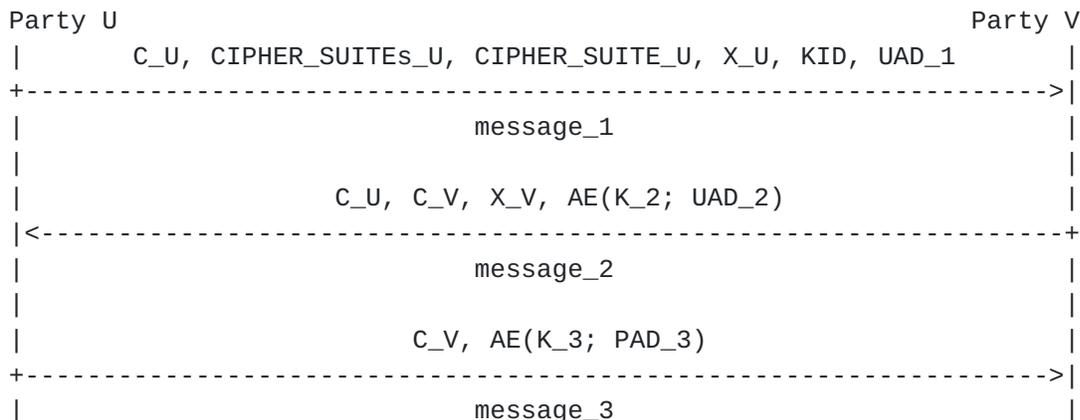
   Figure 4: Overview of EDHOC with symmetric key authentication.

   EDHOC with symmetric key authentication is very similar to EDHOC with
   asymmetric key authentication. In the following subsections the
   differences compared to EDHOC with asymmetric key authentication are
   described.

5.2. EDHOC Message 1

5.2.1. Formatting of Message 1

   message_1 SHALL be a sequence of CBOR data items (see Appendix A.1)
   as defined below

```
   message_1 = (
     MSG_TYPE : int,
     C_U : bstr,
     CIPHER_SUITEs_U : suites,
     CIPHER_SUITE_U : uint,
     X_U : bstr,
     KID : bstr,
     ? UAD_1 : bstr,
   )
```

   where:

   o MSG_TYPE = 2

   o KID - bstr enabling the retrieval of the pre-shared key

5.3. EDHOC Message 2

5.3.1. **Processing of Message 2**

   o COSE_Sign1 is not used.

   o COSE_Encrypt0 is computed as defined in Section 5.3 of [RFC8152],
      with the AEAD algorithm in the cipher suite CIPHER_SUITE_U, K_2,
      IV_2, and the following parameters. The protected header SHALL be
      empty. The unprotected header MAY contain parameters (e.g.
      'alg').

      * external_aad = aad_2

      * plaintext = h'' / UAD_2

      * UAD_2 = bstr containing opaque unprotected application data

5.4. **EDHOC Message 3**

5.4.1. **Processing of Message 3**

   o COSE_Sign1 is not used.

   o COSE_Encrypt0 is computed as defined in Section 5.3 of [RFC8152],
      with the AEAD algorithm in the cipher suite CIPHER_SUITE_U, K_3,
      IV_3, and the following parameters. The protected header SHALL be
      empty. The unprotected header MAY contain parameters (e.g.
      'alg').

      * external_aad = aad_3

      * plaintext = h'' / PAD_3

      * PAD_3 = bstr containing opaque protected application data

6. **Error Handling**

6.1. **EDHOC Error Message**

   This section defines a message format for the EDHOC error message,
   used during the protocol. An EDHOC error message can be send by both
   parties as a response to any non-error EDHOC message. After sending
   an error message, the protocol MUST be discontinued. Errors at the
   EDHOC layer are sent as normal successful messages in the lower
   layers (e.g. CoAP POST and 2.04 Changed). An advantage of using
   such a construction is to avoid issues created by usage of cross
   protocol proxies (e.g. UDP to TCP).

error SHALL be a sequence of CBOR data items (see Appendix A.1) as
defined below

```
error = (
  MSG_TYPE : int,
  ERR_MSG : tstr,
  ? CIPHER_SUITEs_V : suites,
)
```

suites : int / [2* int]

where:

o MSG_TYPE = 0

o ERR_MSG - text string containing the diagnostic payload, defined
   in the same way as in Section 5.5.2 of [RFC7252]

o CIPHER_SUITEs_V - cipher suites from CIPHER_SUITEs_U or the EDHOC
   cipher suites registry that V supports. Note that CIPHER_SUITEs_V
   contains the values from the EDHOC cipher suites registry and not
   indexes.

