ACE Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 12, 2019

G. Selander
J. Mattsson
F. Palombini
Ericsson AB

March 11, 2019

Ephemeral Diffie-Hellman Over COSE (EDHOC)
draft-selander-ace-cose-ecdhe-13

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, perfect 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 footprint can be kept very low.

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 protection needs to work over 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]. Object Security for Constrained RESTful Environments (OSCORE) [I-D.ietf-core-object-security] is a method for application-layer protection of the Constrained Application Protocol (CoAP), using COSE.

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. 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. A main use case for EDHOC is to establish an OSCORE security context. EDHOC uses COSE for cryptography, CBOR for encoding, and CoAP for transport. By reusing existing libraries, the additional code footprint can be kept very low. 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], LoRaWAN [LoRa1][LoRa2]. These network technologies are characterized by their low throughput, low power consumption, and small frame sizes. 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 B.

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. The use of COSE provides crypto agility and enables use of future algorithms and headers designed for constrained IoT.

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 Section 7 describes how EDHOC can be transferred in CoAP and used to establish an OSCORE security context.

1.1. Rationale for EDHOC

Many constrained IoT systems today do not use any security at all, and when they do, they often do not follow best practices. One reason is that many current security protocols are not designed with constrained IoT in mind. Constrained IoT systems often deal with personal information, valuable business data, and actuators interacting with the physical world. Not only do such systems need security and privacy, they often need end-to-end protection with source authentication and perfect forward secrecy. EDHOC and OSCORE [I-D.ietf-core-object-security] enables security following current best practices to devices and systems where current security protocols are impractical.

EDHOC is optimized for small message sizes and can therefore be sent over a small number of radio frames. The message size of a key exchange protocol may have a large impact on the performance of an IoT deployment, especially in noisy environments. 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 in constrained environments can be translated into key exchange overhead. In networks technologies with transmission back-off time,

each additional frame significantly increases the latency even if no other devices are transmitting.

Power consumption for wireless devices is highly dependent on message transmission, listening, 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 reasons.

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. EDHOC is not bound to a particular transport layer, but it is recommended to transport the EDHOC message in CoAP payloads. 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. Maximum code complexity (ROM/Flash) is often a constraint in many devices and by reusing already existing libraries, the additional code footprint for EDHOC + OSCORE can be kept very low.

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].

Readers are expected to be familiar with the terms and concepts described in CBOR [I-D.ietf-cbor-7049bis], COSE [RFC8152], and CDDL [I-D.ietf-cbor-cddl]. The Concise Data Definition Language (CDDL) is used to express CBOR data structures [I-D.ietf-cbor-7049bis]. 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 [RFC8446], 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, AEAD( K_2; ID_CRED_V, Sig(V; CRED_V, X_U, X_V) )  |
      |<------------------------------------------------------+
      |                                                       |
      |     AEAD( 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 AEAD(K; .) denotes authenticated encryption with additional data using the key K derived from the shared secret. The authenticated encryption MUST NOT be replaced by plain encryption, see Section 9.

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 Transcript hashes TH_2, TH_3, TH_4 used for key derivation and as additional authenticated data.
- 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 Method 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, COSE, and CoAP libraries.

To simplify for implementors, the use of CBOR and COSE in EDHOC is summarized in Appendix A and example messages in CBOR diagnostic notation are given in Appendix B.

3. EDHOC Overview

EDHOC consists of three flights (message_1, message_2, message_3) that maps directly to the three messages in SIGMA-I, plus an EDHOC error message. All EDHOC messages consist of a sequence of CBOR encoded data items, where the first data item of message_1 is an int (TYPE) specifying the method (asymmetric, symmetric, error) and the correlation properties of the transport used. 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 is 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 8.

```
   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, and that session keys are derived from the ECDH shared secret and the PSK.

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 and can be e.g. be used to transfer access tokens that are protected outside of EDHOC. Protected application data (PAD_3) may be used to transfer any application data in message_3.

Cryptographically, EDHOC does not put requirements on the lower layers. EDHOC is not bound to a particular transport layer, and can be used in environments without IP. It is recommended to transport the EDHOC message in CoAP payloads, see Section 7. An implementation may support only Party U or only Party V.

3.1. Cipher Suites

EDHOC cipher suites consist of a set of COSE algorithms: an AEAD algorithm, an ECDH algorithm (including HKDF algorithm), an ECDH curve, a signature algorithm, and signature algorithm parameters. The signature algorithm is not used when EDHOC is authenticated with symmetric keys. Each cipher suite is either identified with a pre-

defined int label or with an array of labels and values from the COSE Algorithms and Elliptic Curves registries.

