



**HAL**  
open science

## Traces of EDHOC

Göran Selander, John Preuss Mattsson, Marek Serafin, Marco Tiloca, Mališa Vučinić

► **To cite this version:**

Göran Selander, John Preuss Mattsson, Marek Serafin, Marco Tiloca, Mališa Vučinić. Traces of EDHOC. Internet Engineering Task Force RFC series, In press. hal-04395899

**HAL Id: hal-04395899**

<https://inria.hal.science/hal-04395899v1>

Submitted on 15 Jan 2024

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Distributed under a Creative Commons Attribution 4.0 International License

Workgroup: LAKE Working Group  
Internet-Draft: draft-ietf-lake-traces-08  
Published: 22 September 2023  
Intended Status: Informational  
Expires: 25 March 2024  
Authors: G. Selander    J. Preuß Mattsson    M. Serafin  
          Ericsson        Ericsson                ASSA ABLOY  
          M. Tiloca     M. Vučinić  
          RISE            Inria

## Traces of EDHOC

### Abstract

This document contains some example traces of Ephemeral Diffie-Hellman Over COSE (EDHOC).

### Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 25 March 2024.

### Copyright Notice

Copyright (c) 2023 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

## Table of Contents

- [1. Introduction](#)
    - [1.1. Setup](#)
    - [1.2. Terminology and Requirements Language](#)
  - [2. Authentication with Signatures, X.509 Certificates Identified by 'x5t'](#)
    - [2.1. message 1](#)
    - [2.2. message 2](#)
    - [2.3. message 3](#)
    - [2.4. message 4](#)
    - [2.5. PRK\\_out and PRK\\_exporter](#)
    - [2.6. OSCORE Parameters](#)
    - [2.7. Key Update](#)
    - [2.8. Certificates](#)
  - [3. Authentication with Static DH, CCS Identified by 'kid'](#)
    - [3.1. message 1 \(first time\)](#)
    - [3.2. error](#)
    - [3.3. message 1 \(second time\)](#)
    - [3.4. message 2](#)
    - [3.5. message 3](#)
    - [3.6. message 4](#)
    - [3.7. PRK\\_out and PRK\\_exporter](#)
    - [3.8. OSCORE Parameters](#)
    - [3.9. Key Update](#)
  - [4. Invalid Traces](#)
    - [4.1. Encoding Errors](#)
    - [4.2. Crypto-related Errors](#)
    - [4.3. Non-deterministic CBOR](#)
  - [5. Security Considerations](#)
  - [6. IANA Considerations](#)
  - [7. References](#)
    - [7.1. Normative References](#)
    - [7.2. Informative References](#)
- [Acknowledgments](#)
- [Authors' Addresses](#)

### 1. Introduction

EDHOC [[I-D.ietf-lake-edhoc](#)] is a lightweight authenticated key exchange protocol designed for highly constrained settings. This document contains annotated traces of EDHOC sessions, with input, output, and intermediate processing results to simplify testing of implementations. The traces have been verified by two independent implementations.

## 1.1. Setup

EDHOC is run between an Initiator (I) and a Responder (R). The private/public key pairs and credentials of the Initiator and the Responder required to produce the protocol messages are shown in the traces when needed for the calculations.

EDHOC messages and intermediate results are encoded in CBOR [[RFC8949](#)] and can therefore be displayed in CBOR diagnostic notation using, e.g., the CBOR playground [[CborMe](#)], which makes them easy to parse for humans. Credentials can also be encoded in CBOR, e.g. CBOR Web Tokens (CWT) [[RFC8392](#)].

The document contains two traces:

\*[Section 2](#) - Authentication with signature keys identified by the hash value of the X.509 certificates (provided in [Section 2.8](#)). The endpoints use EdDSA [[RFC8032](#)] for authentication and X25519 [[RFC7748](#)] for ephemeral-ephemeral Diffie-Hellman key exchange.

\*[Section 3](#) - Authentication with static Diffie-Hellman keys identified by short key identifiers labelling CWT Claim Sets (CCSs) [[RFC8392](#)]. The endpoints use NIST P-256 [[SP-800-186](#)] for both ephemeral-ephemeral and static-ephemeral Diffie-Hellman key exchange. This trace also illustrates the cipher suite negotiation, and provides an example of low protocol overhead, with messages sizes of (39, 45, 19) bytes.

Examples of invalid EDHOC messages are found in [Section 4](#).

NOTE 1. The same name is used for hexadecimal byte strings and their CBOR encodings. The traces contain both the raw byte strings and the corresponding CBOR encoded data items.

NOTE 2. If not clear from the context, remember that CBOR sequences and CBOR arrays assume CBOR encoded data items as elements.

NOTE 3. When the protocol transporting EDHOC messages does not inherently provide correlation across all messages, like CoAP [[RFC7252](#)], then some messages typically are prepended with connection identifiers and potentially a message\_1 indicator (see Sections [3.4.1](#) and [A.2](#) of [[I-D.ietf-lake-edhoc](#)]). Those bytes are not included in the traces in this document.

## 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.

## 2. Authentication with Signatures, X.509 Certificates Identified by 'x5t'

In this example the Initiator (I) and Responder (R) are authenticated with digital signatures (METHOD = 0). Both the Initiator and the Responder support cipher suite 0, which determines the algorithms:

```
*EDHOC AEAD algorithm = AES-CCM-16-64-128
*EDHOC hash algorithm = SHA-256
*EDHOC MAC length in bytes (Static DH) = 8
*EDHOC key exchange algorithm (ECDH curve) = X25519
*EDHOC signature algorithm = EdDSA
*Application AEAD algorithm = AES-CCM-16-64-128
*Application hash algorithm = SHA-256
```

The public keys are represented with X.509 certificates identified by the COSE header parameter 'x5t'.

### 2.1. message\_1

Both endpoints are authenticated with signatures, i.e., METHOD = 0:

```
METHOD (CBOR Data Item) (1 byte)
00
```

The Initiator selects cipher suite 0. A single cipher suite is encoded as an int:

```
SUITES_I (CBOR Data Item) (1 byte)
00
```

The Initiator creates an ephemeral key pair for use with the EDHOC key exchange algorithm:

```
Initiator's ephemeral private key
X (Raw Value) (32 bytes)
89 2e c2 8e 5c b6 66 91 08 47 05 39 50 0b 70 5e 60 d0 08 d3 47 c5 81
7e e9 f3 32 7c 8a 87 bb 03
```

Initiator's ephemeral public key

G\_X (Raw Value) (32 bytes)

```
31 f8 2c 7b 5b 9c bb f0 f1 94 d9 13 cc 12 ef 15 32 d3 28 ef 32 63 2a
48 81 a1 c0 70 1e 23 7f 04
```

Initiator's ephemeral public key

G\_X (CBOR Data Item) (34 bytes)

```
58 20 31 f8 2c 7b 5b 9c bb f0 f1 94 d9 13 cc 12 ef 15 32 d3 28 ef 32
63 2a 48 81 a1 c0 70 1e 23 7f 04
```

The Initiator selects its connection identifier C\_I to be the byte string 0x2d, which since it is represented by the 1-byte CBOR int -14 is encoded as 0x2d:

Connection identifier chosen by Initiator

C\_I (Raw Value) (1 byte)

```
2d
```

Connection identifier chosen by Initiator

C\_I (CBOR Data Item) (1 byte)

```
2d
```

No external authorization data:

EAD\_1 (CBOR Sequence) (0 bytes)

The Initiator constructs message\_1:

message\_1 =

```
(
  0,
  0,
  h'31f82c7b5b9cbbf0f194d913cc12ef1532d328ef32632a48
    81a1c0701e237f04',
  -14
)
```

message\_1 (CBOR Sequence) (37 bytes)

```
00 00 58 20 31 f8 2c 7b 5b 9c bb f0 f1 94 d9 13 cc 12 ef 15 32 d3 28
ef 32 63 2a 48 81 a1 c0 70 1e 23 7f 04 2d
```

## 2.2. message\_2

The Responder supports the most preferred and selected cipher suite 0, so SUITES\_I is acceptable.

The Responder creates an ephemeral key pair for use with the EDHOC key exchange algorithm:

Responder's ephemeral private key

Y (Raw Value) (32 bytes)

e6 9c 23 fb f8 1b c4 35 94 24 46 83 7f e8 27 bf 20 6c 8f a1 0a 39 db  
47 44 9e 5a 81 34 21 e1 e8

Responder's ephemeral public key

G\_Y (Raw Value) (32 bytes)

dc 88 d2 d5 1d a5 ed 67 fc 46 16 35 6b c8 ca 74 ef 9e be 8b 38 7e 62  
3a 36 0b a4 80 b9 b2 9d 1c

Responder's ephemeral public key

G\_Y (CBOR Data Item) (34 bytes)

58 20 dc 88 d2 d5 1d a5 ed 67 fc 46 16 35 6b c8 ca 74 ef 9e be 8b 38  
7e 62 3a 36 0b a4 80 b9 b2 9d 1c

The Responder selects its connection identifier C\_R to be the byte string 0x18, which since it is not represented as a 1-byte CBOR int is encoded as h'18' = 0x4118:

Connection identifier chosen by Responder

C\_R (Raw Value) (1 byte)

18

Connection identifier chosen by Responder

C\_R (CBOR Data Item) (2 bytes)

41 18

The transcript hash TH\_2 is calculated using the EDHOC hash algorithm:

$TH_2 = H( G_Y, H(message_1) )$

H(message\_1) (Raw Value) (32 bytes)

c1 65 d6 a9 9d 1b ca fa ac 8d bf 2b 35 2a 6f 7d 71 a3 0b 43 9c 9d 64  
d3 49 a2 38 48 03 8e d1 6b

H(message\_1) (CBOR Data Item) (34 bytes)

58 20 c1 65 d6 a9 9d 1b ca fa ac 8d bf 2b 35 2a 6f 7d 71 a3 0b 43 9c  
9d 64 d3 49 a2 38 48 03 8e d1 6b

The input to calculate TH\_2 is the CBOR sequence:

G\_Y, H(message\_1)

Input to calculate TH\_2 (CBOR Sequence) (68 bytes)

58 20 dc 88 d2 d5 1d a5 ed 67 fc 46 16 35 6b c8 ca 74 ef 9e be 8b 38  
7e 62 3a 36 0b a4 80 b9 b2 9d 1c 58 20 c1 65 d6 a9 9d 1b ca fa ac 8d  
bf 2b 35 2a 6f 7d 71 a3 0b 43 9c 9d 64 d3 49 a2 38 48 03 8e d1 6b

TH\_2 (Raw Value) (32 bytes)

c6 40 5c 15 4c 56 74 66 ab 1d f2 03 69 50 0e 54 0e 9f 14 bd 3a 79 6a  
06 52 ca e6 6c 90 61 68 8d

TH\_2 (CBOR Data Item) (34 bytes)

58 20 c6 40 5c 15 4c 56 74 66 ab 1d f2 03 69 50 0e 54 0e 9f 14 bd 3a  
79 6a 06 52 ca e6 6c 90 61 68 8d

PRK\_2e is specified in [Section 4.1.1.1](#) of [[I-D.ietf-lake-edhoc](#)].

First, the ECDH shared secret G\_XY is computed from G\_X and Y, or G\_Y and X:

G\_XY (Raw Value) (ECDH shared secret) (32 bytes)

e5 cd f3 a9 86 cd ac 5b 7b f0 46 91 e2 b0 7c 08 e7 1f 53 99 8d 8f 84  
2b 7c 3f b4 d8 39 cf 7b 28

Then, PRK\_2e is calculated using EDHOC\_Extract() determined by the EDHOC hash algorithm:

PRK\_2e = EDHOC\_Extract( salt, G\_XY ) =  
= HMAC-SHA-256( salt, G\_XY )

where salt is TH\_2:

salt (Raw Value) (32 bytes)

c6 40 5c 15 4c 56 74 66 ab 1d f2 03 69 50 0e 54 0e 9f 14 bd 3a 79 6a  
06 52 ca e6 6c 90 61 68 8d

PRK\_2e (Raw Value) (32 bytes)

d5 84 ac 2e 5d ad 5a 77 d1 4b 53 eb e7 2e f1 d5 da a8 86 0d 39 93 73  
bf 2c 24 0a fa 7b a8 04 da

Since METHOD = 0, the Responder authenticates using signatures.  
Since the selected cipher suite is 0, the EDHOC signature algorithm is EdDSA.

The Responder's signature key pair using EdDSA:

Responder's private authentication key

SK\_R (Raw Value) (32 bytes)

ef 14 0f f9 00 b0 ab 03 f0 c0 8d 87 9c bb d4 b3 1e a7 1e 6e 7e e7 ff  
cb 7e 79 55 77 7a 33 27 99

Responder's public authentication key

PK\_R (Raw Value) (32 bytes)

a1 db 47 b9 51 84 85 4a d1 2a 0c 1a 35 4e 41 8a ac e3 3a a0 f2 c6 62  
c0 0b 3a c5 5d e9 2f 93 59

PRK\_3e2m is specified in [Section 4.1.1.2](#) of [[I-D.ietf-lake-edhoc](#)].



Since the Responder authenticates with signatures PRK\_3e2m = PRK\_2e.

PRK\_3e2m (Raw Value) (32 bytes)

d5 84 ac 2e 5d ad 5a 77 d1 4b 53 eb e7 2e f1 d5 da a8 86 0d 39 93 73  
bf 2c 24 0a fa 7b a8 04 da

The Responder constructs the remaining input needed to calculate MAC\_2:

MAC\_2 = EDHOC\_KDF( PRK\_3e2m, 2, context\_2, mac\_length\_2 )

context\_2 = << ID\_CRED\_R, TH\_2, CRED\_R, ? EAD\_2 >>

CRED\_R is identified by a 64-bit hash:

ID\_CRED\_R =

```
{  
  34 : [-15, h'79f2a41b510c1f9b']  
}
```

where the COSE header value 34 ('x5t') indicates a hash of an X.509 certificate, and the COSE algorithm -15 indicates the hash algorithm SHA-256 truncated to 64 bits.

ID\_CRED\_R (CBOR Data Item) (14 bytes)

a1 18 22 82 2e 48 79 f2 a4 1b 51 0c 1f 9b

CRED\_R is a CBOR byte string of the DER encoding of the X.509 certificate in [Section 2.8.1](#):

CRED\_R (Raw Value) (241 bytes)

30 81 ee 30 81 a1 a0 03 02 01 02 02 04 62 31 9e c4 30 05 06 03 2b 65  
70 30 1d 31 1b 30 19 06 03 55 04 03 0c 12 45 44 48 4f 43 20 52 6f 6f  
74 20 45 64 32 35 35 31 39 30 1e 17 0d 32 32 30 33 31 36 30 38 32 34  
33 36 5a 17 0d 32 39 31 32 33 31 32 33 30 30 30 30 5a 30 22 31 20 30  
1e 06 03 55 04 03 0c 17 45 44 48 4f 43 20 52 65 73 70 6f 6e 64 65 72  
20 45 64 32 35 35 31 39 30 2a 30 05 06 03 2b 65 70 03 21 00 a1 db 47  
b9 51 84 85 4a d1 2a 0c 1a 35 4e 41 8a ac e3 3a a0 f2 c6 62 c0 0b 3a  
c5 5d e9 2f 93 59 30 05 06 03 2b 65 70 03 41 00 b7 23 bc 01 ea b0 92  
8e 8b 2b 6c 98 de 19 cc 38 23 d4 6e 7d 69 87 b0 32 47 8f ec fa f1 45  
37 a1 af 14 cc 8b e8 29 c6 b7 30 44 10 18 37 eb 4a bc 94 95 65 d8 6d  
ce 51 cf ae 52 ab 82 c1 52 cb 02

CRED\_R (CBOR Data Item) (243 bytes)

```
58 f1 30 81 ee 30 81 a1 a0 03 02 01 02 02 04 62 31 9e c4 30 05 06 03
2b 65 70 30 1d 31 1b 30 19 06 03 55 04 03 0c 12 45 44 48 4f 43 20 52
6f 6f 74 20 45 64 32 35 35 31 39 30 1e 17 0d 32 32 30 33 31 36 30 38
32 34 33 36 5a 17 0d 32 39 31 32 33 31 32 33 30 30 30 30 5a 30 22 31
20 30 1e 06 03 55 04 03 0c 17 45 44 48 4f 43 20 52 65 73 70 6f 6e 64
65 72 20 45 64 32 35 35 31 39 30 2a 30 05 06 03 2b 65 70 03 21 00 a1
db 47 b9 51 84 85 4a d1 2a 0c 1a 35 4e 41 8a ac e3 3a a0 f2 c6 62 c0
0b 3a c5 5d e9 2f 93 59 30 05 06 03 2b 65 70 03 41 00 b7 23 bc 01 ea
b0 92 8e 8b 2b 6c 98 de 19 cc 38 23 d4 6e 7d 69 87 b0 32 47 8f ec fa
f1 45 37 a1 af 14 cc 8b e8 29 c6 b7 30 44 10 18 37 eb 4a bc 94 95 65
d8 6d ce 51 cf ae 52 ab 82 c1 52 cb 02
```