7. IANA Considerations

7.1. The Well-Known URI Registry

IANA has added the well-known URI 'edhoc' to the Well-Known URIs
registry.

o URI suffix: edhoc

o Change controller: IETF

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

o Related information: None

7.2. Media Types Registry

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

o Type name: application

o Subtype name: edhoc

o Required parameters: N/A

o Optional parameters: N/A

o Encoding considerations: binary

o Security considerations: See Section 7 of this document.

o Interoperability considerations: N/A

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

o Applications that use this media type: To be identified

o Fragment identifier considerations: N/A

o Additional information:

   * Magic number(s): N/A

   * File extension(s): N/A

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

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

o Intended usage: COMMON

o Restrictions on usage: N/A

o Author: See "Authors' Addresses" section.

o Change Controller: IESG

7.3. CoAP Content-Formats Registry

IANA has added the media type 'application/edhoc' to the CoAP
Content-Formats registry.

o Media Type: application/edhoc

o Encoding:

o ID: TBD42

o Reference: [[this document]]

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 perfect forward secrecy, mutual authentication with aliveness, consistency, peer awareness, and identity protection. As described in [SIGMA], peer awareness is provided to Party V, but not to Party U.

EDHOC with asymmetric authentication offers identity protection of Party U against active attacks and identity protection of Party V against passive attacks. The roles should be assigned to protect the most sensitive identity, typically that which is not possible to infer from routing information in the lower layers.

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

EDHOC also adds negotiation 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 Party V to verify that the selected cipher suite is the most preferred cipher suite by U which is supported by both U and V.

8.2. Cryptographic Considerations

The security of the SIGMA protocol requires the MAC to be bound to the identity of the signer. 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. EDHOC implements SIGMA-I using the same Sign-then-MAC approach as TLS 1.3.

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 aliveness, and to protect against interleaving attacks. For this reason, the ephemeral keys MUST NOT be reused, and both parties SHALL generate fresh random ephemeral key pairs.

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. Party U
and V should enforce a minimum security level.

The data rates in many IoT deployments are very limited. Given that
the application keys are protected as well as the long-term
authentication keys they can often be used for years or even decades
before the cryptographic limits are reached. If the application keys
established through EDHOC need to be renewed, the communicating
parties can derive application keys with other labels or run EDHOC
again.

8.3. Mandatory to Implement Cipher Suite

Cipher suite number 1 (AES-CCM-64-64-128, ECDH-SS + HKDF-256, X25519,
Ed25519) is mandatory to implement. For many constrained IoT devices
it is problematic to support more than one cipher suites, so some
deployments with P-256 may not support the mandatory cipher suite.
This is not a problem for local deployments.

8.4. Unprotected Data

Party U and V 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 UAD_1, ID_CRED_V, UAD_2, and ERR_MSG in the asymmetric
case, and KID, UAD_1, and ERR_MSG in the symmetric case. Using the
same KID or UAD_1 in several EDHOC sessions allows passive
eavesdroppers to correlate the different sessions. The communicating
parties may therefore anonymize KID. Another consideration is that
the list of supported cipher suites may be used to identify the
application.

Party U and V must also make sure that unauthenticated data does not
trigger any harmful actions. In particular, this applies to UAD_1
and ERR_MSG in the asymmetric case, and KID, UAD_1, and ERR_MSG in
the symmetric case.

8.5. Denial-of-Service

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 Party V 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 [I-D.ietf-core-echo-request-tag] that forces the

initiator to demonstrate reachability at their apparent network
address.