```
suite = int / [ 4*4 algs: int / tstr, ? para: any ]
```

This document specifies two pre-defined cipher suites.

0. [12, -27, 4, -8, 6]
   (AES-CCM-64-64-128, ECDH-SS + HKDF-256, X25519, EdDSA, Ed25519)

1. [12, -27, 1, -7]
   (AES-CCM-64-64-128, ECDH-SS + HKDF-256, P-256, ES256)

Two additional numbers are registered for application defined cipher suites. Application defined cipher suites MUST only use algorithms specified for COSE, are not interoperable with other deployments and can therefore only be used in local networks.

-24. First application defined cipher suite.
-23. Second application defined cipher suite.

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 is 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 selected cipher suite.

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. The secret and the salt are both inputs to the HKDF extract stage and the PSK is used as 'salt' to simplify implementation. 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 one of the transcript hashes TH_2, TH_3, or TH_4 as defined in Sections 4.3.1, 4.4.1, and 3.3.1.

   * SuppPrivInfo is omitted

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_2 and message_3, the keys K_2 and K_3 SHALL be derived using other set to the transcript hashes TH_2 and TH_3 respectively. The key SHALL be derived using AlgorithmID set to the integer value of the AEAD in the selected cipher suite, and keyDataLength equal to the key length of the AEAD.

If the AEAD algorithm uses an IV, then IV_2 and IV_3 for message_2 and message_3 SHALL be derived using other set to the transcript hashes TH_2 and TH_3 respectively. 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, TH_4)
```

The output of the EDHOC-Exporter function SHALL be derived using other = TR_4, 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. The transcript hash TR_4, in non-CDDL notation, is:

```
TH_4 = H( bstr .cborseq [ TH_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. Example use of the EDHOC-Exporter is given in Sections 3.3.2 and 7.1.1.

3.3.2. EDHOC PSK Chaining

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

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

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:

- Only Party V SHALL have access to the private authentication key of Party V,

- Only Party U SHALL have access to the private authentication key of Party U,

- Party U is able to retrieve Party V's public authentication key using ID_CRED_V,

- Party V is able to retrieve Party U's public authentication key using ID_CRED_U,

where the identifiers ID_CRED_U and ID_CRED_V are COSE header maps containing COSE header parameter that can identify a public authentication key, see Appendix A.2. In the following we give some examples of possible COSE header parameters.

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

o ID_CRED_x = { 4 : bstr }, for x = U or V.

Public key certificates can be identified in different ways. Several header parameters for identifying X.509 certificates are defined in [I-D.schaad-cose-x509] (the exact labels are TBD):

o by a hash value with the 'x5t' parameter;

   * ID_CRED_x = { TBD1 : COSE_CertHash }, for x = U or V,

o by a URL with the 'x5u' parameter;

   * ID_CRED_x = { TBD2 : uri }, for x = U or V,

o or by a bag of certificates with the 'x5bag' parameter;

   * ID_CRED_x = { TBD3 : COSE_X509 }, for x = U or V.

o by a certificate chain with the 'x5chain' parameter;

   * ID_CRED_x = { TBD4 : COSE_X509 }, for x = U or V,

In the latter two examples, ID_CRED_U and ID_CRED_V contain the actual credential used for authentication. The purpose of ID_CRED_U and ID_CRED_V is to facilitate retrieval of a public authentication key and when they do not contain the actual credential, they may be very short. It is RECOMMENDED that they uniquely identify the public authentication key as the recipient may otherwise have to try several 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 to prevent duplicate-signature key selection (DSKS) attacks, see Section 4.4.1 and Section 4.3.1. Party U and Party V MAY use different types of credentials, e.g. one uses RPK and the other uses certificate.

The connection identifiers C_U and C_V do not have any cryptographic purpose in EDHOC. They contain information facilitating retrieval of the protocol state and may therefore be very short. The connection identifier MAY be used with an application protocol (e.g. OSCORE) for which EDHOC establishes keys, in which case the connection identifiers SHALL adhere to the requirements for that protocol. Each

party choses a connection identifier it desires the other party to use in outgoing messages.

The first data item of message_1 is an int TYPE = 4 * method + corr specifying the method and the correlation properties of the transport used. corr = 0 is used when there is no external correlation mechanism. corr = 1 is used when there is an external correlation mechanism (e.g. the Token in CoAP) that enables Party U to correlate message_1 and message_2. corr = 2 is used when there is an external correlation mechanism that enables Party V to correlate message_2 and message_3. corr = 3 is used when there is an external correlation mechanism that enables the parties to correlate all the messages. The use of the correlation parameter is exemplified in Section 7.1.

EDHOC with asymmetric key authentication is illustrated in Figure 3.