No external authorization data:

EAD\_2 (CBOR Sequence) (0 bytes)

context\_2 = << ID\_CRED\_R, TH\_2, CRED\_R, ? EAD\_2 >>

context\_2 (CBOR Sequence) (291 bytes)

```
a1 18 22 82 2e 48 79 f2 a4 1b 51 0c 1f 9b 58 20 c6 40 5c 15 4c 56 74
66 ab 1d f2 03 69 50 0e 54 0e 9f 14 bd 3a 79 6a 06 52 ca e6 6c 90 61
68 8d 58 f1 30 81 ee 30 81 a1 a0 03 02 01 02 02 04 62 31 9e c4 30 05
06 03 2b 65 70 30 1d 31 1b 30 19 06 03 55 04 03 0c 12 45 44 48 4f 43
20 52 6f 6f 74 20 45 64 32 35 35 31 39 30 1e 17 0d 32 32 30 33 31 36
30 38 32 34 33 36 5a 17 0d 32 39 31 32 33 31 32 33 30 30 30 30 5a 30
22 31 20 30 1e 06 03 55 04 03 0c 17 45 44 48 4f 43 20 52 65 73 70 6f
6e 64 65 72 20 45 64 32 35 35 31 39 30 2a 30 05 06 03 2b 65 70 03 21
00 a1 db 47 b9 51 84 85 4a d1 2a 0c 1a 35 4e 41 8a ac e3 3a a0 f2 c6
62 c0 0b 3a c5 5d e9 2f 93 59 30 05 06 03 2b 65 70 03 41 00 b7 23 bc
01 ea b0 92 8e 8b 2b 6c 98 de 19 cc 38 23 d4 6e 7d 69 87 b0 32 47 8f
ec fa f1 45 37 a1 af 14 cc 8b e8 29 c6 b7 30 44 10 18 37 eb 4a bc 94
95 65 d8 6d ce 51 cf ae 52 ab 82 c1 52 cb 02
```

context\_2 (CBOR byte string) (294 bytes)

```
59 01 23 a1 18 22 82 2e 48 79 f2 a4 1b 51 0c 1f 9b 58 20 c6 40 5c 15
4c 56 74 66 ab 1d f2 03 69 50 0e 54 0e 9f 14 bd 3a 79 6a 06 52 ca e6
6c 90 61 68 8d 58 f1 30 81 ee 30 81 a1 a0 03 02 01 02 02 04 62 31 9e
c4 30 05 06 03 2b 65 70 30 1d 31 1b 30 19 06 03 55 04 03 0c 12 45 44
48 4f 43 20 52 6f 6f 74 20 45 64 32 35 35 31 39 30 1e 17 0d 32 32 30
33 31 36 30 38 32 34 33 36 5a 17 0d 32 39 31 32 33 31 32 33 30 30 30
30 5a 30 22 31 20 30 1e 06 03 55 04 03 0c 17 45 44 48 4f 43 20 52 65
73 70 6f 6e 64 65 72 20 45 64 32 35 35 31 39 30 2a 30 05 06 03 2b 65
70 03 21 00 a1 db 47 b9 51 84 85 4a d1 2a 0c 1a 35 4e 41 8a ac e3 3a
a0 f2 c6 62 c0 0b 3a c5 5d e9 2f 93 59 30 05 06 03 2b 65 70 03 41 00
b7 23 bc 01 ea b0 92 8e 8b 2b 6c 98 de 19 cc 38 23 d4 6e 7d 69 87 b0
32 47 8f ec fa f1 45 37 a1 af 14 cc 8b e8 29 c6 b7 30 44 10 18 37 eb
4a bc 94 95 65 d8 6d ce 51 cf ae 52 ab 82 c1 52 cb 02
```

MAC\_2 is computed through EDHOC\_Expand() using the EDHOC hash algorithm, see [Section 4.1.2](#) of [[I-D.ietf-lake-edhoc](#)]:

MAC\_2 = HKDF-Expand(PRK\_3e2m, info, mac\_length\_2), where

info = ( 2, context\_2, mac\_length\_2 )

Since METHOD = 0, mac\_length\_2 is given by the EDHOC hash algorithm.

info for MAC\_2 is:

```
info =  
(  
  2,  
  h'a11822822e4879f2a41b510c1f9b5820c6405c154c567466  
    ab1df20369500e540e9f14bd3a796a0652cae66c9061688d  
    58f13081ee3081a1a003020102020462319ec4300506032b  
    6570301d311b301906035504030c124544484f4320526f6f  
    742045643235353139301e170d3232303331363038323433  
    365a170d3239313233313233303030305a30223120301e06  
    035504030c174544484f4320526573706f6e646572204564  
    3235353139302a300506032b6570032100a1db47b9518485  
    4ad12a0c1a354e418aace33aa0f2c662c00b3ac55de92f93  
    59300506032b6570034100b723bc01eab0928e8b2b6c98de  
    19cc3823d46e7d6987b032478fecfaf14537a1af14cc8be8  
    29c6b73044101837eb4abc949565d86dce51cfae52ab82c1  
    52cb02',  
  32  
)
```

where the last value is the output size of the EDHOC hash algorithm in bytes.

info for MAC\_2 (CBOR Sequence) (297 bytes)

```
02 59 01 23 a1 18 22 82 2e 48 79 f2 a4 1b 51 0c 1f 9b 58 20 c6 40 5c  
15 4c 56 74 66 ab 1d f2 03 69 50 0e 54 0e 9f 14 bd 3a 79 6a 06 52 ca  
e6 6c 90 61 68 8d 58 f1 30 81 ee 30 81 a1 a0 03 02 01 02 02 04 62 31  
9e c4 30 05 06 03 2b 65 70 30 1d 31 1b 30 19 06 03 55 04 03 0c 12 45  
44 48 4f 43 20 52 6f 6f 74 20 45 64 32 35 35 31 39 30 1e 17 0d 32 32  
30 33 31 36 30 38 32 34 33 36 5a 17 0d 32 39 31 32 33 31 32 33 30 30  
30 30 5a 30 22 31 20 30 1e 06 03 55 04 03 0c 17 45 44 48 4f 43 20 52  
65 73 70 6f 6e 64 65 72 20 45 64 32 35 35 31 39 30 2a 30 05 06 03 2b  
65 70 03 21 00 a1 db 47 b9 51 84 85 4a d1 2a 0c 1a 35 4e 41 8a ac e3  
3a a0 f2 c6 62 c0 0b 3a c5 5d e9 2f 93 59 30 05 06 03 2b 65 70 03 41  
00 b7 23 bc 01 ea b0 92 8e 8b 2b 6c 98 de 19 cc 38 23 d4 6e 7d 69 87  
b0 32 47 8f ec fa f1 45 37 a1 af 14 cc 8b e8 29 c6 b7 30 44 10 18 37  
eb 4a bc 94 95 65 d8 6d ce 51 cf ae 52 ab 82 c1 52 cb 02 18 20
```

MAC\_2 (Raw Value) (32 bytes)

36 9c a4 39 2c 83 ed 63 d6 1a d2 18 42 0e a3 67 06 00 84 78 d5 bc 30  
49 fb 8c 59 42 44 4b 13 33

MAC\_2 (CBOR Data Item) (34 bytes)

58 20 36 9c a4 39 2c 83 ed 63 d6 1a d2 18 42 0e a3 67 06 00 84 78 d5  
bc 30 49 fb 8c 59 42 44 4b 13 33

Since METHOD = 0, Signature\_or\_MAC\_2 is the 'signature' of the COSE\_Sign1 object.

The Responder constructs the message to be signed:

```
[ "Signature1", << ID_CRED_R >>,  
<< TH_2, CRED_R, ? EAD_2 >>, MAC_2 ] =
```

```
[  
  "Signature1",  
  h'a11822822e4879f2a41b510c1f9b',  
  h'5820c6405c154c567466ab1df20369500e540e9f14bd3a79  
    6a0652cae66c9061688d58f13081ee3081a1a00302010202  
    0462319ec4300506032b6570301d311b301906035504030c  
    124544484f4320526f6f742045643235353139301e170d32  
    32303331363038323433365a170d32393132333132333030  
    30305a30223120301e06035504030c174544484f43205265  
    73706f6e6465722045643235353139302a300506032b6570  
    032100a1db47b95184854ad12a0c1a354e418aace33aa0f2  
    c662c00b3ac55de92f9359300506032b6570034100b723bc  
    01eab0928e8b2b6c98de19cc3823d46e7d6987b032478fec  
    faf14537a1af14cc8be829c6b73044101837eb4abc949565  
    d86dce51cfae52ab82c152cb02',  
  h'369ca4392c83ed63d61ad218420ea36706008478d5bc3049  
    fb8c5942444b1333'  
]
```

Message to be signed 2 (CBOR Data Item) (341 bytes)

```
84 6a 53 69 67 6e 61 74 75 72 65 31 4e a1 18 22 82 2e 48 79 f2 a4 1b
51 0c 1f 9b 59 01 15 58 20 c6 40 5c 15 4c 56 74 66 ab 1d f2 03 69 50
0e 54 0e 9f 14 bd 3a 79 6a 06 52 ca e6 6c 90 61 68 8d 58 f1 30 81 ee
30 81 a1 a0 03 02 01 02 02 04 62 31 9e c4 30 05 06 03 2b 65 70 30 1d
31 1b 30 19 06 03 55 04 03 0c 12 45 44 48 4f 43 20 52 6f 6f 74 20 45
64 32 35 35 31 39 30 1e 17 0d 32 32 30 33 31 36 30 38 32 34 33 36 5a
17 0d 32 39 31 32 33 31 32 33 30 30 30 30 5a 30 22 31 20 30 1e 06 03
55 04 03 0c 17 45 44 48 4f 43 20 52 65 73 70 6f 6e 64 65 72 20 45 64
32 35 35 31 39 30 2a 30 05 06 03 2b 65 70 03 21 00 a1 db 47 b9 51 84
85 4a d1 2a 0c 1a 35 4e 41 8a ac e3 3a a0 f2 c6 62 c0 0b 3a c5 5d e9
2f 93 59 30 05 06 03 2b 65 70 03 41 00 b7 23 bc 01 ea b0 92 8e 8b 2b
6c 98 de 19 cc 38 23 d4 6e 7d 69 87 b0 32 47 8f ec fa f1 45 37 a1 af
14 cc 8b e8 29 c6 b7 30 44 10 18 37 eb 4a bc 94 95 65 d8 6d ce 51 cf
ae 52 ab 82 c1 52 cb 02 58 20 36 9c a4 39 2c 83 ed 63 d6 1a d2 18 42
0e a3 67 06 00 84 78 d5 bc 30 49 fb 8c 59 42 44 4b 13 33
```

The Responder signs using the private authentication key SK\_R

Signature\_or\_MAC\_2 (Raw Value) (64 bytes)

```
41 e6 91 27 5b 84 04 24 25 5a cb 87 e6 33 d7 5d da 71 50 2d a2 e3 da
5f ce ee c4 e3 f7 60 74 48 6f 87 e6 6f 2a ca a1 bb d4 8c e0 e6 6a 5d
64 38 91 54 48 2f 9a 5e 57 22 70 63 31 59 f2 b1 7e 0e
```

Signature\_or\_MAC\_2 (CBOR Data Item) (66 bytes)

```
58 40 41 e6 91 27 5b 84 04 24 25 5a cb 87 e6 33 d7 5d da 71 50 2d a2
e3 da 5f ce ee c4 e3 f7 60 74 48 6f 87 e6 6f 2a ca a1 bb d4 8c e0 e6
6a 5d 64 38 91 54 48 2f 9a 5e 57 22 70 63 31 59 f2 b1 7e 0e
```

The Responder constructs PLAINTEXT\_2:

PLAINTEXT\_2 =

```
(
  C_R,
  ID_CRED_R / bstr / -24..23,
  Signature_or_MAC_2,
  ? EAD_2
)
```

PLAINTEXT\_2 (CBOR Sequence) (82 bytes)

```
41 18 a1 18 22 82 2e 48 79 f2 a4 1b 51 0c 1f 9b 58 40 41 e6 91 27 5b
84 04 24 25 5a cb 87 e6 33 d7 5d da 71 50 2d a2 e3 da 5f ce ee c4 e3
f7 60 74 48 6f 87 e6 6f 2a ca a1 bb d4 8c e0 e6 6a 5d 64 38 91 54 48
2f 9a 5e 57 22 70 63 31 59 f2 b1 7e 0e
```

The input needed to calculate KEYSTREAM\_2 is defined in [Section 4.1.2](#) of [[I-D.ietf-lake-edhoc](#)], using EDHOC\_Expand() with the EDHOC hash algorithm:

```
KEYSTREAM_2 = EDHOC_KDF( PRK_2e, 0, TH_2, plaintext_length ) =  
    = HKDF-Expand( PRK_2e, info, plaintext_length )
```

where plaintext\_length is the length in bytes of PLAINTEXT\_2 in bytes, and info for KEYSTREAM\_2 is:

```
info =  
(  
    0,  
    h'c6405c154c567466ab1df20369500e540e9f14bd3a796a06  
      52cae66c9061688d',  
    82  
)
```

where the last value is the length in bytes of PLAINTEXT\_2.

```
info for KEYSTREAM_2 (CBOR Sequence) (37 bytes)  
00 58 20 c6 40 5c 15 4c 56 74 66 ab 1d f2 03 69 50 0e 54 0e 9f 14 bd  
3a 79 6a 06 52 ca e6 6c 90 61 68 8d 18 52
```

```
KEYSTREAM_2 (Raw Value) (82 bytes)  
fd 3e 7c 3f 2d 6b ee 64 3d 3c 9d 2f 28 47 03 5d 73 e2 ec b0 f8 db 5c  
d1 c6 85 4e 24 89 6a f2 11 88 b2 c4 34 4e 68 9e c2 98 42 83 d9 fb c6  
9c e1 c5 db 10 dc ff f2 4d f9 a4 9a 04 a9 40 58 27 7b c7 fa 9a d6 c6  
b1 94 ab 32 8b 44 5e b0 80 49 0c d7 86
```

The Responder calculates CIPHERTEXT\_2 as XOR between PLAINTEXT\_2 and KEYSTREAM\_2:

```
CIPHERTEXT_2 (Raw Value) (82 bytes)  
bc 26 dd 27 0f e9 c0 2c 44 ce 39 34 79 4b 1c c6 2b a2 ad 56 69 fc 07  
55 c2 a1 6b 7e 42 ed 14 22 5f ef 1e 45 1e 45 3c 21 42 1d 4d 37 3f 25  
6b 81 b1 93 7f 5b 19 9d 67 33 05 21 d0 25 a0 be 4d 26 a3 c2 0b 82 8e  
9e 0e f5 65 a9 34 3d 81 d9 bb bd a9 88
```

The Responder constructs message\_2:

```
message_2 =  
(  
    G_Y_CIPHERTEXT_2  
)
```

where G\_Y\_CIPHERTEXT\_2 is the bstr encoding of the concatenation of the raw values of G\_Y and CIPHERTEXT\_2.

message\_2 (CBOR Sequence) (116 bytes)

```
58 72 dc 88 d2 d5 1d a5 ed 67 fc 46 16 35 6b c8 ca 74 ef 9e be 8b 38
7e 62 3a 36 0b a4 80 b9 b2 9d 1c bc 26 dd 27 0f e9 c0 2c 44 ce 39 34
79 4b 1c c6 2b a2 ad 56 69 fc 07 55 c2 a1 6b 7e 42 ed 14 22 5f ef 1e
45 1e 45 3c 21 42 1d 4d 37 3f 25 6b 81 b1 93 7f 5b 19 9d 67 33 05 21
d0 25 a0 be 4d 26 a3 c2 0b 82 8e 9e 0e f5 65 a9 34 3d 81 d9 bb bd a9
88
```

### 2.3. message\_3

Since METHOD = 0, the Initiator authenticates using signatures.  
Since the selected cipher suite is 0, the EDHOC signature algorithm is EdDSA.

The Initiator's signature key pair using EdDSA:

Initiator's private authentication key

SK\_I (Raw Value) (32 bytes)

```
4c 5b 25 87 8f 50 7c 6b 9d ae 68 fb d4 fd 3f f9 97 53 3d b0 af 00 b2
5d 32 4e a2 8e 6c 21 3b c8
```

Initiator's public authentication key

PK\_I (Raw Value) (32 bytes)

```
ed 06 a8 ae 61 a8 29 ba 5f a5 45 25 c9 d0 7f 48 dd 44 a3 02 f4 3e 0f
23 d8 cc 20 b7 30 85 14 1e
```

PRK\_4e3m is specified in [Section 4.1.1.3](#) of [\[I-D.ietf-lake-edhoc\]](#).

Since the Initiator authenticates with signatures PRK\_4e3m = PRK\_3e2m.