8.6. Implementation Considerations

The availability of a secure pseudorandom number generator and truly
random seeds are 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. If ECDSA is supported,
"deterministic ECDSA" as specified in RFC6979 is RECOMMENDED.

The referenced processing instructions in [SP-800-56A] must be
complied with, including deleting the intermediate computed values
along with any ephemeral ECDH secrets after the key derivation is
completed. The ECDH shared secret, keys (K_2, K_3), and IVs (IV_2,
IV_3) MUST be secret. Implementations should provide countermeasures
to side-channel attacks such as timing attacks.

Party U and V 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.

Party U and V are allowed to select the connection identifiers C_U
and C_V, respectively, for the other party to use in the ongoing
EDHOC protocol as well as in a subsequent application protocol (e.g.
OSCORE [I-D.ietf-core-object-security]). 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).

8.7. Other Documents Referencing EDHOC

EDHOC has been analyzed in several other documents. A formal
verification of EDHOC was done in [SSR18], an analysis of EDHOC for
certificate enrollment was done in [Kron18], the use of EDHOC in
LoRaWAN is analyzed in [LoRa1] and [LoRa2], the use of EDHOC in IoT
bootstrapping is analyzed in [Perez18], and the use of EDHOC in
6TiSCH is described in [I-D.ietf-6tisch-dtsecurity-zerotouch-join].

9. References

9.1. Normative References

[I-D.ietf-cbor-7049bis]
          Bormann, C. and P. Hoffman, "Concise Binary Object
          Representation (CBOR)", draft-ietf-cbor-7049bis-04 (work
          in progress), October 2018.

[I-D.ietf-cbor-cddl]
          Birkholz, H., Vigano, C., and C. Bormann, "Concise data
          definition language (CDDL): a notational convention to
          express CBOR and JSON data structures", draft-ietf-cbor-
          cddl-06 (work in progress), November 2018.

[I-D.ietf-core-echo-request-tag]
          Amsuess, C., Mattsson, J., and G. Selander, "Echo and
          Request-Tag", draft-ietf-core-echo-request-tag-03 (work in
          progress), October 2018.

[I-D.ietf-core-object-security]
          Selander, G., Mattsson, J., Palombini, F., and L. Seitz,
          "Object Security for Constrained RESTful Environments
          (OSCORE)", draft-ietf-core-object-security-15 (work in
          progress), August 2018.

[I-D.schaad-cose-x509]
          Schaad, J., "CBOR Object Signing and Encryption (COSE):
          Headers for carrying and referencing X.509 certificates",
          draft-schaad-cose-x509-03 (work in progress), December
          2018.

[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>.

[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>.

   [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>.

   [RFC8152] Schaad, J., "CBOR Object Signing and Encryption (COSE)",
              RFC 8152, DOI 10.17487/RFC8152, July 2017,
              <https://www.rfc-editor.org/info/rfc8152>.

   [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>.

   [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>.

9.2. Informative References

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

   [I-D.hartke-core-e2e-security-reqs]
              Selander, G., Palombini, F., and K. Hartke, "Requirements
              for CoAP End-To-End Security", draft-hartke-core-e2e-
              security-reqs-03 (work in progress), July 2017.

   [I-D.ietf-6tisch-dtsecurity-zerotouch-join]
              Richardson, M., "6tisch Zero-Touch Secure Join protocol",
              draft-ietf-6tisch-dtsecurity-zerotouch-join-03 (work in
              progress), October 2018.

   [I-D.ietf-ace-oauth-authz]
              Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., and
              H. Tschofenig, "Authentication and Authorization for
              Constrained Environments (ACE) using the OAuth 2.0
              Framework (ACE-OAuth)", draft-ietf-ace-oauth-authz-17
              (work in progress), November 2018.

   [I-D.ietf-ace-oscore-profile]
              Palombini, F., Seitz, L., Selander, G., and M. Gunnarsson,
              "OSCORE profile of the Authentication and Authorization
              for Constrained Environments Framework", draft-ietf-ace-
              oscore-profile-05 (work in progress), November 2018.