```
Party U                                                       Party V
|                  TYPE, SUITES_U, X_U, C_U, UAD_1                  |
+------------------------------------------------------------------>|
|                             message_1                             |
|                                                                   |
|  C_U, X_V, C_V, AEAD(K_2; ID_CRED_V, Sig(V; CRED_V, TH_2), UAD_2) |
|<------------------------------------------------------------------+
|                             message_2                             |
|                                                                   |
|       C_V, AEAD(K_3; ID_CRED_U, Sig(U; CRED_U, TH_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 = (
  TYPE : int,
  SUITES_U : suite / [ index: uint, 2* suite ],
  X_U : bstr,
  C_U : bstr,
  ? UAD_1 : bstr,
)
```

where:

- TYPE = 4 * method + corr, where the method = 0 and the connection parameter corr is chosen based on the transport and determines which connection identifiers that are omitted (see Section 4.1).

- SUITES_U - cipher suites which Party U supports, in order of decreasing preference. If a single cipher suite is conveyed, a single suite is used, if multiple cipher suites are conveyed, an array of suites and an index is used. The zero-based index (i.e. 0 for the first, 1 for the second, etc.) identifies a single selected cipher suite from the array.

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

- C_U - variable length connection identifier

- 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:

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

- Determine the cipher suite 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 SUITES_U MUST be used.

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

- Choose a connection identifier C_U and store it for the length of the protocol.

- 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 selected cipher suite is supported and that no prior cipher suites in 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 selected cipher suite, then SUITES_V MUST include one or more supported cipher suites. If V does not support the selected cipher suite, but supports another cipher suite in SUITES_U, then SUITES_V MUST include the first supported cipher suite in 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,
  X_V : bstr,
  C_V : bstr,
)

TH_2 : bstr
```

where the transcript hash TH_2, in non-CDDL notation, is:

```
TH_2 = H( bstr .cborseq [ message_1, data_2 ] )
```

where:

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

o C_V - variable length connection identifier

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 If TYPE mod 4 equals 1 or 3, C_U is omitted, otherwise C_U is not omitted.

o Generate an ephemeral ECDH key pair as specified in Section 5 of [SP-800-56A] using the curve in the selected cipher suite. 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.

o Compute COSE_Sign1 as defined in Section 4.4 of [RFC8152], using the signature algorithm in the selected cipher suite, the private authentication key of Party V, and the following parameters (further clarifications in Appendix A.2.2). The unprotected header (not included in the EDHOC message) MAY contain parameters (e.g. 'alg').

  * protected = bstr .cbor ID_CRED_V

  * payload = CRED_V

  * external_aad = TH_2

  * ID_CRED_V - identifier to facilitate retrieval of a 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 selected cipher suite, K_2, IV_2, and the following parameters (further clarifications in Appendix A.2.2). The protected header SHALL be empty. The

unprotected header (not included in the EDHOC message) MAY contain parameters (e.g. 'alg').