PRK\_4e3m (Raw Value) (32 bytes)

```
d5 84 ac 2e 5d ad 5a 77 d1 4b 53 eb e7 2e f1 d5 da a8 86 0d 39 93 73
bf 2c 24 0a fa 7b a8 04 da
```

The transcript hash TH\_3 is calculated using the EDHOC hash algorithm:

TH\_3 = H(TH\_2, PLAINTEXT\_2, CRED\_R)

Input to calculate TH\_3 (CBOR Sequence) (359 bytes)

```
58 20 c6 40 5c 15 4c 56 74 66 ab 1d f2 03 69 50 0e 54 0e 9f 14 bd 3a
79 6a 06 52 ca e6 6c 90 61 68 8d 41 18 a1 18 22 82 2e 48 79 f2 a4 1b
51 0c 1f 9b 58 40 41 e6 91 27 5b 84 04 24 25 5a cb 87 e6 33 d7 5d da
71 50 2d a2 e3 da 5f ce ee c4 e3 f7 60 74 48 6f 87 e6 6f 2a ca a1 bb
d4 8c e0 e6 6a 5d 64 38 91 54 48 2f 9a 5e 57 22 70 63 31 59 f2 b1 7e
0e 58 f1 30 81 ee 30 81 a1 a0 03 02 01 02 02 04 62 31 9e c4 30 05 06
03 2b 65 70 30 1d 31 1b 30 19 06 03 55 04 03 0c 12 45 44 48 4f 43 20
52 6f 6f 74 20 45 64 32 35 35 31 39 30 1e 17 0d 32 32 30 33 31 36 30
38 32 34 33 36 5a 17 0d 32 39 31 32 33 31 32 33 30 30 30 5a 30 22
31 20 30 1e 06 03 55 04 03 0c 17 45 44 48 4f 43 20 52 65 73 70 6f 6e
64 65 72 20 45 64 32 35 35 31 39 30 2a 30 05 06 03 2b 65 70 03 21 00
a1 db 47 b9 51 84 85 4a d1 2a 0c 1a 35 4e 41 8a ac e3 3a a0 f2 c6 62
c0 0b 3a c5 5d e9 2f 93 59 30 05 06 03 2b 65 70 03 41 00 b7 23 bc 01
ea b0 92 8e 8b 2b 6c 98 de 19 cc 38 23 d4 6e 7d 69 87 b0 32 47 8f ec
fa f1 45 37 a1 af 14 cc 8b e8 29 c6 b7 30 44 10 18 37 eb 4a bc 94 95
65 d8 6d ce 51 cf ae 52 ab 82 c1 52 cb 02
```

TH\_3 (Raw Value) (32 bytes)

```
e0 91 12 1a f5 ac 6c e2 14 5d 48 25 e0 90 12 f2 97 98 e8 f7 13 ac 98
91 43 2d 22 56 b6 f6 78 e9
```

TH\_3 (CBOR Data Item) (34 bytes)

```
58 20 e0 91 12 1a f5 ac 6c e2 14 5d 48 25 e0 90 12 f2 97 98 e8 f7 13
ac 98 91 43 2d 22 56 b6 f6 78 e9
```

The Initiator constructs the remaining input needed to calculate MAC\_3:

```
MAC_3 = EDHOC_KDF( PRK_4e3m, 6, context_3, mac_length_3 )
```

where

```
context_3 = << ID_CRED_I, TH_3, CRED_I, ? EAD_3 >>
```

CRED\_I is identified by a 64-bit hash:

```
ID_CRED_I =
{
  34 : [-15, h'c24ab2fd7643c79f']
}
```

where the COSE header value 34 ('x5t') indicates a hash of an X.509 certificate, and the COSE algorithm -15 indicates the hash algorithm SHA-256 truncated to 64 bits.

ID\_CRED\_I (CBOR Data Item) (14 bytes)

```
a1 18 22 82 2e 48 c2 4a b2 fd 76 43 c7 9f
```



CRED\_I is a CBOR byte string of the DER encoding of the X.509 certificate in [Section 2.8.2](#):

CRED\_I (Raw Value) (241 bytes)

```
30 81 ee 30 81 a1 a0 03 02 01 02 02 04 62 31 9e a0 30 05 06 03 2b 65
70 30 1d 31 1b 30 19 06 03 55 04 03 0c 12 45 44 48 4f 43 20 52 6f 6f
74 20 45 64 32 35 35 31 39 30 1e 17 0d 32 32 30 33 31 36 30 38 32 34
30 30 5a 17 0d 32 39 31 32 33 31 32 33 30 30 30 30 5a 30 22 31 20 30
1e 06 03 55 04 03 0c 17 45 44 48 4f 43 20 49 6e 69 74 69 61 74 6f 72
20 45 64 32 35 35 31 39 30 2a 30 05 06 03 2b 65 70 03 21 00 ed 06 a8
ae 61 a8 29 ba 5f a5 45 25 c9 d0 7f 48 dd 44 a3 02 f4 3e 0f 23 d8 cc
20 b7 30 85 14 1e 30 05 06 03 2b 65 70 03 41 00 52 12 41 d8 b3 a7 70
99 6b cf c9 b9 ea d4 e7 e0 a1 c0 db 35 3a 3b df 29 10 b3 92 75 ae 48
b7 56 01 59 81 85 0d 27 db 67 34 e3 7f 67 21 22 67 dd 05 ee ff 27 b9
e7 a8 13 fa 57 4b 72 a0 0b 43 0b
```

CRED\_I (CBOR Data Item) (243 bytes)

```
58 f1 30 81 ee 30 81 a1 a0 03 02 01 02 02 04 62 31 9e a0 30 05 06 03
2b 65 70 30 1d 31 1b 30 19 06 03 55 04 03 0c 12 45 44 48 4f 43 20 52
6f 6f 74 20 45 64 32 35 35 31 39 30 1e 17 0d 32 32 30 33 31 36 30 38
32 34 30 30 5a 17 0d 32 39 31 32 33 31 32 33 30 30 30 30 5a 30 22 31
20 30 1e 06 03 55 04 03 0c 17 45 44 48 4f 43 20 49 6e 69 74 69 61 74
6f 72 20 45 64 32 35 35 31 39 30 2a 30 05 06 03 2b 65 70 03 21 00 ed
06 a8 ae 61 a8 29 ba 5f a5 45 25 c9 d0 7f 48 dd 44 a3 02 f4 3e 0f 23
d8 cc 20 b7 30 85 14 1e 30 05 06 03 2b 65 70 03 41 00 52 12 41 d8 b3
a7 70 99 6b cf c9 b9 ea d4 e7 e0 a1 c0 db 35 3a 3b df 29 10 b3 92 75
ae 48 b7 56 01 59 81 85 0d 27 db 67 34 e3 7f 67 21 22 67 dd 05 ee ff
27 b9 e7 a8 13 fa 57 4b 72 a0 0b 43 0b
```

No external authorization data:

EAD\_3 (CBOR Sequence) (0 bytes)

context\_3 = << ID\_CRED\_I, TH\_3, CRED\_I, ? EAD\_3 >>

context\_3 (CBOR Sequence) (291 bytes)

```
a1 18 22 82 2e 48 c2 4a b2 fd 76 43 c7 9f 58 20 e0 91 12 1a f5 ac 6c
e2 14 5d 48 25 e0 90 12 f2 97 98 e8 f7 13 ac 98 91 43 2d 22 56 b6 f6
78 e9 58 f1 30 81 ee 30 81 a1 a0 03 02 01 02 02 04 62 31 9e a0 30 05
06 03 2b 65 70 30 1d 31 1b 30 19 06 03 55 04 03 0c 12 45 44 48 4f 43
20 52 6f 6f 74 20 45 64 32 35 35 31 39 30 1e 17 0d 32 32 30 33 31 36
30 38 32 34 30 30 5a 17 0d 32 39 31 32 33 31 32 33 30 30 30 30 5a 30
22 31 20 30 1e 06 03 55 04 03 0c 17 45 44 48 4f 43 20 49 6e 69 74 69
61 74 6f 72 20 45 64 32 35 35 31 39 30 2a 30 05 06 03 2b 65 70 03 21
00 ed 06 a8 ae 61 a8 29 ba 5f a5 45 25 c9 d0 7f 48 dd 44 a3 02 f4 3e
0f 23 d8 cc 20 b7 30 85 14 1e 30 05 06 03 2b 65 70 03 41 00 52 12 41
d8 b3 a7 70 99 6b cf c9 b9 ea d4 e7 e0 a1 c0 db 35 3a 3b df 29 10 b3
92 75 ae 48 b7 56 01 59 81 85 0d 27 db 67 34 e3 7f 67 21 22 67 dd 05
ee ff 27 b9 e7 a8 13 fa 57 4b 72 a0 0b 43 0b
```

```
context_3 (CBOR byte string) (294 bytes)
59 01 23 a1 18 22 82 2e 48 c2 4a b2 fd 76 43 c7 9f 58 20 e0 91 12 1a
f5 ac 6c e2 14 5d 48 25 e0 90 12 f2 97 98 e8 f7 13 ac 98 91 43 2d 22
56 b6 f6 78 e9 58 f1 30 81 ee 30 81 a1 a0 03 02 01 02 02 04 62 31 9e
a0 30 05 06 03 2b 65 70 30 1d 31 1b 30 19 06 03 55 04 03 0c 12 45 44
48 4f 43 20 52 6f 6f 74 20 45 64 32 35 35 31 39 30 1e 17 0d 32 32 30
33 31 36 30 38 32 34 30 30 5a 17 0d 32 39 31 32 33 31 32 33 30 30 30
30 5a 30 22 31 20 30 1e 06 03 55 04 03 0c 17 45 44 48 4f 43 20 49 6e
69 74 69 61 74 6f 72 20 45 64 32 35 35 31 39 30 2a 30 05 06 03 2b 65
70 03 21 00 ed 06 a8 ae 61 a8 29 ba 5f a5 45 25 c9 d0 7f 48 dd 44 a3
02 f4 3e 0f 23 d8 cc 20 b7 30 85 14 1e 30 05 06 03 2b 65 70 03 41 00
52 12 41 d8 b3 a7 70 99 6b cf c9 b9 ea d4 e7 e0 a1 c0 db 35 3a 3b df
29 10 b3 92 75 ae 48 b7 56 01 59 81 85 0d 27 db 67 34 e3 7f 67 21 22
67 dd 05 ee ff 27 b9 e7 a8 13 fa 57 4b 72 a0 0b 43 0b
```

MAC\_3 is computed through EDHOC\_Expand() using the EDHOC hash algorithm, see [Section 4.1.2](#) of [[I-D.ietf-lake-edhoc](#)]:

MAC\_3 = HKDF-Expand(PRK\_4e3m, info, mac\_length\_3), where

info = ( 6, context\_3, mac\_length\_3 )

where context\_3 = << ID\_CRED\_I, TH\_3, CRED\_I, ? EAD\_3 >>

Since METHOD = 0, mac\_length\_3 is given by the EDHOC hash algorithm.

info for MAC\_3 is:

```
info =
(
  6,
  h'a11822822e48c24ab2fd7643c79f5820e091121af5ac6ce2
  145d4825e09012f29798e8f713ac9891432d2256b6f678e9
  58f13081ee3081a1a003020102020462319ea0300506032b
  6570301d311b301906035504030c124544484f4320526f6f
  742045643235353139301e170d3232303331363038323430
  305a170d3239313233313233303030305a30223120301e06
  035504030c174544484f4320496e69746961746f72204564
  3235353139302a300506032b6570032100ed06a8ae61a829
  ba5fa54525c9d07f48dd44a302f43e0f23d8cc20b7308514
  1e300506032b6570034100521241d8b3a770996bcfc9b9ea
  d4e7e0a1c0db353a3bdf2910b39275ae48b756015981850d
  27db6734e37f67212267dd05eeff27b9e7a813fa574b72a0
  0b430b',
  32
)
```

where the last value is the output size of the EDHOC hash algorithm in bytes.

info for MAC\_3 (CBOR Sequence) (297 bytes)

```
06 59 01 23 a1 18 22 82 2e 48 c2 4a b2 fd 76 43 c7 9f 58 20 e0 91 12
1a f5 ac 6c e2 14 5d 48 25 e0 90 12 f2 97 98 e8 f7 13 ac 98 91 43 2d
22 56 b6 f6 78 e9 58 f1 30 81 ee 30 81 a1 a0 03 02 01 02 02 04 62 31
9e a0 30 05 06 03 2b 65 70 30 1d 31 1b 30 19 06 03 55 04 03 0c 12 45
44 48 4f 43 20 52 6f 6f 74 20 45 64 32 35 35 31 39 30 1e 17 0d 32 32
30 33 31 36 30 38 32 34 30 30 5a 17 0d 32 39 31 32 33 31 32 33 30 30
30 30 5a 30 22 31 20 30 1e 06 03 55 04 03 0c 17 45 44 48 4f 43 20 49
6e 69 74 69 61 74 6f 72 20 45 64 32 35 35 31 39 30 2a 30 05 06 03 2b
65 70 03 21 00 ed 06 a8 ae 61 a8 29 ba 5f a5 45 25 c9 d0 7f 48 dd 44
a3 02 f4 3e 0f 23 d8 cc 20 b7 30 85 14 1e 30 05 06 03 2b 65 70 03 41
00 52 12 41 d8 b3 a7 70 99 6b cf c9 b9 ea d4 e7 e0 a1 c0 db 35 3a 3b
df 29 10 b3 92 75 ae 48 b7 56 01 59 81 85 0d 27 db 67 34 e3 7f 67 21
22 67 dd 05 ee ff 27 b9 e7 a8 13 fa 57 4b 72 a0 0b 43 0b 18 20
```

MAC\_3 (Raw Value) (32 bytes)

```
51 c9 68 a7 f9 fd ea 19 c7 02 3f 70 22 b4 d9 f2 14 77 2e f5 88 59 05
24 05 76 f6 2d 03 6e 69 dc
```

MAC\_3 (CBOR Data Item) (34 bytes)

```
58 20 51 c9 68 a7 f9 fd ea 19 c7 02 3f 70 22 b4 d9 f2 14 77 2e f5 88
59 05 24 05 76 f6 2d 03 6e 69 dc
```

Since METHOD = 0, Signature\_or\_MAC\_3 is the 'signature' of the COSE\_Sign1 object.

The Initiator constructs the message to be signed:

```
[ "Signature1", << ID_CRED_I >>,
  << TH_3, CRED_I, ? EAD_3 >>, MAC_3 ] =
```

```
[
  "Signature1",
  h'a11822822e48c24ab2fd7643c79f',
  h'5820e091121af5ac6ce2145d4825e09012f29798e8f713ac
  9891432d2256b6f678e958f13081ee3081a1a00302010202
  0462319ea0300506032b6570301d311b301906035504030c
  124544484f4320526f6f742045643235353139301e170d32
  32303331363038323430305a170d32393132333132333030
  30305a30223120301e06035504030c174544484f4320496e
  69746961746f722045643235353139302a300506032b6570
  032100ed06a8ae61a829ba5fa54525c9d07f48dd44a302f4
  3e0f23d8cc20b73085141e300506032b6570034100521241
  d8b3a770996bcfc9b9ead4e7e0a1c0db353a3bdf2910b392
  75ae48b756015981850d27db6734e37f67212267dd05eeff
  27b9e7a813fa574b72a00b430b',
  h'51c968a7f9fdea19c7023f7022b4d9f214772ef588590524
  0576f62d036e69dc'
]
```

Message to be signed 3 (CBOR Data Item) (341 bytes)

```
84 6a 53 69 67 6e 61 74 75 72 65 31 4e a1 18 22 82 2e 48 c2 4a b2 fd
76 43 c7 9f 59 01 15 58 20 e0 91 12 1a f5 ac 6c e2 14 5d 48 25 e0 90
12 f2 97 98 e8 f7 13 ac 98 91 43 2d 22 56 b6 f6 78 e9 58 f1 30 81 ee
30 81 a1 a0 03 02 01 02 02 04 62 31 9e a0 30 05 06 03 2b 65 70 30 1d
31 1b 30 19 06 03 55 04 03 0c 12 45 44 48 4f 43 20 52 6f 6f 74 20 45
64 32 35 35 31 39 30 1e 17 0d 32 32 30 33 31 36 30 38 32 34 30 30 5a
17 0d 32 39 31 32 33 31 32 33 30 30 30 30 5a 30 22 31 20 30 1e 06 03
55 04 03 0c 17 45 44 48 4f 43 20 49 6e 69 74 69 61 74 6f 72 20 45 64
32 35 35 31 39 30 2a 30 05 06 03 2b 65 70 03 21 00 ed 06 a8 ae 61 a8
29 ba 5f a5 45 25 c9 d0 7f 48 dd 44 a3 02 f4 3e 0f 23 d8 cc 20 b7 30
85 14 1e 30 05 06 03 2b 65 70 03 41 00 52 12 41 d8 b3 a7 70 99 6b cf
c9 b9 ea d4 e7 e0 a1 c0 db 35 3a 3b df 29 10 b3 92 75 ae 48 b7 56 01
59 81 85 0d 27 db 67 34 e3 7f 67 21 22 67 dd 05 ee ff 27 b9 e7 a8 13
fa 57 4b 72 a0 0b 43 0b 58 20 51 c9 68 a7 f9 fd ea 19 c7 02 3f 70 22
b4 d9 f2 14 77 2e f5 88 59 05 24 05 76 f6 2d 03 6e 69 dc
```