   [I-D.ietf-core-resource-directory]
              Shelby, Z., Koster, M., Bormann, C., Stok, P., and C.
              Amsuess, "CoRE Resource Directory", draft-ietf-core-
              resource-directory-18 (work in progress), December 2018.

   [I-D.ietf-tls-dtls13]
              Rescorla, E., Tschofenig, H., and N. Modadugu, "The
              Datagram Transport Layer Security (DTLS) Protocol Version
              1.3", draft-ietf-tls-dtls13-30 (work in progress),
              November 2018.

   [Kron18]   Krontiris, A., "Evaluation of Certificate Enrollment over
              Application Layer Security", May 2018,
              <https://www.nada.kth.se/~ann/exjobb/
              alexandros_krontiris.pdf>.

   [LoRa1]   Sanchez-Iborra, R., Sanchez-Gomez, J., Perez, S.,
              Fernandez, P., Santa, J., Hernandez-Ramos, J., and A.
              Skarmeta, "Enhancing LoRaWAN Security through a
              Lightweight and Authenticated Key Management Approach",
              June 2018,
              <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6021899/pdf/
              sensors-18-01833.pdf>.

   [LoRa2]   Sanchez-Iborra, R., Sanchez-Gomez, J., Perez, S.,
              Fernandez, P., Santa, J., Hernandez-Ramos, J., and A.
              Skarmeta, "Internet Access for LoRaWAN Devices Considering
              Security Issues", June 2018,
              <https://ants.inf.um.es/~josesanta/doc/GIoTS1.pdf>.

[Perez18]   Perez, S., Garcia-Carrillo, D., Marin-Lopez, R.,
            Hernandez-Ramos, J., Marin-Perez, R., and A. Skarmeta,
            "Architecture of security association establishment based
            on bootstrapping technologies for enabling critical IoT
            infrastructures", October 2018, <http://www.anastacia-
            h2020.eu/publications/Architecture_of_security_association
            _establishment_based_on_bootstrapping_technologies_for_ena
            bling_critical_IoT_infrastructures.pdf>.

[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>.

[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>.

[SSR18]   Bruni, A., Sahl Joergensen, T., Groenbech Petersen, T.,
            and C. Schuermann, "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>.

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

   This Appendix is intended to simplify for implementors not familiar
   with CBOR [I-D.ietf-cbor-7049bis], CDDL [I-D.ietf-cbor-cddl], COSE
   [RFC8152], and HKDF [RFC5869].

A.1. CBOR and CDDL

   The Concise Binary Object Representation (CBOR)
   [I-D.ietf-cbor-7049bis] 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) [I-D.ietf-cbor-cddl] provides
   a way to express structures for protocol messages and APIs that use
   CBOR. [I-D.ietf-cbor-cddl] 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 (e.g. null), byte strings (bstr),

and text strings (tstr), CBOR also supports arrays [] of data items
and maps {} of pairs of data items. Some examples are given below.
For a complete specification and more examples, see
[I-D.ietf-cbor-7049bis] and [I-D.ietf-cbor-cddl]. 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
null                0xf6                    simple value
h'12cd'             0x4212cd                byte string
'12cd'              0x4431326364            byte string
"12cd"              0x6431326364            text string
<< 1, 2, null >>    0x430102f6              byte string
[ 1, 2, null ]      0x830102f6              array
[_ 1, 2, null ]     0x9f0102f6ff            array (indefinite-length)
( 1, 2, null )      0x0102f6                group
{ 4: h'cd' }        0xa10441cd              map
-----------------------------------------------------------------
```

All EDHOC messages consist of a sequence of CBOR encoded data items.
While an EDHOC message in itself is not a CBOR data item, it may be
viewed as the CBOR encoding of an indefinite-length array [_
message_i] without the first byte (0x9f) and the last byte (0xff),
for i = 1, 2 and 3. The same applies to the EDHOC error message.

The message format specification uses the constructs '.cbor',
'.cborseq' and '~' enabling conversion between different CDDL types
matching different CBOR items with different encodings. Some
examples are given below.

A type (e.g. an uint) may be wrapped in a byte string (bstr), and
back again:

```
CDDL Type                       Diagnostic              Encoded
-----------------------------------------------------------------
uint                            24                      0x1818
bstr .cbor uint                 << 24 >>                0x421818
~ bstr .cbor uint               24                      0x1818
-----------------------------------------------------------------
```

A array, say of an uint and a byte string, may be converted into a
byte string (bstr):

```
CDDL Type                         Diagnostic            Encoded
   -------------------------------------------------------------------
   bstr                              h'cd'                 0x41cd
   [ uint, bstr ]                    [ 24, h'cd' ]         0x82181841cd
   bstr .cborseq [ uint, bstr ]      << 24, h'cd' >>       0x44181841cd
   -------------------------------------------------------------------
```

A.2. COSE

CBOR Object Signing and Encryption (COSE) [RFC8152] 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, COSE_Sign1, and COSE_KDF_Context objects.

A.2.1. Encryption and Decryption

The COSE parameters used in COSE_Encrypt0 (see Section 5.2 of
[RFC8152]) are constructed as described below. Note that "i" in
"K_i", "IV_i" and "aad_i" is a variable with value i = 2 or 3,
depending on whether the calculation is made over message_2 or
message_3.

o The secret key K_i is a CBOR bstr, generated with the EDHOC-Key-
   Derivation function as defined in Section 3.3.

o The initialization vector IV_i is a CBOR bstr, also generated with
   the EDHOC-Key-Derivation function as defined in Section 3.3.

o The plaintext is a CBOR bstr. If the application data (UAD and
   PAD) is omitted, then plaintext = h'' in the symmetric case, and
   plaintext = << ~protected, signature >> in the asymmetric case.
   For instance, if protected = h'a10140' and signature = h'050607'
   (CBOR encoding 0x43050607), then plaintext = h'a1014043050607'.

o The external_aad is a CBOR bstr. It is always set to aad_i.

COSE constructs the input to the AEAD [RFC5116] as follows:

o The key K is the value of the key K_i.

o The nonce N is the value of the initialization vector IV_i.

o The plaintext P is the value of the COSE plaintext. E.g. if the
   COSE plaintext = h'010203', then P = 0x010203.

o The associated data A is the CBOR encoding of:

      ["Encrypt0", h'', aad_i]

      This equals the concatenation of 0x8368456e63727970743040 and the
      CBOR encoding of aad_i. For instance if aad_2 = h'010203' (CBOR
      encoding 0x43010203), then A = 0x8368456e6372797074304043010203.

A.2.2. Signing and Verification

   The COSE parameters used in COSE_Sign1 (see Section 4.2 of [RFC8152])
   are constructed as described below. Note that "i" in "aad_i" is a
   variable with values i = 2 or 3, depending on whether the calculation
   is made over message_2 or message_3. Note also that "x" in
   "ID_CRED_x" and "CRED_x" is a variable with values x = U or V,
   depending on whether it is the credential of U or of V that is used
   in the relevant protocol message.

   o The key is the private authentication key of U or V. This may be
      stored as a COSE_KEY object or as a certificate.

   o The protected parameter is a map { abc : ID_CRED_x } wrapped in a
      byte string.

   o The payload is a bstr containing the CBOR encoding of a COSE_KEY
      or a single certificate.

   o external_aad = aad_i.

   COSE constructs the input to the Signature Algorithm as follows:

   o The key is the private authentication key of U or V.

   o The message to be signed M is the CBOR encoding of:

      ["Signature1", << { abc : ID_CRED_x } >>, aad_i, CRED_x]

      For instance if abc = 4 (CBOR encoding 0x04), ID_CRED_U = h'1111'
      (CBOR encoding 0x421111), aad_3 = h'222222' (CBOR encoding
      0x43222222), and CRED_U = h'55555555' (CBOR encoding
      0x4455555555), then M =
      0x846a5369676e61747572653145A1044211114322222224455555555.

A.2.3. Key Derivation

   Assuming use of the mandatory-to-implement algorithms HKDF SHA-256
   and AES-CCM-16-64-128, the extract phase of HKDF produces a
   pseudorandom key (PRK) as follows:

   PRK = HMAC-SHA-256(salt, ECDH shared secret)

   where salt = 0x in the asymmetric case and salt = PSK in the
   symmetric case. As the output length L is smaller than the hash
   function output size, the expand phase of HKDF consists of a single
   HMAC invocation, and K_i and IV_i are therefore the first 16 and 13
   bytes, respectively, of

   output parameter = HMAC-SHA-256(PRK, info || 0x01)

   where || means byte string concatenation, and info is the CBOR
   encoding of