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

* external_aad = TH_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 only 'ciphertext' of the COSE_Encrypt0 object are used in message_2, see next bullet. If ID_CRED_V contains a single 'kid' parameter, i.e., ID_CRED_V = { 4 : bstr }, only the bstr is conveyed in the plaintext, in CDDL notation

plaintext = bstr .cborseq [bstr / header_map, bstr, ? bstr]

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/or other external information such as the CoAP Token and 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 selected cipher suite, K_2, and IV_2.

o Verify COSE_Sign1 as defined in Section 4.4 of [RFC8152], using the signature algorithm in the selected cipher suite 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,
)

TH_3 : bstr
```

where the transcript hash TH_3, in non-CDDL notation, is:

```
TH_3 = H( bstr .cborseq [ TH_2, CIPHERTEXT_2, data_3 ] )
```

4.4.2. Party U Processing of Message 3

Party U SHALL compose message_3 as follows:

- o If TYPE mod 4 equals 2 or 3, C_V is omitted, otherwise C_V is not omitted.

- o Compute COSE_Sign1 as defined in Section 4.4 of [RFC8152], using the signature algorithm in the selected cipher suite, the private authentication key of Party U, and the following parameters. The unprotected header (not included in the EDHOC message) MAY contain parameters (e.g. 'alg').

  * protected = bstr .cbor ID_CRED_U

  * payload = CRED_U

  * external_aad = TH_3

  * ID_CRED_U - identifier to facilitate retrieval of a 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 selected cipher suite, K_3, and IV_3 and the following parameters. The protected header SHALL be empty. The unprotected header (not included in the EDHOC message) MAY contain parameters (e.g. 'alg').

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

  * external_aad = TH_3

  * 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. If ID_CRED_U contains a single 'kid' parameter, i.e., ID_CRED_U = { 4 : bstr }, only the bstr is conveyed in the plaintext.

- 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 selected cipher suite 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/or other external information such as the CoAP Token and the 5-tuple.

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

- o Verify COSE_Sign1 as defined in Section 4.4 of [RFC8152], using the signature algorithm in the selected cipher suite 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 selected cipher suite 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 Only Party U and Party V SHALL have access to the PSK,

o Party V is able to retrieve the PSK using ID_PSK.

where the identifier ID_PSK is a COSE header maps containing COSE header parameter that can identify a pre-shared key. Pre-shared keys are typically stored as COSE_Key objects and identified with a 'kid' parameter (see [RFC8152]):

o ID_PSK = { 4 : bstr }

The purpose of ID_PSK is to facilitate retrieval of the PSK and in the case a 'kid' parameter is used it may be very short. It is RECOMMENDED that it uniquely identify the PSK as the recipient may otherwise have to try several keys.

EDHOC with symmetric key authentication is illustrated in Figure 4.

```
Party U                                                       Party V
|              TYPE, SUITES_U, X_U, C_U, ID_PSK, UAD_1              |
+------------------------------------------------------------------>|
|                             message_1                             |
|                                                                   |
|               C_U, X_V, C_V, AEAD(K_2; TH_2, UAD_2)               |
|<------------------------------------------------------------------+
|                             message_2                             |
|                                                                   |
|                    C_V, AEAD(K_3; TH_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 = (
  TYPE : int,
  SUITES_U : suite / [ index: uint, 2* suite ],
  X_U : bstr,
  C_U : bstr,
  ID_PSK : bstr / header_map,
  ? UAD_1 : bstr,
)
```

where:

- o TYPE = 4 * method + corr, where the method = 1 and the connection parameter corr is chosen based on the transport and determines which connection identifiers that are omitted (see Section 4.1).

- o ID_PSK - identifier to facilitate retrieval of the pre-shared key. If ID_PSK contains a single 'kid' parameter, i.e., ID_PSK = { 4 : bstr }, only the bstr used.

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 selected cipher suite, 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 = TH_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 selected cipher suite, 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 = TH_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 = (
  TYPE : int,
  ERR_MSG : tstr,
  ? SUITES_V : suite / [ 2* suite ],
)
```

where:

- o TYPE = -1

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

o SUITES_V - cipher suites from SUITES_U or the EDHOC cipher suites registry that V supports. Note that SUITEs_V only contains the values from the EDHOC cipher suites registry and no index.