The Initiator signs using the private authentication key SK\_I:

Signature\_or\_MAC\_3 (Raw Value) (64 bytes)

```
fc 10 7e c0 0f 74 ba 31 47 40 04 da 60 c5 b0 e1 eb 18 37 c0 f2 1e 00
81 6f bd bb e9 75 a8 05 68 3d 12 69 5b 1f a4 dc 71 f6 4c 6e 9e e9 32
0a 19 19 85 57 41 e2 7a 16 02 97 8a 13 4f 3e 57 4f 06
```

Signature\_or\_MAC\_3 (CBOR Data Item) (66 bytes)

```
58 40 fc 10 7e c0 0f 74 ba 31 47 40 04 da 60 c5 b0 e1 eb 18 37 c0 f2
1e 00 81 6f bd bb e9 75 a8 05 68 3d 12 69 5b 1f a4 dc 71 f6 4c 6e 9e
e9 32 0a 19 19 85 57 41 e2 7a 16 02 97 8a 13 4f 3e 57 4f 06
```

The Initiator constructs PLAINTEXT\_3:

PLAINTEXT\_3 =

```
(
  ID_CRED_I / bstr / -24..23,
  Signature_or_MAC_3,
  ? EAD_3
)
```

PLAINTEXT\_3 (CBOR Sequence) (80 bytes)

```
a1 18 22 82 2e 48 c2 4a b2 fd 76 43 c7 9f 58 40 fc 10 7e c0 0f 74 ba
31 47 40 04 da 60 c5 b0 e1 eb 18 37 c0 f2 1e 00 81 6f bd bb e9 75 a8
05 68 3d 12 69 5b 1f a4 dc 71 f6 4c 6e 9e e9 32 0a 19 19 85 57 41 e2
7a 16 02 97 8a 13 4f 3e 57 4f 06
```

The Initiator constructs the associated data for message\_3:

```
A_3 =
[
  "Encrypt0",
  h'',
  h'e091121af5ac6ce2145d4825e09012f29798e8f713ac9891
  432d2256b6f678e9'
]
```

A\_3 (CBOR Data Item) (45 bytes)  
83 68 45 6e 63 72 79 70 74 30 40 58 20 e0 91 12 1a f5 ac 6c e2 14 5d  
48 25 e0 90 12 f2 97 98 e8 f7 13 ac 98 91 43 2d 22 56 b6 f6 78 e9

The Initiator constructs the input needed to derive the key K\_3, see [Section 4.1.2](#) of [[I-D.ietf-lake-edhoc](#)], using the EDHOC hash algorithm:

```
K_3 = EDHOC_KDF( PRK_3e2m, 3, TH_3, key_length )
      = HKDF-Expand( PRK_3e2m, info, key_length ),
```

where key\_length is the key length in bytes for the EDHOC AEAD algorithm, and info for K\_3 is:

```
info =
(
  3,
  h'e091121af5ac6ce2145d4825e09012f29798e8f713ac9891
  432d2256b6f678e9',
  16
)
```

where the last value is the key length in bytes for the EDHOC AEAD algorithm.

info for K\_3 (CBOR Sequence) (36 bytes)  
03 58 20 e0 91 12 1a f5 ac 6c e2 14 5d 48 25 e0 90 12 f2 97 98 e8 f7  
13 ac 98 91 43 2d 22 56 b6 f6 78 e9 10

K\_3 (Raw Value) (16 bytes)  
95 65 a2 09 f6 7f d0 e1 62 9e 6f e7 c0 cc 3e 4a

The Initiator constructs the input needed to derive the nonce IV\_3, see [Section 4.1.2](#) of [[I-D.ietf-lake-edhoc](#)], using the EDHOC hash algorithm:

```
IV_3 = EDHOC_KDF( PRK_3e2m, 4, TH_3, iv_length )
      = HKDF-Expand( PRK_3e2m, info, iv_length ),
```

where iv\_length is the nonce length in bytes for the EDHOC AEAD algorithm, and info for IV\_3 is:

```
info =  
(  
  4,  
  h'e091121af5ac6ce2145d4825e09012f29798e8f713ac9891  
    432d2256b6f678e9',  
  13  
)
```

where the last value is the nonce length in bytes for the EDHOC AEAD algorithm.

```
info for IV_3 (CBOR Sequence) (36 bytes)  
04 58 20 e0 91 12 1a f5 ac 6c e2 14 5d 48 25 e0 90 12 f2 97 98 e8 f7  
13 ac 98 91 43 2d 22 56 b6 f6 78 e9 0d
```

```
IV_3 (Raw Value) (13 bytes)  
b6 a7 79 c4 b0 e7 40 fd 8d 77 4d 0a d6
```

The Initiator calculates CIPHERTEXT\_3 as 'ciphertext' of COSE\_Encrypt0 applied using the EDHOC AEAD algorithm with plaintext PLAINTEXT\_3, additional data A\_3, key K\_3 and nonce IV\_3.

```
CIPHERTEXT_3 (Raw Value) (88 bytes)  
aa 96 6a 1a a4 fa 44 9a 17 2a 16 0b 96 e6 44 f6 a3 33 29 f2 7c 6a f5  
bb ef c6 11 58 d0 ad dd 99 06 9b 9a 19 7f f7 c9 0e 62 f3 b5 56 64 c5  
83 74 7b 9a 40 2c cd 68 90 7f e4 58 b1 6a d5 2d 63 a0 0e 5a 85 df 95  
ee 7b 1b 49 8a c9 83 42 00 8c 04 71 c1 ae 8d 75 82 50 44
```

message\_3 is the CBOR bstr encoding of CIPHERTEXT\_3:

```
message_3 (CBOR Sequence) (90 bytes)  
58 58 aa 96 6a 1a a4 fa 44 9a 17 2a 16 0b 96 e6 44 f6 a3 33 29 f2 7c  
6a f5 bb ef c6 11 58 d0 ad dd 99 06 9b 9a 19 7f f7 c9 0e 62 f3 b5 56  
64 c5 83 74 7b 9a 40 2c cd 68 90 7f e4 58 b1 6a d5 2d 63 a0 0e 5a 85  
df 95 ee 7b 1b 49 8a c9 83 42 00 8c 04 71 c1 ae 8d 75 82 50 44
```

The transcript hash TH\_4 is calculated using the EDHOC hash algorithm:

```
TH_4 = H( TH_3, PLAINTEXT_3, CRED_I )
```

Input to calculate TH\_4 (CBOR Sequence) (357 bytes)

```
58 20 e0 91 12 1a f5 ac 6c e2 14 5d 48 25 e0 90 12 f2 97 98 e8 f7 13
ac 98 91 43 2d 22 56 b6 f6 78 e9 a1 18 22 82 2e 48 c2 4a b2 fd 76 43
c7 9f 58 40 fc 10 7e c0 0f 74 ba 31 47 40 04 da 60 c5 b0 e1 eb 18 37
c0 f2 1e 00 81 6f bd bb e9 75 a8 05 68 3d 12 69 5b 1f a4 dc 71 f6 4c
6e 9e e9 32 0a 19 19 85 57 41 e2 7a 16 02 97 8a 13 4f 3e 57 4f 06 58
f1 30 81 ee 30 81 a1 a0 03 02 01 02 02 04 62 31 9e a0 30 05 06 03 2b
65 70 30 1d 31 1b 30 19 06 03 55 04 03 0c 12 45 44 48 4f 43 20 52 6f
6f 74 20 45 64 32 35 35 31 39 30 1e 17 0d 32 32 30 33 31 36 30 38 32
34 30 30 5a 17 0d 32 39 31 32 33 31 32 33 30 30 30 30 5a 30 22 31 20
30 1e 06 03 55 04 03 0c 17 45 44 48 4f 43 20 49 6e 69 74 69 61 74 6f
72 20 45 64 32 35 35 31 39 30 2a 30 05 06 03 2b 65 70 03 21 00 ed 06
a8 ae 61 a8 29 ba 5f a5 45 25 c9 d0 7f 48 dd 44 a3 02 f4 3e 0f 23 d8
cc 20 b7 30 85 14 1e 30 05 06 03 2b 65 70 03 41 00 52 12 41 d8 b3 a7
70 99 6b cf c9 b9 ea d4 e7 e0 a1 c0 db 35 3a 3b df 29 10 b3 92 75 ae
48 b7 56 01 59 81 85 0d 27 db 67 34 e3 7f 67 21 22 67 dd 05 ee ff 27
b9 e7 a8 13 fa 57 4b 72 a0 0b 43 0b
```

TH\_4 (Raw Value) (32 bytes)

```
6b 13 32 5a 49 bd 9f 97 0d 31 91 ee 31 79 62 df 1d 44 38 c6 64 15 ea
a4 ce dd 62 b5 b4 9d 7b b7
```

TH\_4 (CBOR Data Item) (34 bytes)

```
58 20 6b 13 32 5a 49 bd 9f 97 0d 31 91 ee 31 79 62 df 1d 44 38 c6 64
15 ea a4 ce dd 62 b5 b4 9d 7b b7
```

#### 2.4. message\_4

No external authorization data:

EAD\_4 (CBOR Sequence) (0 bytes)

The Responder constructs PLAINTEXT\_4:

PLAINTEXT\_4 =

```
(
  ? EAD_4
)
```

PLAINTEXT\_4 (CBOR Sequence) (0 bytes)

The Responder constructs the associated data for message\_4:

A\_4 =

```
[
  "Encrypt0",
  h'',
  h'6b13325a49bd9f970d3191ee317962df1d4438c66415aaa4
  cedd62b5b49d7bb7'
]
```

A\_4 (CBOR Data Item) (45 bytes)

```
83 68 45 6e 63 72 79 70 74 30 40 58 20 6b 13 32 5a 49 bd 9f 97 0d 31
91 ee 31 79 62 df 1d 44 38 c6 64 15 ea a4 ce dd 62 b5 b4 9d 7b b7
```

The Responder constructs the input needed to derive the EDHOC message\_4 key, see [Section 4.1.2](#) of [[I-D.ietf-lake-edhoc](#)], using the EDHOC hash algorithm:

```
K_4 = EDHOC_KDF( PRK_4e3m, 8, TH_4, key_length )
      = HKDF-Expand( PRK_4x3m, info, key_length )
```

where key\_length is the key length in bytes for the EDHOC AEAD algorithm, and info for K\_4 is:

```
info =
(
  8,
  h'6b13325a49bd9f970d3191ee317962df1d4438c66415eaa4
  cedd62b5b49d7bb7',
  16
)
```

where the last value is the key length in bytes for the EDHOC AEAD algorithm.

info for K\_4 (CBOR Sequence) (36 bytes)

```
08 58 20 6b 13 32 5a 49 bd 9f 97 0d 31 91 ee 31 79 62 df 1d 44 38 c6
64 15 ea a4 ce dd 62 b5 b4 9d 7b b7 10
```

K\_4 (Raw Value) (16 bytes)

```
c9 f5 87 9d dd 4e 25 68 f6 94 46 c3 06 52 5f ef
```

The Responder constructs the input needed to derive the EDHOC message\_4 nonce, see [Section 4.1.2](#) of [[I-D.ietf-lake-edhoc](#)], using the EDHOC hash algorithm:

```
IV_4 = EDHOC_KDF( PRK_4e3m, 9, TH_4, iv_length )
      = HKDF-Expand( PRK_4x3m, info, iv_length )
```

where length is the nonce length in bytes for the EDHOC AEAD algorithm, and info for IV\_4 is:

```
info =
(
  9,
  h'6b13325a49bd9f970d3191ee317962df1d4438c66415eaa4
  cedd62b5b49d7bb7',
  13
)
```



where the last value is the nonce length in bytes for the EDHOC AEAD algorithm.

info for IV\_4 (CBOR Sequence) (36 bytes)

```
09 58 20 6b 13 32 5a 49 bd 9f 97 0d 31 91 ee 31 79 62 df 1d 44 38 c6
64 15 ea a4 ce dd 62 b5 b4 9d 7b b7 0d
```

IV\_4 (Raw Value) (13 bytes)

```
a8 e0 4c e7 56 ee 38 e8 23 b7 7b 3e e0
```

The Responder calculates CIPHERTEXT\_4 as 'ciphertext' of COSE\_Encrypt0 applied using the EDHOC AEAD algorithm with plaintext PLAINTEXT\_4, additional data A\_4, key K\_4 and nonce IV\_4.

CIPHERTEXT\_4 (8 bytes)

```
ee 12 0e 8b 5e 2a 00 8f
```

message\_4 is the CBOR bstr encoding of CIPHERTEXT\_4:

message\_4 (CBOR Sequence) (9 bytes)

```
48 ee 12 0e 8b 5e 2a 00 8f
```

## 2.5. PRK\_out and PRK\_exporter

PRK\_out is specified in [Section 4.1.3](#) of [[I-D.ietf-lake-edhoc](#)].

```
PRK_out = EDHOC_KDF( PRK_4e3m, 7, TH_4, hash_length ) =
           = HKDF-Expand( PRK_4e3m, info, hash_length )
```

where hash\_length is the length in bytes of the output of the EDHOC hash algorithm, and info for PRK\_out is:

```
info =
(
  7,
  h'6b13325a49bd9f970d3191ee317962df1d4438c66415eaa4
  cedd62b5b49d7bb7',
  32
)
```

where the last value is the length in bytes of the output of the EDHOC hash algorithm.

info for PRK\_out (CBOR Sequence) (37 bytes)

```
07 58 20 6b 13 32 5a 49 bd 9f 97 0d 31 91 ee 31 79 62 df 1d 44 38 c6
64 15 ea a4 ce dd 62 b5 b4 9d 7b b7 18 20
```

PRK\_out (Raw Value) (32 bytes)

```
45 06 92 9a d5 95 d5 d4 e5 9b 5f 21 ea b6 7d ea b6 4a 3b d2 c7 d9 d6
87 7d 60 61 81 9c 2d 02 0d
```

The OSCORE Master Secret and OSCORE Master Salt are derived with the EDHOC\_Exporter as specified in [Section 4.2.1](#) of [\[I-D.ietf-lake-edhoc\]](#).

```
EDHOC_Exporter( label, context, length )  
= EDHOC_KDF( PRK_exporter, label, context, length )
```

where PRK\_exporter is derived from PRK\_out:

```
PRK_exporter = EDHOC_KDF( PRK_out, 10, h'', hash_length ) =  
              = HKDF-Expand( PRK_out, info, hash_length )
```

where hash\_length is the length in bytes of the output of the EDHOC hash algorithm, and info for the PRK\_exporter is:

```
info =  
(  
  10,  
  h'',  
  32  
)
```

where the last value is the length in bytes of the output of the EDHOC hash algorithm.

```
info for PRK_exporter (CBOR Sequence) (4 bytes)  
0a 40 18 20
```

```
PRK_exporter (Raw Value) (32 bytes)  
ad 33 a8 f2 e0 6f ff 3e 5d 7e e1 10 9e db f2 b6 d2 56 4c b3 f4 08 68  
e6 46 11 e4 20 92 4c e4 09
```

## 2.6. OSCORE Parameters

The derivation of OSCORE parameters is specified in [Appendix A.1](#) of [\[I-D.ietf-lake-edhoc\]](#).

The AEAD and Hash algorithms to use in OSCORE are given by the selected cipher suite:

```
Application AEAD Algorithm (int)  
10
```

```
Application Hash Algorithm (int)  
-16
```

The mapping from EDHOC connection identifiers to OSCORE Sender/Recipient IDs is defined in [Section 3.3.3](#) of [\[I-D.ietf-lake-edhoc\]](#).

C\_R is mapped to the Recipient ID of the server, i.e., the Sender ID of the client. The byte string 0x18, which as C\_R is encoded as the CBOR byte string 0x4118, is converted to the server Recipient ID 0x18.

Client's OSCORE Sender ID (Raw Value) (1 byte)  
18

C\_I is mapped to the Recipient ID of the client, i.e., the Sender ID of the server. The byte string 0x2d, which as C\_I is encoded as the CBOR integer 0x2d is converted to the client Recipient ID 0x2d.