```
   COSE_KDF_Context = [
     AlgorithmID,
     [ null, null, null ],
     [ null, null, null ],
     [ keyDataLength, h'', aad_i ]
   ]
```

   If AES-CCM-16-64-128 then AlgorithmID = 10 and keyDataLength = 128
   for K_i, and AlgorithmID = "IV-GENERATION" (CBOR encoding
   0x6d49562d47454e45524154494f4e) and keyDataLength = 104 for IV_i.
   Hence, if aad_2 = h'aaaa' then

```
   K_2  = HMAC-SHA-256( PRK, 0x840a83f6f6f683f6f6f68318804042aaaa01 )
   IV_2 = HMAC-SHA-256( PRK, 0x846d49562d47454e45524154494f4e
                                  83f6f6f683f6f6f68318804042aaaa01 )
```

[Appendix B](#). Test Vectors

   This appendix provides a wealth of test vectors to ease
   implementation and ensure interoperability.

   TODO: This section needs to be updated.

[Appendix C](#). EDHOC PSK Chaining

   An application using EDHOC may want to derive new PSKs to use for
   authentication in future EDHOC sessions. In this case, the new PSK
   and KID SHOULD be derived as follows where length is the key length
   (in bytes) of the AEAD Algorithm.

```
   PSK = EDHOC-Exporter("EDHOC Chaining PSK", length)
   KID = EDHOC-Exporter("EDHOC Chaining KID", 4)
```

Appendix D. EDHOC with CoAP and OSCORE

D.1. Transferring EDHOC in CoAP

   EDHOC can be transferred as an exchange of CoAP [RFC7252] messages.
   By default, the CoAP client is Party U and the CoAP server is Party
   V, but the roles SHOULD be chosen to protect the most sensitive
   identity, see Section 8. By default, EDHOC is transferred in POST
   requests and 2.04 (Changed) responses to the Uri-Path: "/.well-known/
   edhoc", but 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.

   An example of a successful EDHOC exchange using CoAP is shown in
   Figure 5.