6.1.1. Example Use of EDHOC Error Message with SUITES_V

Assuming that Party U supports the five cipher suites {5, 6, 7, 8, 9} in decreasing order of preference, Figures 5 and 6 show examples of how Party U can truncate SUITES_U and how SUITES_V is used by Party V to give Party U information about the cipher suites that Party V supports. In Figure 5, Party V supports cipher suite 6 but not cipher suite 5.

```
Party U                                                       Party V
|             TYPE, SUITES_U {0, 5, 6, 7}, X_U, C_U, UAD_1           |
+------------------------------------------------------------------->|
|                              message_1                             |
|                                                                    |
|                     TYPE, ERR_MSG, SUITES_V {6}                    |
|<-------------------------------------------------------------------+
|                                error                               |
|                                                                    |
|              TYPE, SUITES_U {1, 5, 6}, X_U, C_U, UAD_1             |
+------------------------------------------------------------------->|
|                              message_1                             |
```

Figure 5: Example use of error message with SUITES_V.

In Figure 6, Party V supports cipher suite 7 but not cipher suites 5 and 6.

```
Party U                                                       Party V
|              TYPE, SUITES_U {0, 5, 6}, X_U, C_U, UAD_1             |
+------------------------------------------------------------------->|
|                              message_1                             |
|                                                                    |
|                    TYPE, ERR_MSG, SUITES_V {7, 9}                  |
|<-------------------------------------------------------------------+
|                                error                               |
|                                                                    |
|             TYPE, SUITES_U {2, 5, 6, 7}, X_U, C_U, UAD_1           |
+------------------------------------------------------------------->|
|                              message_1                             |
```

Figure 6: Example use of error message with SUITES_V.

As Party U's list of supported cipher suites and order of preference is fixed, and Party V only accepts message_1 if the selected cipher

suite is the first cipher suite in SUITES_U that Party V supports, the parties can verify that the selected cipher suite is the most preferred (by Party U) cipher suite supported by both parties. If the selected cipher suite is not the first cipher suite in SUITES_U that Party V supports, Party V will discontinue the protocol.

7. Transferring EDHOC and Deriving Application Keys

7.1. Transferring EDHOC in CoAP

It is recommended is to transport EDHOC as an exchange of CoAP [RFC7252] messages. CoAP is a reliable transport that can preserve packet ordering and handle message duplication. CoAP can also perform fragmentation and protect against denial of service attacks. It is recommended to carry the EDHOC flights in Confirmable messages, especially if fragmentation is used.

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 9. 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 7. In this case the CoAP Token enables Party U to correlate message_1 and message_2 so the correlation parameter corr = 1.

```
          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 7: Transferring EDHOC in CoAP

The exchange in Figure 7 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 8. In this case the CoAP Token enables Party V to correlate message_2 and message_3 so the correlation parameter corr = 2.

```
          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"
            |          | Content-Format: application/edhoc
            |          | Payload: EDHOC message_2
            |          |
            |<---------+ Header: 2.04 Changed
            |   2.04   | Content-Format: application/edhoc
            |          | Payload: EDHOC message_3
            |          |
```

Figure 8: Transferring EDHOC in CoAP

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 [I-D.ietf-core-echo-request-tag]. 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].

7.1.1. 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. The CoAP client and server MUST be able to retrieve the OCORE protocol state using its chosen connection identifier and optionally other information such as the 5-tuple. 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 selected cipher suite.

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)
```

7.2. Transferring EDHOC over Other Protocols

EDHOC may be transported over a different transport than CoAP. In this case the lower layers need to handle message loss, reordering, message duplication, fragmentation, and denial of service protection.