Server's OSCORE Sender ID (Raw Value) (1 byte)  
2d

The OSCORE Master Secret is computed through EDHOC\_Expand() using the Application hash algorithm, see [Appendix A.1](#) of [\[I-D.ietf-lake-edhoc\]](#):

```
OSCORE Master Secret = EDHOC_Exporter( 0, h'', oscore_key_length )
= EDHOC_KDF( PRK_exporter, 0, h'', oscore_key_length )
= HKDF-Expand( PRK_exporter, info, oscore_key_length )
```

where oscore\_key\_length is by default the key length in bytes for the Application AEAD algorithm, and info for the OSCORE Master Secret is:

```
info =
(
  0,
  h'',
  16
)
```

where the last value is the key length in bytes for the Application AEAD algorithm.

info for OSCORE Master Secret (CBOR Sequence) (3 bytes)  
00 40 10

OSCORE Master Secret (Raw Value) (16 bytes)  
fc 9c fb 05 63 ca 3e 28 f8 80 48 3b 9c 06 bd 03

The OSCORE Master Salt is computed through EDHOC\_Expand() using the Application hash algorithm, see [Section 4.2](#) of [\[I-D.ietf-lake-edhoc\]](#):

```
OSCORE Master Salt = EDHOC_Exporter( 1, h'', oscore_salt_length )
= EDHOC_KDF( PRK_exporter, 1, h'', oscore_salt_length )
= HKDF-Expand( PRK_4x3m, info, oscore_salt_length )
```

where `oscore_salt_length` is the length in bytes of the OSCORE Master Salt, and `info` for the OSCORE Master Salt is:

```
info =  
(  
  1,  
  h'',  
  8  
)
```

where the last value is the length in bytes of the OSCORE Master Salt.

info for OSCORE Master Salt (CBOR Sequence) (3 bytes)  
01 40 08

OSCORE Master Salt (Raw Value) (8 bytes)  
0e c0 9d 45 3b 08 98 34

## 2.7. Key Update

Key update is defined in [Appendix H](#) of [[I-D.ietf-lake-edhoc](#)].

```
EDHOC_KeyUpdate( context ):  
PRK_out = EDHOC_KDF( PRK_out, 11, context, hash_length )  
          = HKDF-Expand( PRK_out, info, hash_length )
```

where `hash_length` is the length in bytes of the output of the EDHOC hash function, and `context` for `KeyUpdate` is

context for `KeyUpdate` (Raw Value) (16 bytes)  
d6 be 16 96 02 b8 bc ea a0 11 58 fd b8 20 89 0c

context for `KeyUpdate` (CBOR Data Item) (17 bytes)  
50 d6 be 16 96 02 b8 bc ea a0 11 58 fd b8 20 89 0c

and where `info` for key update is:

```
info =  
(  
  11,  
  h'd6be169602b8bceaa01158fdb820890c',  
  32  
)
```

PRK\_out after `KeyUpdate` (Raw Value) (32 bytes)  
0c 1d e2 f0 6d 9a d7 5a 21 32 90 5f 95 c6 96 40 42 76 af 81 f1 14 4a  
a7 61 af bf 78 d6 8c a1 b4

After key update, the `PRK_exporter` needs to be derived anew:

```
PRK_exporter = EDHOC_KDF( PRK_out, 10, h'', hash_length ) =  
= HKDF-Expand( PRK_out, info, hash_length )
```

where info and hash\_length are unchanged as in [Section 2.5](#).

PRK\_exporter (Raw Value) (32 bytes)

```
f0 4e 4c 40 1d e8 db 34 f7 b5 06 b2 33 10 9a 24 c4 9c 4b 09 65 d0 7c  
6e 47 7b 23 a3 7b 53 c2 35
```

The OSCORE Master Secret is derived with the updated PRK\_exporter:

```
OSCORE Master Secret =  
= HKDF-Expand(PRK_exporter, info, oscore_key_length)
```

where info and key\_length are unchanged as in [Section 2.6](#).

OSCORE Master Secret after KeyUpdate (Raw Value) (16 bytes)

```
50 48 6d 75 82 3a 59 2d 1e fd 28 6a 70 7f e8 7d
```

The OSCORE Master Salt is derived with the updated PRK\_exporter:

```
OSCORE Master Salt = HKDF-Expand(PRK_exporter, info, salt_length)
```

where info and salt\_length are unchanged as in [Section 2.6](#).

OSCORE Master Salt after KeyUpdate (Raw Value) (8 bytes)

```
61 95 cb b1 ce 03 1c ae
```

## 2.8. Certificates

### 2.8.1. Responder Certificate

Version: 3 (0x2)  
Serial Number: 1647419076 (0x62319ec4)  
Signature Algorithm: ED25519  
Issuer: CN = EDHOC Root Ed25519  
Validity  
    Not Before: Mar 16 08:24:36 2022 GMT  
    Not After : Dec 31 23:00:00 2029 GMT  
Subject: CN = EDHOC Responder Ed25519  
Subject Public Key Info:  
    Public Key Algorithm: ED25519  
    ED25519 Public-Key:  
    pub:  
        a1 db 47 b9 51 84 85 4a d1 2a 0c 1a 35 4e 41  
        8a ac e3 3a a0 f2 c6 62 c0 0b 3a c5 5d e9 2f  
        93 59  
Signature Algorithm: ED25519  
Signature Value:  
    b7 23 bc 01 ea b0 92 8e 8b 2b 6c 98 de 19 cc 38 23 d4  
    6e 7d 69 87 b0 32 47 8f ec fa f1 45 37 a1 af 14 cc 8b  
    e8 29 c6 b7 30 44 10 18 37 eb 4a bc 94 95 65 d8 6d ce  
    51 cf ae 52 ab 82 c1 52 cb 02

### 2.8.2. Initiator Certificate

Version: 3 (0x2)  
Serial Number: 1647419040 (0x62319ea0)  
Signature Algorithm: ED25519  
Issuer: CN = EDHOC Root Ed25519  
Validity  
    Not Before: Mar 16 08:24:00 2022 GMT  
    Not After : Dec 31 23:00:00 2029 GMT  
Subject: CN = EDHOC Initiator Ed25519  
Subject Public Key Info:  
    Public Key Algorithm: ED25519  
    ED25519 Public-Key:  
    pub:  
        ed 06 a8 ae 61 a8 29 ba 5f a5 45 25 c9 d0 7f  
        48 dd 44 a3 02 f4 3e 0f 23 d8 cc 20 b7 30 85  
        14 1e  
Signature Algorithm: ED25519  
Signature Value:  
    52 12 41 d8 b3 a7 70 99 6b cf c9 b9 ea d4 e7 e0 a1 c0  
    db 35 3a 3b df 29 10 b3 92 75 ae 48 b7 56 01 59 81 85  
    0d 27 db 67 34 e3 7f 67 21 22 67 dd 05 ee ff 27 b9 e7  
    a8 13 fa 57 4b 72 a0 0b 43 0b

### 2.8.3. Common Root Certificate

```
Version: 3 (0x2)
Serial Number: 1647418996 (0x62319e74)
Signature Algorithm: ED25519
Issuer: CN = EDHOC Root Ed25519
Validity
    Not Before: Mar 16 08:23:16 2022 GMT
    Not After : Dec 31 23:00:00 2029 GMT
Subject: CN = EDHOC Root Ed25519
Subject Public Key Info:
    Public Key Algorithm: ED25519
    ED25519 Public-Key:
    pub:
        2b 7b 3e 80 57 c8 64 29 44 d0 6a fe 7a 71 d1
        c9 bf 96 1b 62 92 ba c4 b0 4f 91 66 9b bb 71
        3b e4
X509v3 extensions:
    X509v3 Key Usage: critical
        Certificate Sign
    X509v3 Basic Constraints: critical
        CA:TRUE
Signature Algorithm: ED25519
Signature Value:
    4b b5 2b bf 15 39 b7 1a 4a af 42 97 78 f2 9e da 7e 81
    46 80 69 8f 16 c4 8f 2a 6f a4 db e8 25 41 c5 82 07 ba
    1b c9 cd b0 c2 fa 94 7f fb f0 f0 ec 0e e9 1a 7f f3 7a
    94 d9 25 1f a5 cd f1 e6 7a 0f
```

### 3. Authentication with Static DH, CCS Identified by 'kid'

In this example the Initiator and the Responder are authenticated with ephemeral-static Diffie-Hellman (METHOD = 3). The Initiator supports cipher suites 6 and 2 (in order of preference) and the Responder only supports cipher suite 2. After an initial negotiation message exchange, cipher suite 2 is used, which determines the algorithms:

- \*EDHOC AEAD algorithm = AES-CCM-16-64-128
- \*EDHOC hash algorithm = SHA-256
- \*EDHOC MAC length in bytes (Static DH) = 8
- \*EDHOC key exchange algorithm (ECDH curve) = P-256
- \*EDHOC signature algorithm = ES256
- \*Application AEAD algorithm = AES-CCM-16-64-128

\*Application hash algorithm = SHA-256

The public keys are represented as raw public keys (RPK), encoded in a CWT Claims Set (CCS) and identified by the COSE header parameter 'kid'.

### 3.1. message\_1 (first time)

Both endpoints are authenticated with static DH, i.e., METHOD = 3:

METHOD (CBOR Data Item) (1 byte)

03

The Initiator selects its preferred cipher suite 6. A single cipher suite is encoded as an int:

SUITES\_I (CBOR Data Item) (1 byte)

06

The Initiator creates an ephemeral key pair for use with the EDHOC key exchange algorithm:

Initiator's ephemeral private key

X (Raw Value) (32 bytes)

5c 41 72 ac a8 b8 2b 5a 62 e6 6f 72 22 16 f5 a1 0f 72 aa 69 f4 2c 1d  
1c d3 cc d7 bf d2 9c a4 e9

Initiator's ephemeral public key, 'x'-coordinate

G\_X (Raw Value) (32 bytes)

74 1a 13 d7 ba 04 8f bb 61 5e 94 38 6a a3 b6 1b ea 5b 3d 8f 65 f3 26  
20 b7 49 be e8 d2 78 ef a9

Initiator's ephemeral public key, 'y'-coordinate

G\_Y (CBOR Data Item) (34 bytes)

58 20 74 1a 13 d7 ba 04 8f bb 61 5e 94 38 6a a3 b6 1b ea 5b 3d 8f 65  
f3 26 20 b7 49 be e8 d2 78 ef a9

The Initiator selects its connection identifier C\_I to be the byte string 0x0e, which since it is represented by the 1-byte CBOR int 14 is encoded as 0x0e:

Connection identifier chosen by Initiator

C\_I (Raw Value) (1 byte)

0e

Connection identifier chosen by Initiator

C\_I (CBOR Data Item) (1 byte)

0e

No external authorization data:



EAD\_1 (CBOR Sequence) (0 bytes)

The Initiator constructs message\_1:

```
message_1 =  
(  
  3,  
  6,  
  h'741a13d7ba048fbb615e94386aa3b61bea5b3d8f65f32620  
    b749bee8d278efa9',  
  14  
)
```

message\_1 (CBOR Sequence) (37 bytes)

```
03 06 58 20 74 1a 13 d7 ba 04 8f bb 61 5e 94 38 6a a3 b6 1b ea 5b 3d  
8f 65 f3 26 20 b7 49 be e8 d2 78 ef a9 0e
```

### 3.2. error

The Responder does not support cipher suite 6 and sends an error with ERR\_CODE 2 containing SUITES\_R as ERR\_INFO. The Responder proposes cipher suite 2, a single cipher suite thus encoded as an int.

```
SUITES_R  
02
```

```
error (CBOR Sequence) (2 bytes)  
02 02
```

### 3.3. message\_1 (second time)

Same steps are performed as for message\_1 the first time, [Section 3.1](#), but with updated SUITES\_I.

Both endpoints are authenticated with static DH, i.e., METHOD = 3:

```
METHOD (CBOR Data Item) (1 byte)  
03
```

The Initiator selects cipher suite 2 and indicates the more preferred cipher suite(s), in this case 6, all encoded as the array [6, 2]:

```
SUITES_I (CBOR Data Item) (3 bytes)  
82 06 02
```

The Initiator creates an ephemeral key pair for use with the EDHOC key exchange algorithm:

Initiator's ephemeral private key

X (Raw Value) (32 bytes)

36 8e c1 f6 9a eb 65 9b a3 7d 5a 8d 45 b2 1b dc 02 99 dc ea a8 ef 23  
5f 3c a4 2c e3 53 0f 95 25

Initiator's ephemeral public key, 'x'-coordinate

G\_X (Raw Value) (32 bytes)

8a f6 f4 30 eb e1 8d 34 18 40 17 a9 a1 1b f5 11 c8 df f8 f8 34 73 0b  
96 c1 b7 c8 db ca 2f c3 b6

Initiator's ephemeral public key, one 'y'-coordinate

(Raw Value) (32 bytes)

51 e8 af 6c 6e db 78 16 01 ad 1d 9c 5f a8 bf 7a a1 57 16 c7 c0 6a 5d  
03 85 03 c6 14 ff 80 c9 b3

Initiator's ephemeral public key, 'x'-coordinate

G\_X (CBOR Data Item) (34 bytes)

58 20 8a f6 f4 30 eb e1 8d 34 18 40 17 a9 a1 1b f5 11 c8 df f8 f8 34  
73 0b 96 c1 b7 c8 db ca 2f c3 b6

The Initiator selects its connection identifier C\_I to be the byte string 0x37, which since it is represented by the 1-byte CBOR int -24 is encoded as 0x37:

Connection identifier chosen by Initiator

C\_I (Raw Value) (1 byte)

37

Connection identifier chosen by Initiator

C\_I (CBOR Data Item) (1 byte)

37

No external authorization data:

EAD\_1 (CBOR Sequence) (0 bytes)

The Initiator constructs message\_1:

message\_1 =

```
(  
  3,  
  [6, 2],  
  h'8af6f430ebe18d34184017a9a11bf511c8dff8f834730b96  
    c1b7c8dbca2fc3b6',  
  -24  
)
```

message\_1 (CBOR Sequence) (39 bytes)

03 82 06 02 58 20 8a f6 f4 30 eb e1 8d 34 18 40 17 a9 a1 1b f5 11 c8  
df f8 f8 34 73 0b 96 c1 b7 c8 db ca 2f c3 b6 37

### 3.4. message\_2

The Responder supports the selected cipher suite 2 and not the by the Initiator more preferred cipher suite(s) 6, so SUITES\_I is acceptable.

The Responder creates an ephemeral key pair for use with the EDHOC key exchange algorithm:

Responder's ephemeral private key

Y (Raw Value) (32 bytes)

e2 f4 12 67 77 20 5e 85 3b 43 7d 6e ac a1 e1 f7 53 cd cc 3e 2c 69 fa  
88 4b 0a 1a 64 09 77 e4 18

Responder's ephemeral public key, 'x'-coordinate

G\_Y (Raw Value) (32 bytes)

41 97 01 d7 f0 0a 26 c2 dc 58 7a 36 dd 75 25 49 f3 37 63 c8 93 42 2c  
8e a0 f9 55 a1 3a 4f f5 d5

Responder's ephemeral public key, one 'y'-coordinate

(Raw Value) (32 bytes)

5e 4f 0d d8 a3 da 0b aa 16 b9 d3 ad 56 a0 c1 86 0a 94 0a f8 59 14 91  
5e 25 01 9b 40 24 17 e9 9d

Responder's ephemeral public key, 'x'-coordinate

G\_Y (CBOR Data Item) (34 bytes)

58 20 41 97 01 d7 f0 0a 26 c2 dc 58 7a 36 dd 75 25 49 f3 37 63 c8 93  
42 2c 8e a0 f9 55 a1 3a 4f f5 d5

The Responder selects its connection identifier C\_R to be the byte string 0x27, which since it is represented by the 1-byte CBOR int -8 is encoded as 0x27:

Connection identifier chosen by Responder

C\_R (raw value) (1 byte)

27

Connection identifier chosen by Responder

C\_R (CBOR Data Item) (1 byte)

27

The transcript hash TH\_2 is calculated using the EDHOC hash algorithm:

TH\_2 = H( G\_Y, H(message\_1) )

H(message\_1) (Raw Value) (32 bytes)

ca 02 ca bd a5 a8 90 27 49 b4 2f 71 10 50 bb 4d bd 52 15 3e 87 52 75  
94 b3 9f 50 cd f0 19 88 8c

H(message\_1) (CBOR Data Item) (34 bytes)  
58 20 ca 02 ca bd a5 a8 90 27 49 b4 2f 71 10 50 bb 4d bd 52 15 3e 87  
52 75 94 b3 9f 50 cd f0 19 88 8c

The input to calculate TH\_2 is the CBOR sequence:

G\_Y, H(message\_1)

Input to calculate TH\_2 (CBOR Sequence) (68 bytes)  
58 20 41 97 01 d7 f0 0a 26 c2 dc 58 7a 36 dd 75 25 49 f3 37 63 c8 93  
42 2c 8e a0 f9 55 a1 3a 4f f5 d5 58 20 ca 02 ca bd a5 a8 90 27 49 b4  
2f 71 10 50 bb 4d bd 52 15 3e 87 52 75 94 b3 9f 50 cd f0 19 88 8c

TH\_2 (Raw Value) (32 bytes)  
35 6e fd 53 77 14 25 e0 08 f3 fe 3a 86 c8 3f f4 c6 b1 6e 57 02 8f f3  
9d 52 36 c1 82 b2 02 08 4b

TH\_2 (CBOR Data Item) (34 bytes)  
58 20 35 6e fd 53 77 14 25 e0 08 f3 fe 3a 86 c8 3f f4 c6 b1 6e 57 02  
8f f3 9d 52 36 c1 82 b2 02 08 4b

PRK\_2e is specified in [Section 4.1.1.1](#) of [[I-D.ietf-lake-edhoc](#)].