```
          Client     Server
            |          |
            +--------->| Header: POST (Code=0.02)
            |   POST   | Uri-Path: "/.well-known/edhoc"
            |          | Content-Format: application/edhoc
            |          | Payload: 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"
            |          | Content-Format: application/edhoc
            |          | Payload: EDHOC message_3
            |          |
            |<---------+ Header: 2.04 Changed
            |   2.04   |
            |          |
```
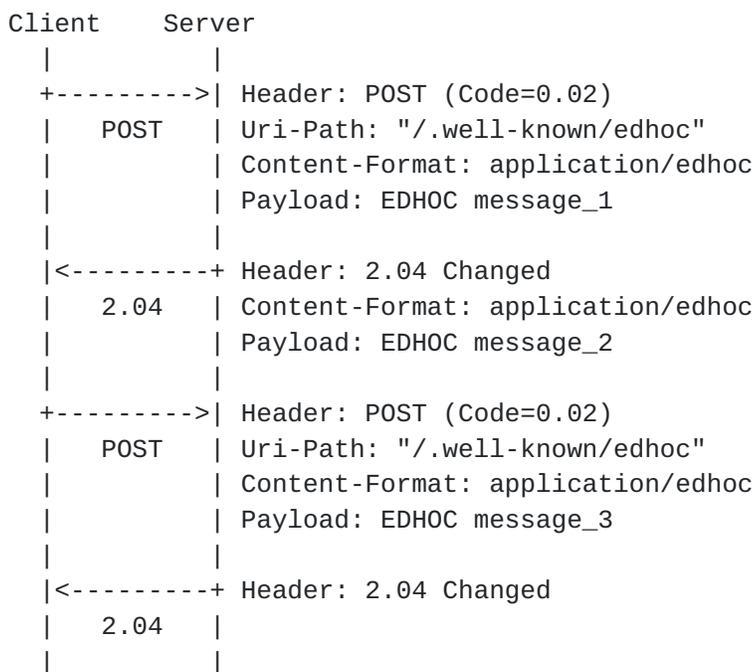
            Figure 5: Example of transferring EDHOC in CoAP

D.2. Deriving an OSCORE context from EDHOC

When EDHOC is used to derive parameters for OSCORE
[I-D.ietf-core-object-security], the parties must make sure that the
EDHOC connection identifiers are unique, i.e. C_V MUST NOT be equal
to C_U. In case that the CoAP client is party U and the CoAP server
is party V:

o The client's OSCORE Sender ID is C_V and the server's OSCORE
   Sender ID is C_U, as defined in this document

o The AEAD Algorithm and the HMAC-based Key Derivation Function
   (HKDF) are the AEAD and HKDF algorithms in the cipher suite
   CIPHER_SUITE_U.

o The Master Secret and Master Salt are derived as follows where
   length is the key length (in bytes) of the AEAD Algorithm.

   Master Secret = EDHOC-Exporter("OSCORE Master Secret", length)
   Master Salt   = EDHOC-Exporter("OSCORE Master Salt", 8)

Appendix E. Message Sizes

This appendix gives an estimate of the message sizes of EDHOC with
different authentication methods. Note that the examples in this
appendix are not test vectors, the cryptographic parts are just
replaced with byte strings of the same length. All examples are
given in CBOR diagnostic notation and hexadecimal.

E.1. Message Sizes RPK

E.1.1. message_1

```
message_1 = (
  1,
  h'c3',
  0,
  0,
  h'000102030405060708090a0b0c0d0e0f10111213141516171819181a1b1c1d
    1e1f'
)

message_1 (39 bytes):
01 41 C3 00 00 58 20 00 01 02 03 04 05 06 07 08 09 0A 0B 0C
0D 0E 0F 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F
```

E.1.2. message_2

```
plaintext = <<
  { 4 : 'acdc' },
  h'000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d
    1e1f202122232425262728292a2b2c2d2e2f303132333435363738393a3b
    3c3d3e3f'
>>
```

The protected header map is 7 bytes. The length of plaintext is 73
bytes so assuming a 64-bit MAC value the length of ciphertext is 81
bytes.

```
message_2 = (
  null,
  h'c4',
  h'000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d
    1e1f',
  h'000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d
    1e1f202122232425262728292a2b2c2d2e2f303132333435363738393a3b
    3c3d3e3f404142434445464748494a4b4c4d4e4f50'
)
```

```
message_2 (120 bytes):
F6 41 C4 58 20 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E
0F 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 58 51 00
01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14
15 16 17 18 19 1A 1B 1C 1D 1E 1F 20 21 22 23 24 25 26 27 28
29 2A 2B 2C 2D 2E 2F 30 31 32 33 34 35 36 37 38 39 3A 3B 3C
3D 3E 3F 40 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F 50
```

E.1.3. message_3

The plaintext and ciphertext in message_3 are assumed to be of equal
sizes as in message_2.

```
message_3 = (
  h'c3',
  h'000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d
    1e1f202122232425262728292a2b2c2d2e2f303132333435363738393a3b
    3c3d3e3f404142434445464748494a4b4c4d4e4f50'
)
```

```
message_3 (85 bytes):
41 C3 58 51 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20 21 22 23
24 25 26 27 28 29 2A 2B 2C 2D 2E 2F 30 31 32 33 34 35 36 37
38 39 3A 3B 3C 3D 3E 3F 40 41 42 43 44 45 46 47 48 49 4A 4B
4C 4D 4E 4F 50
```

E.2. Message Sizes Certificates

When the certificates are distributed out-of-band and identified with
the x5t header and a SHA256/64 hash value, the protected header map
will be 13 bytes instead of 7 bytes (assuming labels in the range
-24...23).