8. IANA Considerations

8.1. EDHOC Cipher Suites Registry

IANA has created a new registry titled "EDHOC Cipher Suites".

TODO

8.2. EDHOC Method Type Registry

IANA has created a new registry titled "EDHOC Method Type".

TODO

8.3. The Well-Known URI Registry

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

- URI suffix: edhoc
- Change controller: IETF
- Specification document(s): [[this document]]
- Related information: None

8.4. 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

- 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

8.5. 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]]

9. Security Considerations

9.1. 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 also inherits Key Compromise Impersonation (KCI) resistance from SIGMA-I.

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 method 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.

As required by [RFC7258], IETF protocols need to mitigate pervasive monitoring when possible. One way to mitigate pervasive monitoring is to use a key exchange that provides perfect forward secrecy. EDHOC therefore only supports methods with perfect forward secrecy. 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 (PSK or private authentication keys) does not compromise the security of completed EDHOC exchanges. Compromising the private authentication keys of one party lets the 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. Compromising the PSK lets the attacker impersonate Party U in EDHOC exchanges with Party V and impersonate Party V in EDHOC exchanges with Party U. Compromise of the HDKF input parameters (ECDH shared secret and/or PSK) leads to compromise of all session keys derived from that compromised shared secret. Compromise of one session key does not compromise other session keys.

9.2. 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 liveness, 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.

9.3. Mandatory to Implement Cipher Suite

Cipher suite number 0 (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.

9.4. 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 ID_PSK, UAD_1, and ERR_MSG in the symmetric case. Using the same ID_PSK or UAD_1 in several EDHOC sessions allows passive eavesdroppers to correlate the different sessions. The communicating parties may therefore anonymize ID_PSK. 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 ID_PSK, UAD_1, and ERR_MSG in the symmetric case.

9.5. 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 its apparent network address.

9.6. 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 and the PSK (even though it is used as salt) 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).

Party V MUST finish the verification step of message_3 before passing PAD_3 to the application.

9.7. 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].

10. References

10.1. Normative References

[I-D.ietf-cbor-7049bis]
Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", draft-ietf-cbor-7049bis-05 (work in progress), January 2019.

[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-07 (work in progress), February 2019.

[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>.

[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>.

[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>.

[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>.

10.2. 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-22 (work in progress), March 2019.

[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-07 (work in progress), February 2019.

[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-19 (work in progress), January 2019.

[I-D.ietf-lwig-security-protocol-comparison]
Mattsson, J. and F. Palombini, "Comparison of CoAP Security Protocols", draft-ietf-lwig-security-protocol-comparison-02 (work in progress), January 2019.

[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_enabling_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>.

[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>.

[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' and '.cborseq' 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):

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

An 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 "TH_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 = << ID_CRED_x, signature >> in the asymmetric case. Note that if ID_CRED_x contains a single 'kid' parameter, i.e., ID_CRED_x = { 4 : bstr }, only the bstr is conveyed in the plaintext. For instance, if ID_CRED_x = { 4 : h'40' } (CBOR encoding 0xA1044140) and signature = h'050607' (CBOR encoding 0x43050607), then plaintext = h'414043050607'.