First, the ECDH shared secret G\_XY is computed from G\_X and Y, or G\_Y and X:

G\_XY (Raw Value) (ECDH shared secret) (32 bytes)  
2f 0c b7 e8 60 ba 53 8f bf 5c 8b de d0 09 f6 25 9b 4b 62 8f e1 eb 7d  
be 93 78 e5 ec f7 a8 24 ba

Then, PRK\_2e is calculated using EDHOC\_Extract() determined by the EDHOC hash algorithm:

PRK\_2e = EDHOC\_Extract( salt, G\_XY ) =  
= HMAC-SHA-256( salt, G\_XY )

where salt is TH\_2:

salt (Raw Value) (32 bytes)  
35 6e fd 53 77 14 25 e0 08 f3 fe 3a 86 c8 3f f4 c6 b1 6e 57 02 8f f3  
9d 52 36 c1 82 b2 02 08 4b

PRK\_2e (Raw Value) (32 bytes)  
5a a0 d6 9f 3e 3d 1e 0c 47 9f 0b 8a 48 66 90 c9 80 26 30 c3 46 6b 1d  
c9 23 71 c9 82 56 31 70 b5

Since METHOD = 3, the Responder authenticates using static DH. The EDHOC key exchange algorithm is based on the same curve as for the ephemeral keys, which is P-256, since the selected cipher suite is 2.

The Responder's static Diffie-Hellman P-256 key pair:

Responder's private authentication key

SK\_R (Raw Value) (32 bytes)

72 cc 47 61 db d4 c7 8f 75 89 31 aa 58 9d 34 8d 1e f8 74 a7 e3 03 ed  
e2 f1 40 dc f3 e6 aa 4a ac

Responder's public authentication key, 'x'-coordinate

(Raw Value) (32 bytes)

bb c3 49 60 52 6e a4 d3 2e 94 0c ad 2a 23 41 48 dd c2 17 91 a1 2a fb  
cb ac 93 62 20 46 dd 44 f0

Responder's public authentication key, 'y'-coordinate

(Raw Value) (32 bytes)

45 19 e2 57 23 6b 2a 0c e2 02 3f 09 31 f1 f3 86 ca 7a fd a6 4f cd e0  
10 8c 22 4c 51 ea bf 60 72

Since the Responder authenticates with static DH (METHOD = 3),  
PRK\_3e2m is derived from SALT\_3e2m and G\_RX.

The input needed to calculate SALT\_3e2m is defined in [Section 4.1.2](#)  
of [[I-D.ietf-lake-edhoc](#)], using EDHOC\_Expand() with the EDHOC hash  
algorithm:

```
SALT_3e2m = EDHOC_KDF( PRK_2e, 1, TH_2, hash_length ) =  
            = HKDF-Expand( PRK_2e, info, hash_length )
```

where hash\_length is the length in bytes of the output of the EDHOC  
hash algorithm, and info for SALT\_3e2m is:

```
info =  
(  
  1,  
  h'356efd53771425e008f3fe3a86c83ff4c6b16e57028ff39d  
    5236c182b202084b',  
  32  
)
```

info for SALT\_3e2m (CBOR Sequence) (37 bytes)

01 58 20 35 6e fd 53 77 14 25 e0 08 f3 fe 3a 86 c8 3f f4 c6 b1 6e 57  
02 8f f3 9d 52 36 c1 82 b2 02 08 4b 18 20

SALT\_3e2m (Raw Value) (32 bytes)

af 4e 10 3a 47 cb 3c f3 25 70 d5 c2 5a d2 77 32 bd 8d 81 78 e9 a6 9d  
06 1c 31 a2 7f 8e 3c a9 26

PRK\_3e2m is specified in [Section 4.1.1.2](#) of [[I-D.ietf-lake-edhoc](#)].

PRK\_3e2m is derived from G\_RX using EDHOC\_Extract() with the EDHOC  
hash algorithm:

```
PRK_3e2m = EDHOC_Extract( SALT_3e2m, G_RX ) =
           = HMAC-SHA-256( SALT_3e2m, G_RX )
```

where G\_RX is the ECDH shared secret calculated from G\_X and R, or G\_R and X.

G\_RX (Raw Value) (ECDH shared secret) (32 bytes)

```
f2 b6 ee a0 22 20 b9 5e ee 5a 0b c7 01 f0 74 e0 0a 84 3e a0 24 22 f6
08 25 fb 26 9b 3e 16 14 23
```

PRK\_3e2m (Raw Value) (32 bytes)

```
0c a3 d3 39 82 96 b3 c0 39 00 98 76 20 c1 1f 6f ce 70 78 1c 1d 12 19
72 0f 9e c0 8c 12 2d 84 34
```

The Responder constructs the remaining input needed to calculate MAC\_2:

```
MAC_2 = EDHOC_KDF( PRK_3e2m, 2, context_2, mac_length_2 )
```

```
context_2 = << ID_CRED_R, TH_2, CRED_R, ? EAD_2 >>
```

CRED\_R is identified by a 'kid' with byte string value 0x32:

ID\_CRED\_R =

```
{
  4 : h'32'
}
```

ID\_CRED\_R (CBOR Data Item) (4 bytes)

```
a1 04 41 32
```

CRED\_R is an RPK encoded as a CCS:

```
{
  2 : "example.edu", /CCS/
  8 : { /sub/
    1 : { /cnf/
      1 : 2, /COSE_Key/
      2 : h'32', /kty/
      -1 : 1, /kid/
      -2 : h'BBC34960526EA4D32E940CAD2A234148
          DDC21791A12AFBCBAC93622046DD44F0', /crv/
      -3 : h'4519E257236B2A0CE2023F0931F1F386
          CA7AFDA64FCDE0108C224C51EABF6072' /x/
    } /y/
  }
}
```

CRED\_R (CBOR Data Item) (95 bytes)

```
a2 02 6b 65 78 61 6d 70 6c 65 2e 65 64 75 08 a1 01 a5 01 02 02 41 32
20 01 21 58 20 bb c3 49 60 52 6e a4 d3 2e 94 0c ad 2a 23 41 48 dd c2
17 91 a1 2a fb cb ac 93 62 20 46 dd 44 f0 22 58 20 45 19 e2 57 23 6b
2a 0c e2 02 3f 09 31 f1 f3 86 ca 7a fd a6 4f cd e0 10 8c 22 4c 51 ea
bf 60 72
```

No external authorization data:

EAD\_2 (CBOR Sequence) (0 bytes)

```
context_2 = << ID_CRED_R, TH_2, CRED_R, ? EAD_2 >>
```

context\_2 (CBOR Sequence) (133 bytes)

```
a1 04 41 32 58 20 35 6e fd 53 77 14 25 e0 08 f3 fe 3a 86 c8 3f f4 c6
b1 6e 57 02 8f f3 9d 52 36 c1 82 b2 02 08 4b a2 02 6b 65 78 61 6d 70
6c 65 2e 65 64 75 08 a1 01 a5 01 02 02 41 32 20 01 21 58 20 bb c3 49
60 52 6e a4 d3 2e 94 0c ad 2a 23 41 48 dd c2 17 91 a1 2a fb cb ac 93
62 20 46 dd 44 f0 22 58 20 45 19 e2 57 23 6b 2a 0c e2 02 3f 09 31 f1
f3 86 ca 7a fd a6 4f cd e0 10 8c 22 4c 51 ea bf 60 72
```

context\_2 (CBOR byte string) (135 bytes)

```
58 85 a1 04 41 32 58 20 35 6e fd 53 77 14 25 e0 08 f3 fe 3a 86 c8 3f
f4 c6 b1 6e 57 02 8f f3 9d 52 36 c1 82 b2 02 08 4b a2 02 6b 65 78 61
6d 70 6c 65 2e 65 64 75 08 a1 01 a5 01 02 02 41 32 20 01 21 58 20 bb
c3 49 60 52 6e a4 d3 2e 94 0c ad 2a 23 41 48 dd c2 17 91 a1 2a fb cb
ac 93 62 20 46 dd 44 f0 22 58 20 45 19 e2 57 23 6b 2a 0c e2 02 3f 09
31 f1 f3 86 ca 7a fd a6 4f cd e0 10 8c 22 4c 51 ea bf 60 72
```

MAC\_2 is computed through EDHOC\_Expand() using the EDHOC hash algorithm, see [Section 4.1.2](#) of [[I-D.ietf-lake-edhoc](#)]:

MAC\_2 = HKDF-Expand(PRK\_3e2m, info, mac\_length\_2), where

info = ( 2, context\_2, mac\_length\_2 )

Since METHOD = 3, mac\_length\_2 is given by the EDHOC MAC length.

info for MAC\_2 is:

```
info =
(
  2,
  h'a10441325820356efd53771425e008f3fe3a86c83ff4c6b1
  6e57028ff39d5236c182b202084ba2026b6578616d706c65
  2e65647508a101a501020241322001215820bbc34960526e
  a4d32e940cad2a234148ddc21791a12afbcbac93622046dd
  44f02258204519e257236b2a0ce2023f0931f1f386ca7afd
  a64fcde0108c224c51eabf6072',
  8
)
```

where the last value is the EDHOC MAC length in bytes.

info for MAC\_2 (CBOR Sequence) (137 bytes)

```
02 58 85 a1 04 41 32 58 20 35 6e fd 53 77 14 25 e0 08 f3 fe 3a 86 c8
3f f4 c6 b1 6e 57 02 8f f3 9d 52 36 c1 82 b2 02 08 4b a2 02 6b 65 78
61 6d 70 6c 65 2e 65 64 75 08 a1 01 a5 01 02 02 41 32 20 01 21 58 20
bb c3 49 60 52 6e a4 d3 2e 94 0c ad 2a 23 41 48 dd c2 17 91 a1 2a fb
cb ac 93 62 20 46 dd 44 f0 22 58 20 45 19 e2 57 23 6b 2a 0c e2 02 3f
09 31 f1 f3 86 ca 7a fd a6 4f cd e0 10 8c 22 4c 51 ea bf 60 72 08
```

MAC\_2 (Raw Value) (8 bytes)

```
fa 5e fa 2e bf 92 0b f3
```

MAC\_2 (CBOR Data Item) (9 bytes)

```
48 fa 5e fa 2e bf 92 0b f3
```

Since METHOD = 3, Signature\_or\_MAC\_2 is MAC\_2:

Signature\_or\_MAC\_2 (Raw Value) (8 bytes)

```
fa 5e fa 2e bf 92 0b f3
```

Signature\_or\_MAC\_2 (CBOR Data Item) (9 bytes)

```
48 fa 5e fa 2e bf 92 0b f3
```

The Responder constructs PLAINTEXT\_2:

PLAINTEXT\_2 =

```
(
  C_R,
  ID_CRED_R / bstr / -24..23,
  Signature_or_MAC_2,
  ? EAD_2
)
```

Since ID\_CRED\_R contains a single 'kid' parameter, only the byte string value is included in the plaintext, represented as described in [Section 3.3.2](#) of [\[I-D.ietf-lake-edhoc\]](#). The CBOR map { 4 : h'32' } is thus replaced, not by the CBOR byte string 0x4132, but by the



CBOR int 0x32, since that is a one byte encoding of a CBOR integer (-19).

PLAINTEXT\_2 (CBOR Sequence) (11 bytes)  
27 32 48 fa 5e fa 2e bf 92 0b f3

The input needed to calculate KEYSTREAM\_2 is defined in [Section 4.1.2](#) of [[I-D.ietf-lake-edhoc](#)], using EDHOC\_Expand() with the EDHOC hash algorithm:

```
KEYSTREAM_2 = EDHOC_KDF( PRK_2e, 0, TH_2, plaintext_length ) =  
              = HKDF-Expand( PRK_2e, info, plaintext_length )
```

where plaintext\_length is the length in bytes of PLAINTEXT\_2, and info for KEYSTREAM\_2 is:

```
info =  
(  
  0,  
  h'356efd53771425e008f3fe3a86c83ff4c6b16e57028ff39d  
    5236c182b202084b',  
  11  
)
```

where the last value is the length in bytes of PLAINTEXT\_2.

info for KEYSTREAM\_2 (CBOR Sequence) (36 bytes)  
00 58 20 35 6e fd 53 77 14 25 e0 08 f3 fe 3a 86 c8 3f f4 c6 b1 6e 57  
02 8f f3 9d 52 36 c1 82 b2 02 08 4b 0b

KEYSTREAM\_2 (Raw Value) (11 bytes)  
bf 50 e9 e7 ba d0 bb 68 17 33 99

The Responder calculates CIPHERTEXT\_2 as XOR between PLAINTEXT\_2 and KEYSTREAM\_2:

CIPHERTEXT\_2 (Raw Value) (11 bytes)  
98 62 a1 1d e4 2a 95 d7 85 38 6a

The Responder constructs message\_2:

```
message_2 =  
(  
  G_Y_CIPHERTEXT_2,  
)
```

where G\_Y\_CIPHERTEXT\_2 is the bstr encoding of the concatenation of the raw values of G\_Y and CIPHERTEXT\_2.

message\_2 (CBOR Sequence) (45 bytes)

58 2b 41 97 01 d7 f0 0a 26 c2 dc 58 7a 36 dd 75 25 49 f3 37 63 c8 93  
42 2c 8e a0 f9 55 a1 3a 4f f5 d5 98 62 a1 1d e4 2a 95 d7 85 38 6a

### 3.5. message\_3

The transcript hash TH\_3 is calculated using the EDHOC hash algorithm:

TH\_3 = H( TH\_2, PLAINTEXT\_2, CRED\_R )

Input to calculate TH\_3 (CBOR Sequence) (140 bytes)

58 20 35 6e fd 53 77 14 25 e0 08 f3 fe 3a 86 c8 3f f4 c6 b1 6e 57 02  
8f f3 9d 52 36 c1 82 b2 02 08 4b 27 32 48 fa 5e fa 2e bf 92 0b f3 a2  
02 6b 65 78 61 6d 70 6c 65 2e 65 64 75 08 a1 01 a5 01 02 02 41 32 20  
01 21 58 20 bb c3 49 60 52 6e a4 d3 2e 94 0c ad 2a 23 41 48 dd c2 17  
91 a1 2a fb cb ac 93 62 20 46 dd 44 f0 22 58 20 45 19 e2 57 23 6b 2a  
0c e2 02 3f 09 31 f1 f3 86 ca 7a fd a6 4f cd e0 10 8c 22 4c 51 ea bf  
60 72

TH\_3 (Raw Value) (32 bytes)

df e5 b0 65 e6 4c 72 d2 26 d5 00 c1 2d 49 be e6 dc 48 81 de d0 96 5e  
9b df 89 d2 4a 54 f2 e5 9a

TH\_3 (CBOR Data Item) (34 bytes)

58 20 df e5 b0 65 e6 4c 72 d2 26 d5 00 c1 2d 49 be e6 dc 48 81 de d0  
96 5e 9b df 89 d2 4a 54 f2 e5 9a

Since METHOD = 3, the Initiator authenticates using static DH. The EDHOC key exchange algorithm is based on the same curve as for the ephemeral keys, which is P-256, since the selected cipher suite is 2.

The Initiator's static Diffie-Hellman P-256 key pair:

Initiator's private authentication key

SK\_I (Raw Value) (32 bytes)

fb 13 ad eb 65 18 ce e5 f8 84 17 66 08 41 14 2e 83 0a 81 fe 33 43 80  
a9 53 40 6a 13 05 e8 70 6b

Initiator's public authentication key, 'x'-coordinate

(Raw Value) (32 bytes)

ac 75 e9 ec e3 e5 0b fc 8e d6 03 99 88 95 22 40 5c 47 bf 16 df 96 66  
0a 41 29 8c b4 30 7f 7e b6

Initiator's public authentication key, 'y'-coordinate

(Raw Value) (32 bytes)

6e 5d e6 11 38 8a 4b 8a 82 11 33 4a c7 d3 7e cb 52 a3 87 d2 57 e6 db  
3c 2a 93 df 21 ff 3a ff c8

Since I authenticates with static DH (METHOD = 3), PRK\_4e3m is derived from SALT\_4e3m and G\_IY.

The input needed to calculate SALT\_4e3m is defined in [Section 4.1.2](#) of [[I-D.ietf-lake-edhoc](#)], using EDHOC\_Expand() with the EDHOC hash algorithm:

```
SALT_4e3m = EDHOC_KDF( PRK_3e2m, 5, TH_3, hash_length ) =  
            = HKDF-Expand( PRK_3e2m, info, hash_length )
```

where hash\_length is the length in bytes of the output of the EDHOC hash algorithm, and info for SALT\_4e3m is:

```
info =  
(  
  5,  
  h'dfe5b065e64c72d226d500c12d49bee6dc4881ded0965e9b  
    df89d24a54f2e59a',  
  32  
)
```

```
info for SALT_4e3m (CBOR Sequence) (37 bytes)  
05 58 20 df e5 b0 65 e6 4c 72 d2 26 d5 00 c1 2d 49 be e6 dc 48 81 de  
d0 96 5e 9b df 89 d2 4a 54 f2 e5 9a 18 20
```

```
SALT_4e3m (Raw Value) (32 bytes)  
84 f8 a2 a9 53 4d dd 78 dc c7 e7 6e 0d 4d f6 0b fa d7 cd 3a d6 e1 d5  
31 c7 f3 73 a7 ed a5 2d 1c
```

PRK\_4e3m is specified in [Section 4.1.1.3](#) of [[I-D.ietf-lake-edhoc](#)].