```
protected = << { TDB1 : [ TDB6, h'0001020304050607' ] } >>
```

When the certificates are identified with the x5chain header, the
message sizes depends on the size of the (truncated) certificate
chains. The protected header map will be 3 bytes + the size of the
certificate chain (assuming a label in the range -24...23).

```
protected = << { TDB3 : h'0001020304050607...' } >>
```

E.3. Message Sizes PSK

E.3.1. message_1

```
message_1 = (
  2,
  h'c3',
  0,
  0,
  h'000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d
    1e1f',
  'abba'
)
```

```
message_1 (44 bytes):
02 41 C3 00 00 58 20 00 01 02 03 04 05 06 07 08 09 0A 0B 0C
0D 0E 0F 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F
44 61 63 64 63
```

E.3.2. message_2

Assuming a 0 byte plaintext and a 64-bit MAC value the ciphertext is
8 bytes

```
message_2 = (
  null,
  h'c4',
  h'000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d
    1e1f',
  h'0001020304050607'
)

message_2 (46 bytes):
F6 41 C4 58 20 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E
0F 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 48 61 62
63 64 65 66 67 68
```

E.3.3. message_3

The plaintext and ciphertext in message_3 are assumed to be of equal
sizes as in message_2.

```
message_3 = (
  h'c3',
  h'0001020304050607'
)

message_3 (11 bytes):
41 C3 48 00 01 02 03 04 05 06 07
```

E.4. Summary

The previous estimates of typical message sizes are summarized in
Figure 6.

```
=======================================================================
             PSK        RPK        x5t       x5chain
-----------------------------------------------------------------------
message_1       44         39         39         39
message_2       46        120        126        116 + Certificate chain
message_3       11         85         91         81 + Certificate chain
-----------------------------------------------------------------------
Total          101        244        256        236 + Certificate chains
=======================================================================
```

Figure 6: Typical message sizes in bytes

Figure 7 compares of message sizes of EDHOC with the DTLS 1.3
handshake [I-D.ietf-tls-dtls13] with connection ID.

```
=====================================================================
Flight                          #1          #2          #3      Total
---------------------------------------------------------------------
DTLS 1.3 RPK + ECDHE            149         373         213        735
DTLS 1.3 PSK + ECDHE           186         190          57        433
DTLS 1.3 PSK                    136         150          57        343
---------------------------------------------------------------------
EDHOC RPK + ECDHE               39         120          85        244
EDHOC PSK + ECDHE               44          46          11        101
=====================================================================
```

   Figure 7: Comparison of message sizes in bytes with Connection ID

   Figure 8 compares of message sizes of EDHOC with the DTLS 1.3
   [I-D.ietf-tls-dtls13] and TLS 1.3 [RFC8446] handshakes without
   connection ID.

```
=====================================================================
Flight                          #1          #2          #3      Total
---------------------------------------------------------------------
DTLS 1.3 RPK + ECDHE            143         364         212        721
DTLS 1.3 PSK + ECDHE           180         183          56        419
DTLS 1.3 PSK                    130         143          56        329
---------------------------------------------------------------------
TLS 1.3  RPK + ECDHE           129         322         194        645
TLS 1.3  PSK + ECDHE           166         157          50        373
TLS 1.3  PSK                    116         117          50        283
---------------------------------------------------------------------
EDHOC RPK + ECDHE               38         119          84        241
EDHOC PSK + ECDHE               44          45          10         98
=====================================================================
```

   Figure 8: Comparison of message sizes in bytes without Connection ID

Acknowledgments

Authors' Addresses

    Goeran Selander
    Ericsson AB

    Email: goran.selander@ericsson.com

    John Mattsson
    Ericsson AB

    Email: john.mattsson@ericsson.com

    Francesca Palombini
    Ericsson AB

    Email: francesca.palombini@ericsson.com