- o The external_aad is a CBOR bstr. It is always set to the transcript hash TH_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'', TH_i ]
```

This equals the concatenation of 0x8368456e63727970743040 and the CBOR encoding of TH_i. For instance if TH_2 = h'010203' (CBOR encoding 0x43010203), then A = 0x8368456e63727970743040 43010203.

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 "TH_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.

- The key is the private authentication key of U or V. This may be stored as a COSE_KEY object or as a certificate.
- The protected parameter is a map ID_CRED_x = { label : value } is wrapped in a byte string.
- The payload is a bstr containing the CBOR encoding of a COSE_KEY or a single certificate.
- external_aad = TH_i.

COSE constructs the input to the Signature Algorithm as follows:

- The key is the private authentication key of U or V.
- The message to be signed M is the CBOR encoding of:

  ["Signature1", << { label : value } >>, TH_i, CRED_x]

  For instance, if ID_CRED_x = { 4 : h'1111' } (CBOR encoding 0xA104421111), TH_3 = h'222222' (CBOR encoding 0x43222222), and CRED_U = h'55555555' (CBOR encoding 0x4455555555), then M = 0x846a5369676e617475726531 45A104421111 43222222 4455555555.

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 (the empty byte string) 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'', TH_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 TH_2 = h'aaaa' then

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

Appendix B. Example Messages and Sizes

To help implementors, this appendix gives examples in CBOR diagnostic notation and hexadecimal of EDHOC messages and plaintexts with different authentication methods. The examples use 1 byte key identifiers, 1 byte connection IDs, and the default mapping to CoAP (corr = 1). Note that the examples in this appendix are not test vectors, the cryptographic parts are just replaced with byte strings of the same length.

B.1. Message Sizes RPK

B.1.1. message_1

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

```
message_1 (38 bytes):
01 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 41 C3
```

B.1.2. message_2

```
plaintext = <<
  h'a1',
  h'000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d
    1e1f202122232425262728292a2b2c2d2e2f303132333435363738393a3b
    3c3d3e3f'
>>
```

In the plaintext, the header map { 4 : h'a1' } is encoded as the two bytes h'a1'. The length of plaintext is 68 bytes so assuming a 64-bit MAC value the length of ciphertext is 76 bytes.

```
message_2 = (
  h'000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d
    1e1f',
  h'c4',
  h'000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d
    1e1f202122232425262728292a2b2c2d2e2f303132333435363738393a3b
    3c3d3e3f404142434445464748494a4b'
)
```

```
message_2 (114 bytes):
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 41 C4 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
```

B.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'c4',
  h'000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d
    1e1f202122232425262728292a2b2c2d2e2f303132333435363738393a3b
    3c3d3e3f404142434445464748494a4b'
)
```

```
message_3 (80 bytes):
41 C4 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
```

B.2. Message Sizes Certificates

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

```
{ TDB1 : [ TDB6, h'0001020304050607' ] }
```

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

```
{ TDB3 : h'0001020304050607...' }
```

B.3. Message Sizes PSK

B.3.1. message_1

```
message_1 = (
  4,
  0,
  h'000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d
    1e1f',
  h'c3',
  h'a2'
)

message_1 (40 bytes):
04 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 41 C3 41 A2
```

B.3.2. message_2

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

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

```
message_2 (45 bytes):
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 41 C4 48 61 62 63
64 65 66 67 68
```

B.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'c4',
  h'0001020304050607'
)
```

```
message_3 (11 bytes):
41 C4 48 00 01 02 03 04 05 06 07
```

B.4. Summary

The previous examples of typical message sizes are summarized in Figure 9.

	PSK	RPK	x5t	x5chain
message_1	40	38	38	38
message_2	45	114	126	116 + Certificate chain
message_3	11	80	91	81 + Certificate chain
Total	96	232	255	235 + Certificate chains

Figure 9: Typical message sizes in bytes

These examples use 1 byte key identifiers and connection IDs, this is realistic in many scenarios as most constrained devices only have a few keys and connection. In cases where a node only have one connection or key, the identifiers may even be the empty byte string.

For a comparison with other protocols, see [I-D.ietf-lwig-security-protocol-comparison].

Appendix C. Test Vectors

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

TODO: This section needs to be updated.

Acknowledgments

The authors want to thank Alessandro Bruni, Theis Groenbech Petersen, Dan Harkins, Klaus Hartke, Russ Housley, Alexandros Krontiris, Ilari Liusvaara, Karl Norrman, Salvador Perez, Michael Richardson, Thorvald Sahl Joergensen, Jim Schaad, Carsten Schuermann, Ludwig Seitz, Stanislav Smyshlyaev, Valery Smyslov, and Rene Struik for reviewing and commenting on intermediate versions of the draft. We are especially indebted to Jim Schaad for his continuous reviewing and implementation of different versions of the draft.

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