Since I authenticates with static DH (METHOD = 3), PRK\_4e3m is derived from G\_IY using EDHOC\_Extract() with the EDHOC hash algorithm:

```
PRK_4e3m = EDHOC_Extract(SALT_4e3m, G_IY) =  
            = HMAC-SHA-256(SALT_4e3m, G_IY)
```

where G\_IY is the ECDH shared secret calculated from G\_I and Y, or G\_Y and I.

```
G_IY (Raw Value) (ECDH shared secret) (32 bytes)  
08 0f 42 50 85 bc 62 49 08 9e ac 8f 10 8e a6 23 26 85 7e 12 ab 07 d7  
20 28 ca 1b 5f 36 e0 04 b3
```

```
PRK_4e3m (Raw Value) (32 bytes)  
e9 cb 83 2a 24 00 95 d3 d0 64 3d be 12 e9 e2 e7 b1 8f 03 60 a3 17 2c  
ea 7a c0 01 3e e2 40 e0 72
```

The Initiator constructs the remaining input needed to calculate MAC\_3:

```
MAC_3 = EDHOC_KDF( PRK_4e3m, 6, context_3, mac_length_3 )
```

```
context_3 = << ID_CRED_I, TH_3, CRED_I, ? EAD_3 >>
```

CRED\_I is identified by a 'kid' with byte string value 0x2b:

```
ID_CRED_I =  
{  
  4 : h'2b'  
}
```

ID\_CRED\_I (CBOR Data Item) (4 bytes)  
a1 04 41 2b

CRED\_I is an RPK encoded as a CCS:

```
{  
  2 : "42-50-31-FF-EF-37-32-39", /sub/  
  8 : { /cnf/  
    1 : { /COSE_Key/  
      1 : 2, /kty/  
      2 : h'2b', /kid/  
      -1 : 1, /crv/  
      -2 : h'AC75E9ECE3E50BFC8ED6039988952240  
           5C47BF16DF96660A41298CB4307F7EB6' /x/  
      -3 : h'6E5DE611388A4B8A8211334AC7D37ECB  
           52A387D257E6DB3C2A93DF21FF3AFFC8' /y/  
    }  
  }  
}
```

CRED\_I (CBOR Data Item) (107 bytes)  
a2 02 77 34 32 2d 35 30 2d 33 31 2d 46 46 2d 45 46 2d 33 37 2d 33 32  
2d 33 39 08 a1 01 a5 01 02 02 41 2b 20 01 21 58 20 ac 75 e9 ec e3 e5  
0b fc 8e d6 03 99 88 95 22 40 5c 47 bf 16 df 96 66 0a 41 29 8c b4 30  
7f 7e b6 22 58 20 6e 5d e6 11 38 8a 4b 8a 82 11 33 4a c7 d3 7e cb 52  
a3 87 d2 57 e6 db 3c 2a 93 df 21 ff 3a ff c8

No external authorization data:

EAD\_3 (CBOR Sequence) (0 bytes)

```
context_3 = << ID_CRED_I, TH_3, CRED_I, ? EAD_3 >>
```

```
context_3 (CBOR Sequence) (145 bytes)
a1 04 41 2b 58 20 df e5 b0 65 e6 4c 72 d2 26 d5 00 c1 2d 49 be e6 dc
48 81 de d0 96 5e 9b df 89 d2 4a 54 f2 e5 9a a2 02 77 34 32 2d 35 30
2d 33 31 2d 46 46 2d 45 46 2d 33 37 2d 33 32 2d 33 39 08 a1 01 a5 01
02 02 41 2b 20 01 21 58 20 ac 75 e9 ec e3 e5 0b fc 8e d6 03 99 88 95
22 40 5c 47 bf 16 df 96 66 0a 41 29 8c b4 30 7f 7e b6 22 58 20 6e 5d
e6 11 38 8a 4b 8a 82 11 33 4a c7 d3 7e cb 52 a3 87 d2 57 e6 db 3c 2a
93 df 21 ff 3a ff c8
```

```
context_3 (CBOR byte string) (147 bytes)
58 91 a1 04 41 2b 58 20 df e5 b0 65 e6 4c 72 d2 26 d5 00 c1 2d 49 be
e6 dc 48 81 de d0 96 5e 9b df 89 d2 4a 54 f2 e5 9a a2 02 77 34 32 2d
35 30 2d 33 31 2d 46 46 2d 45 46 2d 33 37 2d 33 32 2d 33 39 08 a1 01
a5 01 02 02 41 2b 20 01 21 58 20 ac 75 e9 ec e3 e5 0b fc 8e d6 03 99
88 95 22 40 5c 47 bf 16 df 96 66 0a 41 29 8c b4 30 7f 7e b6 22 58 20
6e 5d e6 11 38 8a 4b 8a 82 11 33 4a c7 d3 7e cb 52 a3 87 d2 57 e6 db
3c 2a 93 df 21 ff 3a ff c8
```

MAC\_3 is computed through EDHOC\_Expand() using the EDHOC hash algorithm, see [Section 4.1.2](#) of [[I-D.ietf-lake-edhoc](#)]:

MAC\_3 = HKDF-Expand(PRK\_4e3m, info, mac\_length\_3), where

info = ( 6, context\_3, mac\_length\_3 )

Since METHOD = 3, mac\_length\_3 is given by the EDHOC MAC length.

info for MAC\_3 is:

```
info =
(
  6,
  h'a104412b5820dfe5b065e64c72d226d500c12d49bee6dc48
  81ded0965e9bdf89d24a54f2e59aa2027734322d35302d33
  312d46462d45462d33372d33322d333908a101a501020241
  2b2001215820ac75e9ece3e50bfc8ed60399889522405c47
  bf16df96660a41298cb4307f7eb62258206e5de611388a4b
  8a8211334ac7d37ecb52a387d257e6db3c2a93df21ff3aff
  c8',
  8
)
```

where the last value is the EDHOC MAC length in bytes.

info for MAC\_3 (CBOR Sequence) (149 bytes)

```
06 58 91 a1 04 41 2b 58 20 df e5 b0 65 e6 4c 72 d2 26 d5 00 c1 2d 49
be e6 dc 48 81 de d0 96 5e 9b df 89 d2 4a 54 f2 e5 9a a2 02 77 34 32
2d 35 30 2d 33 31 2d 46 46 2d 45 46 2d 33 37 2d 33 32 2d 33 39 08 a1
01 a5 01 02 02 41 2b 20 01 21 58 20 ac 75 e9 ec e3 e5 0b fc 8e d6 03
99 88 95 22 40 5c 47 bf 16 df 96 66 0a 41 29 8c b4 30 7f 7e b6 22 58
20 6e 5d e6 11 38 8a 4b 8a 82 11 33 4a c7 d3 7e cb 52 a3 87 d2 57 e6
db 3c 2a 93 df 21 ff 3a ff c8 08
```

MAC\_3 (Raw Value) (8 bytes)

```
a5 ee b9 ef fd ab fc 39
```

MAC\_3 (CBOR Data Item) (9 bytes)

```
48 a5 ee b9 ef fd ab fc 39
```

Since METHOD = 3, Signature\_or\_MAC\_3 is MAC\_3:

Signature\_or\_MAC\_3 (Raw Value) (8 bytes)

```
a5 ee b9 ef fd ab fc 39
```

Signature\_or\_MAC\_3 (CBOR Data Item) (9 bytes)

```
48 a5 ee b9 ef fd ab fc 39
```

The Initiator constructs PLAINTEXT\_3:

PLAINTEXT\_3 =

```
(
  ID_CRED_I / bstr / -24..23,
  Signature_or_MAC_3,
  ? EAD_3
)
```

Since ID\_CRED\_I contains a single 'kid' parameter, only the byte string value is included in the plaintext, represented as described in [Section 3.3.2](#) of [\[I-D.ietf-lake-edhoc\]](#). The CBOR map { 4 : h'2b' } is thus replaced, not by the CBOR byte string 0x412b, but by the CBOR int 0x2b, since that is a one byte encoding of a CBOR integer (-12).

PLAINTEXT\_3 (CBOR Sequence) (10 bytes)

```
2b 48 a5 ee b9 ef fd ab fc 39
```

The Initiator constructs the associated data for message\_3:

```
A_3 =
[
  "Encrypt0",
  h'',
  h'dfe5b065e64c72d226d500c12d49bee6dc4881ded0965e9b
  df89d24a54f2e59a'
]
```

A\_3 (CBOR Data Item) (45 bytes)

```
83 68 45 6e 63 72 79 70 74 30 40 58 20 df e5 b0 65 e6 4c 72 d2 26 d5
00 c1 2d 49 be e6 dc 48 81 de d0 96 5e 9b df 89 d2 4a 54 f2 e5 9a
```

The Initiator constructs the input needed to derive the key K\_3, see [Section 4.1.2](#) of [[I-D.ietf-lake-edhoc](#)], using the EDHOC hash algorithm:

```
K_3 = EDHOC_KDF( PRK_3e2m, 3, TH_3, key_length )
      = HKDF-Expand( PRK_3e2m, info, key_length ),
```

where key\_length is the key length in bytes for the EDHOC AEAD algorithm, and info for K\_3 is:

```
info =
(
  3,
  h'dfe5b065e64c72d226d500c12d49bee6dc4881ded0965e9b
  df89d24a54f2e59a',
  16
)
```

where the last value is the key length in bytes for the EDHOC AEAD algorithm.

info for K\_3 (CBOR Sequence) (36 bytes)

```
03 58 20 df e5 b0 65 e6 4c 72 d2 26 d5 00 c1 2d 49 be e6 dc 48 81 de
d0 96 5e 9b df 89 d2 4a 54 f2 e5 9a 10
```

K\_3 (Raw Value) (16 bytes)

```
ab 3b 2b 52 a0 4b 6a a3 2f 96 31 19 16 88 3a dd
```

The Initiator constructs the input needed to derive the nonce IV\_3, see [Section 4.1.2](#) of [[I-D.ietf-lake-edhoc](#)], using the EDHOC hash algorithm:

```
IV_3 = EDHOC_KDF( PRK_3e2m, 4, TH_3, iv_length )
      = HKDF-Expand( PRK_3e2m, info, iv_length ),
```

where iv\_length is the nonce length in bytes for the EDHOC AEAD algorithm, and info for IV\_3 is:

```
info =
(
  4,
  h'dfe5b065e64c72d226d500c12d49bee6dc4881ded0965e9b
    df89d24a54f2e59a',
  13
)
```

where the last value is the nonce length in bytes for the EDHOC AEAD algorithm.

```
info for IV_3 (CBOR Sequence) (36 bytes)
04 58 20 df e5 b0 65 e6 4c 72 d2 26 d5 00 c1 2d 49 be e6 dc 48 81 de
d0 96 5e 9b df 89 d2 4a 54 f2 e5 9a 0d
```

```
IV_3 (Raw Value) (13 bytes)
05 55 cf a1 6e 40 8d e5 e1 52 3d 04 7d
```

The Initiator calculates CIPHERTEXT\_3 as 'ciphertext' of COSE\_Encrypt0 applied using the EDHOC AEAD algorithm with plaintext PLAINTEXT\_3, additional data A\_3, key K\_3 and nonce IV\_3.

```
CIPHERTEXT_3 (Raw Value) (18 bytes)
47 3d d1 60 77 dd 71 d6 5b 56 e6 bd 71 e7 a4 9d 60 12
```

message\_3 is the CBOR bstr encoding of CIPHERTEXT\_3:

```
message_3 (CBOR Sequence) (19 bytes)
52 47 3d d1 60 77 dd 71 d6 5b 56 e6 bd 71 e7 a4 9d 60 12
```

The transcript hash TH\_4 is calculated using the EDHOC hash algorithm:

TH\_4 = H( TH\_3, PLAINTEXT\_3, CRED\_I )

```
Input to calculate TH_4 (CBOR Sequence) (151 bytes)
58 20 df e5 b0 65 e6 4c 72 d2 26 d5 00 c1 2d 49 be e6 dc 48 81 de d0
96 5e 9b df 89 d2 4a 54 f2 e5 9a 2b 48 a5 ee b9 ef fd ab fc 39 a2 02
77 34 32 2d 35 30 2d 33 31 2d 46 46 2d 45 46 2d 33 37 2d 33 32 2d 33
39 08 a1 01 a5 01 02 02 41 2b 20 01 21 58 20 ac 75 e9 ec e3 e5 0b fc
8e d6 03 99 88 95 22 40 5c 47 bf 16 df 96 66 0a 41 29 8c b4 30 7f 7e
b6 22 58 20 6e 5d e6 11 38 8a 4b 8a 82 11 33 4a c7 d3 7e cb 52 a3 87
d2 57 e6 db 3c 2a 93 df 21 ff 3a ff c8
```

```
TH_4 (Raw Value) (32 bytes)
ba f6 0a db c5 00 fc e7 89 af 25 b1 08 ad a2 27 55 75 05 6c 52 c1 c2
03 6a 2d a4 a6 43 89 1c b4
```



TH\_4 (CBOR Data Item) (34 bytes)  
58 20 ba f6 0a db c5 00 fc e7 89 af 25 b1 08 ad a2 27 55 75 05 6c 52  
c1 c2 03 6a 2d a4 a6 43 89 1c b4

### 3.6. message\_4

No external authorization data:

EAD\_4 (CBOR Sequence) (0 bytes)

The Responder constructs PLAINTEXT\_4:

PLAINTEXT\_4 =

```
(  
  ? EAD_4  
)
```

PLAINTEXT\_4 (CBOR Sequence) (0 bytes)

The Responder constructs the associated data for message\_4:

A\_4 =

```
[  
  "Encrypt0",  
  h'',  
  h'ba6f60adbc500fce789af25b108ada2275575056c52c1c203  
    6a2da4a643891cb4'  
]
```

A\_4 (CBOR Data Item) (45 bytes)

83 68 45 6e 63 72 79 70 74 30 40 58 20 ba f6 0a db c5 00 fc e7 89 af  
25 b1 08 ad a2 27 55 75 05 6c 52 c1 c2 03 6a 2d a4 a6 43 89 1c b4

The Responder constructs the input needed to derive the EDHOC message\_4 key, see [Section 4.1.2](#) of [[I-D.ietf-lake-edhoc](#)], using the EDHOC hash algorithm:

```
K_4 = EDHOC_KDF( PRK_4e3m, 8, TH_4, key_length )  
      = HKDF-Expand( PRK_4e3m, info, key_length )
```

where key\_length is the key length in bytes for the EDHOC AEAD algorithm, and info for K\_4 is:

info =

```
(  
  8,  
  h'ba6f60adbc500fce789af25b108ada2275575056c52c1c203  
    6a2da4a643891cb4',  
  16  
)
```

where the last value is the key length in bytes for the EDHOC AEAD algorithm.

info for K\_4 (CBOR Sequence) (36 bytes)

```
08 58 20 ba f6 0a db c5 00 fc e7 89 af 25 b1 08 ad a2 27 55 75 05 6c
52 c1 c2 03 6a 2d a4 a6 43 89 1c b4 10
```

K\_4 (Raw Value) (16 bytes)

```
22 9d 4c 1d 6d 02 33 7b 1c e3 81 a2 bf a7 9b 2e
```

The Responder constructs the input needed to derive the EDHOC message\_4 nonce, see [Section 4.1.2](#) of [[I-D.ietf-lake-edhoc](#)], using the EDHOC hash algorithm:

```
IV_4 = EDHOC_KDF( PRK_4e3m, 9, TH_4, iv_length )
      = HKDF-Expand( PRK_4e3m, info, iv_length )
```

where iv\_length is the nonce length in bytes for the EDHOC AEAD algorithm, and info for IV\_4 is:

```
info =
(
  9,
  h'ba60adbc500fce789af25b108ada2275575056c52c1c203
  6a2da4a643891cb4',
  13
)
```

where the last value is the nonce length in bytes for the EDHOC AEAD algorithm.

info for IV\_4 (CBOR Sequence) (36 bytes)

```
09 58 20 ba f6 0a db c5 00 fc e7 89 af 25 b1 08 ad a2 27 55 75 05 6c
52 c1 c2 03 6a 2d a4 a6 43 89 1c b4 0d
```

IV\_4 (Raw Value) (13 bytes)

```
98 4d 59 ab 25 5e 3d c6 f8 e0 65 5c b6
```

The Responder calculates CIPHERTEXT\_4 as 'ciphertext' of COSE\_Encrypt0 applied using the EDHOC AEAD algorithm with plaintext PLAINTEXT\_4, additional data A\_4, key K\_4 and nonce IV\_4.

CIPHERTEXT\_4 (8 bytes)

```
89 07 43 64 70 a6 e1 9f
```

message\_4 is the CBOR bstr encoding of CIPHERTEXT\_4:

message\_4 (CBOR Sequence) (9 bytes)

```
48 89 07 43 64 70 a6 e1 9f
```

### 3.7. PRK\_out and PRK\_exporter

PRK\_out is specified in [Section 4.1.3](#) of [\[I-D.ietf-lake-edhoc\]](#).

```
PRK_out = EDHOC_KDF( PRK_4e3m, 7, TH_4, hash_length ) =  
    = HKDF-Expand( PRK_4e3m, info, hash_length )
```

where hash\_length is the length in bytes of the output of the EDHOC hash algorithm, and info for PRK\_out is:

```
info =  
(  
  7,  
  h'baf60adbc500fce789af25b108ada2275575056c52c1c203  
    6a2da4a643891cb4',  
  32  
)
```

where the last value is the length in bytes of the output of the EDHOC hash algorithm.

```
info for PRK_out (CBOR Sequence) (37 bytes)  
07 58 20 ba f6 0a db c5 00 fc e7 89 af 25 b1 08 ad a2 27 55 75 05 6c  
52 c1 c2 03 6a 2d a4 a6 43 89 1c b4 18 20
```

```
PRK_out (Raw Value) (32 bytes)  
6b 2d ae 40 32 30 65 71 cf bc 2e 4f 94 a2 55 fb 9f 1f 3f b2 9c a6 f3  
79 fe c9 89 d4 fa 90 dc f0
```

The OSCORE Master Secret and OSCORE Master Salt are derived with the EDHOC\_Exporter as specified in 4.2.1 of [\[I-D.ietf-lake-edhoc\]](#).

```
EDHOC_Exporter( label, context, length )  
= EDHOC_KDF( PRK_exporter, label, context, length )
```

where PRK\_exporter is derived from PRK\_out:

```
PRK_exporter = EDHOC_KDF( PRK_out, 10, h'', hash_length ) =  
    = HKDF-Expand( PRK_out, info, hash_length )
```

where hash\_length is the length in bytes of the output of the EDHOC hash algorithm, and info for the PRK\_exporter is:

```
info =  
(  
  10,  
  h'',  
  32  
)
```

where the last value is the length in bytes of the output of the EDHOC hash algorithm.

info for PRK\_exporter (CBOR Sequence) (4 bytes)

0a 40 18 20

PRK\_exporter (Raw Value) (32 bytes)

4f 0a 5a 82 3d 06 d0 00 5e 1b ec da 8a 6e 61 f3 c8 c6 7a 8b 15 da 7d  
44 d3 58 5e c5 85 4e 91 e2

### 3.8. OSCORE Parameters

The derivation of OSCORE parameters is specified in [Appendix A.1](#) of [\[I-D.ietf-lake-edhoc\]](#).

The AEAD and Hash algorithms to use in OSCORE are given by the selected cipher suite:

Application AEAD Algorithm (int)

10

Application Hash Algorithm (int)

-16

The mapping from EDHOC connection identifiers to OSCORE Sender/Recipient IDs is defined in [Section 3.3.3](#) of [\[I-D.ietf-lake-edhoc\]](#).

C\_R is mapped to the Recipient ID of the server, i.e., the Sender ID of the client. The byte string 0x27, which as C\_R is encoded as the CBOR integer 0x27, is converted to the server Recipient ID 0x27.

Client's OSCORE Sender ID (Raw Value) (1 byte)

27

C\_I is mapped to the Recipient ID of the client, i.e., the Sender ID of the server. The byte string 0x37, which as C\_I is encoded as the CBOR integer 0x0e is converted to the client Recipient ID 0x37.

Server's OSCORE Sender ID (Raw Value) (1 byte)

37

The OSCORE Master Secret is computed through EDHOC\_Expand() using the Application hash algorithm, see [Appendix A.1](#) of [\[I-D.ietf-lake-edhoc\]](#):

```
OSCORE Master Secret = EDHOC_Exporter( 0, h'', oscore_key_length )  
= EDHOC_KDF( PRK_exporter, 0, h'', oscore_key_length )  
= HKDF-Expand( PRK_exporter, info, oscore_key_length )
```

where `oscore_key_length` is by default the key length in bytes for the Application AEAD algorithm, and `info` for the OSCORE Master Secret is:

```
info =  
(  
  0,  
  h'',  
  16  
)
```

where the last value is the key length in bytes for the Application AEAD algorithm.

info for OSCORE Master Secret (CBOR Sequence) (3 bytes)  
00 40 10

OSCORE Master Secret (Raw Value) (16 bytes)  
8c 40 9a 33 22 23 ad 90 0e 44 f3 43 4d 2d 2c e3

The OSCORE Master Salt is computed through `EDHOC_Expand()` using the Application hash algorithm, see [Section 4.2](#) of [\[I-D.ietf-lake-edhoc\]](#):

```
OSCORE Master Salt = EDHOC_Exporter( 1, h'', oscore_salt_length )  
= EDHOC_KDF( PRK_exporter, 1, h'', oscore_salt_length )  
= HKDF-Expand( PRK_4x3m, info, oscore_salt_length )
```

where `oscore_salt_length` is the length in bytes of the OSCORE Master Salt, and `info` for the OSCORE Master Salt is:

```
info =  
(  
  1,  
  h'',  
  8  
)
```

where the last value is the length in bytes of the OSCORE Master Salt.

info for OSCORE Master Salt (CBOR Sequence) (3 bytes)  
01 40 08

OSCORE Master Salt (Raw Value) (8 bytes)  
61 63 f4 4b e8 62 ad fa

### 3.9. Key Update

Key update is defined in [Appendix H](#) of [\[I-D.ietf-lake-edhoc\]](#).

```
EDHOC_KeyUpdate( context ):
PRK_out = EDHOC_KDF( PRK_out, 11, context, hash_length )
           = HKDF-Expand( PRK_out, info, hash_length )
```

where hash\_length is the length in bytes of the output of the EDHOC hash function, context for KeyUpdate is

```
context for KeyUpdate (Raw Value) (16 bytes)
a0 11 58 fd b8 20 89 0c d6 be 16 96 02 b8 bc ea
```

```
context for KeyUpdate (CBOR Data Item) (17 bytes)
50 a0 11 58 fd b8 20 89 0c d6 be 16 96 02 b8 bc ea
```

and where info for key update is:

```
info =
(
  11,
  h'a01158fdb820890cd6be169602b8bcea',
  32
)
```

```
PRK_out after KeyUpdate (Raw Value) (32 bytes)
5e 5e fc ae dd a8 d1 85 bb 7e 26 1d f1 91 59 1c d9 f7 c9 20 49 e7 0c
23 f6 b4 34 e3 6d fc 1d 1c
```

After key update the PRK\_exporter needs to be derived anew:

```
PRK_exporter = EDHOC_KDF( PRK_out, 10, h'', hash_length ) =
               = HKDF-Expand( PRK_out, info, hash_length )
```

where info and hash\_length are unchanged as in [Section 3.7](#).

```
PRK_exporter (Raw Value) (32 bytes)
bb b3 b7 72 6e 97 9c 1b b3 46 a3 f9 2b f4 e0 28 8d 52 62 7f b5 e7 9a
fd b3 b2 82 02 fd 2e 48 97
```

The OSCORE Master Secret is derived with the updated PRK\_exporter:

```
OSCORE Master Secret =
= HKDF-Expand(PRK_exporter, info, oscore_key_length)
```

where info and key\_length are unchanged as in [Section 2.6](#).

```
OSCORE Master Secret after KeyUpdate (Raw Value) (16 bytes)
c9 1b 16 4c 81 0b 29 a6 3f cb 73 e5 1b c4 55 f3
```

The OSCORE Master Salt is derived with the updated PRK\_exporter:

```
OSCORE Master Salt = HKDF-Expand(PRK_exporter, info, salt_length)
```

where info and salt\_length are unchanged as in [Section 2.6](#).

OSCORE Master Salt after KeyUpdate (Raw Value) (8 bytes)  
73 ce 79 24 59 40 36 80

#### 4. Invalid Traces

This section contains examples of invalid messages, which a compliant implementation will not compose and must or may reject according to [[I-D.ietf-lake-edhoc](#)], [[RFC8949](#)], [[RFC9053](#)], and [[SP-800-56A](#)]. This is just a small set of examples of different reasons a message might be invalid. The same types of invalidities applies to other fields and messages as well. Implementations should make sure to check for similar types of invalidities in all EHDOC fields and messages.

##### 4.1. Encoding Errors

###### 4.1.1. Surplus array encoding of message

Invalid encoding of message\_1 as array. Correct encoding is a CBOR sequence according to Section 5.2.1 of [[I-D.ietf-lake-edhoc](#)].

Invalid message\_1 (38 bytes)  
84 03 02 58 20 74 1a 13 d7 ba 04 8f bb 61 5e 94 38 6a a3 b6 1b ea 5b  
3d 8f 65 f3 26 20 b7 49 be e8 d2 78 ef a9 0e

###### 4.1.2. Surplus bstr encoding of connection identifier

Invalid encoding 41 0e of C\_I = 0x0e. Correct encoding is 0e according to Section 3.3.2 of [[I-D.ietf-lake-edhoc](#)].

Invalid message\_1 (38 bytes)  
03 02 58 20 74 1a 13 d7 ba 04 8f bb 61 5e 94 38 6a a3 b6 1b ea 5b 3d  
8f 65 f3 26 20 b7 49 be e8 d2 78 ef a9 41 0e

###### 4.1.3. Surplus array encoding of ciphersuite

Invalid array encoding 81 02 of SUITES\_I = 2. Correct encoding is 02 according to Section 5.2.2 of [[I-D.ietf-lake-edhoc](#)].

Invalid message\_1 (38 bytes)  
03 81 02 58 20 74 1a 13 d7 ba 04 8f bb 61 5e 94 38 6a a3 b6 1b ea 5b  
3d 8f 65 f3 26 20 b7 49 be e8 d2 78 ef a9 0e

###### 4.1.4. Text string encoding of ephemeral key

Invalid type of the third element (G\_X). Correct encoding is a byte string according to Section 5.2.1 of [[I-D.ietf-lake-edhoc](#)].

Invalid message\_1 (37 bytes)

```
03 02 78 20 20 61 69 72 20 73 70 65 65 64 20 6F 66 20 61 20 75 6E 6C
61 64 65 6E 20 73 77 61 6C 6C 6F 77 20 0e
```

#### 4.1.5. Wrong number of CBOR sequence elements

Invalid number of elements in the CBOR sequence. Correct number of elements is 1 according to Section 5.3.1 of [[I-D.ietf-lake-edhoc](#)].

Invalid message\_2 (46 bytes)

```
58 20 41 97 01 d7 f0 0a 26 c2 dc 58 7a 36 dd 75 25 49 f3 37 63 c8 93
42 2c 8e a0 f9 55 a1 3a 4f f5 d5 4B 98 62 a1 1d e4 2a 95 d7 85 38 6a
```

#### 4.1.6. Surplus map encoding of ID\_CRED field

Invalid encoding a1 04 42 32 10 of ID\_CRED\_R in PLAINTEXT\_2. Correct encoding is 42 32 10 according to Section 3.5.3.2 of [[I-D.ietf-lake-edhoc](#)].

Invalid PLAINTEXT\_2 (15 bytes)

```
27 a1 04 42 32 10 48 fa 5e fa 2e bf 92 0b f3
```

#### 4.1.7. Surplus bstr encoding of ID\_CRED field

Invalid encoding 41 32 of ID\_CRED\_R in PLAINTEXT\_2. Correct encoding is 32 according to Section 3.5.3.2 of [[I-D.ietf-lake-edhoc](#)].

Invalid PLAINTEXT\_2 (12 bytes)

```
27 41 32 48 fa 5e fa 2e bf 92 0b f3
```

## 4.2. Crypto-related Errors

### 4.2.1. Error in length of ephemeral key

Invalid length of the third element (G\_X). Selected cipher suite is cipher suite 24 with curve P-384 according to Sections 5.2.2, and 10.2 of [[I-D.ietf-lake-edhoc](#)]. Correct length of x-coordinate is 48 bytes according to Section 3.7 of [[I-D.ietf-lake-edhoc](#)] and Section 7.1.1 of [[RFC9053](#)].

Invalid message\_1 (40 bytes)

```
03 82 02 18 18 58 20 74 1a 13 d7 ba 04 8f bb 61 5e 94 38 6a a3 b6 1b
ea 5b 3d 8f 65 f3 26 20 b7 49 be e8 d2 78 ef a9 0e
```

### 4.2.2. Error in elliptic curve representation

Invalid x-coordinate in G\_X as  $x \geq p$ . Requirement that  $x < p$  according to Section 9.2 of [[I-D.ietf-lake-edhoc](#)] and Section 5.6.2.3 of [[SP-800-56A](#)].





[[I-D.ietf-lake-edhoc](#)] and Section 4.2.1 of [[RFC8949](#)], which states that the arguments for integers, lengths in major types 2 through 5, and tags are required to be as short as possible.

Invalid message\_1 (39 bytes)

```
19 00 03 02 58 20 74 1a 13 d7 ba 04 8f bb 61 5e 94 38 6a a3 b6 1b ea
5b 3d 8f 65 f3 26 20 b7 49 be e8 d2 78 ef a9 0e
```

#### 4.3.2. Indefinite-length array encoding

Invalid indefinite-length array encoding 9F 06 02 FF of SUITES\_I = [6, 2]. Correct encoding is 82 06 02 according to Section 5.2.2 of [[I-D.ietf-lake-edhoc](#)].

Invalid message\_1 (40 bytes)

```
03 9F 06 02 FF 58 20 74 1a 13 d7 ba 04 8f bb 61 5e 94 38 6a a3 b6 1b
ea 5b 3d 8f 65 f3 26 20 b7 49 be e8 d2 78 ef a9 0e
```

## 5. Security Considerations

This document contains examples of EDHOC [[I-D.ietf-lake-edhoc](#)] whose security considerations apply. The keys printed in these examples cannot be considered secret and MUST NOT be used.

## 6. IANA Considerations

There are no IANA considerations.

## 7. References

### 7.1. Normative References

[[I-D.ietf-lake-edhoc](#)] Selander, G., Mattsson, J. P., and F. Palombini, "Ephemeral Diffie-Hellman Over COSE (EDHOC)", Work in Progress, Internet-Draft, draft-ietf-lake-edhoc-22, 25 August 2023, <<https://datatracker.ietf.org/doc/html/draft-ietf-lake-edhoc-22>>.

[[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/rfc/rfc2119>>.

[[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/rfc/rfc8174>>.

### 7.2. Informative References

[[CborMe](#)]

Bormann, C., "CBOR playground", August 2023, <<https://cbor.me/>>.

- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<https://www.rfc-editor.org/rfc/rfc7252>>.
- [RFC7748] Langley, A., Hamburg, M., and S. Turner, "Elliptic Curves for Security", RFC 7748, DOI 10.17487/RFC7748, January 2016, <<https://www.rfc-editor.org/rfc/rfc7748>>.
- [RFC8032] Josefsson, S. and I. Liusvaara, "Edwards-Curve Digital Signature Algorithm (EdDSA)", RFC 8032, DOI 10.17487/RFC8032, January 2017, <<https://www.rfc-editor.org/rfc/rfc8032>>.
- [RFC8392] Jones, M., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "CBOR Web Token (CWT)", RFC 8392, DOI 10.17487/RFC8392, May 2018, <<https://www.rfc-editor.org/rfc/rfc8392>>.
- [RFC8949] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", STD 94, RFC 8949, DOI 10.17487/RFC8949, December 2020, <<https://www.rfc-editor.org/rfc/rfc8949>>.
- [RFC9053] Schaad, J., "CBOR Object Signing and Encryption (COSE): Initial Algorithms", RFC 9053, DOI 10.17487/RFC9053, August 2022, <<https://www.rfc-editor.org/rfc/rfc9053>>.
- [SP-800-186] Chen, L., Moody, D., Randall, K., Regenscheid, A., and A. Robinson, "Recommendations for Discrete Logarithm-based Cryptography: Elliptic Curve Domain Parameters", NIST Special Publication 800-186, February 2023, <<https://doi.org/10.6028/NIST.SP.800-186>>.
- [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>>.

## Acknowledgments

The authors want to thank all people verifying EDHOC test vectors and/or contributing to the interoperability testing including: Christian Amsüss, Timothy Claeys, Stefan Hristozov, Rikard Höglund, Christos Koulamas, Francesca Palombini, Lidia Pocero, Peter van der Stok, and Michel Veillette.

## Authors' Addresses

Göran Selander  
Ericsson  
Sweden

Email: [goran.selander@ericsson.com](mailto:goran.selander@ericsson.com)

John Preuß Mattsson  
Ericsson  
Sweden

Email: [john.mattsson@ericsson.com](mailto:john.mattsson@ericsson.com)

Marek Serafin  
ASSA ABLOY  
Poland

Email: [marek.serafin@assaabloy.com](mailto:marek.serafin@assaabloy.com)

Marco Tiloca  
RISE  
Sweden

Email: [marco.tiloca@ri.se](mailto:marco.tiloca@ri.se)

Mališa Vučinić  
Inria  
France

Email: [malisa.vucinic@inria.fr](mailto:malisa.vucinic@inria.fr)