

Workgroup: LAKE Working Group  
Internet-Draft: draft-ietf-lake-traces-03  
Published: 24 October 2022  
Intended Status: Informational  
Expires: 27 April 2023

A G. Selander J. Preuß Mattsson M. Serafin  
uEricsson Ericsson ASSA ABLOY  
t  
h  
o  
r  
s  
:  
M. Tiloca  
RISE

## 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 27 April 2023.

### Copyright Notice

Copyright (c) 2022 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](#)
- [2. Setup](#)
- [3. Authentication with signatures, X.509 certificates identified by 'x5t'](#)
  - [3.1. message\\_1](#)
  - [3.2. message\\_2](#)
  - [3.3. message\\_3](#)
  - [3.4. message\\_4](#)
  - [3.5. PRK\\_out and PRK\\_exporter](#)
  - [3.6. OSCORE Parameters](#)
  - [3.7. Key Update](#)
  - [3.8. Certificates](#)
- [4. Authentication with static DH, CCS identified by 'kid'](#)
  - [4.1. message\\_1 \(first time\)](#)
  - [4.2. error](#)
  - [4.3. message\\_1 \(second time\)](#)
  - [4.4. message\\_2](#)
  - [4.5. message\\_3](#)
  - [4.6. message\\_4](#)
  - [4.7. PRK\\_out and PRK\\_exporter](#)
  - [4.8. OSCORE Parameters](#)
  - [4.9. Key Update](#)
- [5. Security Considerations](#)
- [6. IANA Considerations](#)
- [7. 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 protocol runs, with input, output and intermediate processing results to simplify testing of implementations.

The document contains two traces:

\*[Section 3](#) - Authentication with signature keys identified by the hash value of the X.509 certificates (provided in [Section 3.8](#)). The endpoints use EdDSA [[RFC8032](#)] for authentication and X25519 [[RFC7748](#)] for ephemeral-ephemeral Diffie-Hellman key exchange.

\*[Section 4](#) - Authentication with static Diffie-Hellman keys identified by short key identifiers labelling CWT Claim Sets (CCSs) [[RFC8392](#)]. The endpoints use NIST P-256 (FIPS PUB 186-4) 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.

The traces in this draft are valid for version -16 and -17 of [[I-D.ietf-lake-edhoc](#)].

Editor's note: Update reference to test vectors below.

Test vectors for trace 2 can be found at <https://github.com/lake-wg/edhoc/tree/master/test-vectors-16/> )

## 2. Setup

EDHOC is run between an Initiator (I) and a Responder (R). The private/public key pairs and credentials of I and R 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.

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, 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.

## 3. 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 I and R 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'.

### 3.1. message\_1

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

METHOD (CBOR Data Item) (1 byte)

00

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

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

00

I 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

I 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:

C\_I (Raw Value) (Connection identifier chosen by I) (1 byte)

2d

C\_I (CBOR Data Item) (Connection identifier chosen by I) (1 byte)

2d

No external authorization data:

EAD\_1 (CBOR Sequence) (0 bytes)

I 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

### 3.2. message\_2

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

R 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

R 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:

C\_R (Raw Value) (Connection identifier chosen by R) (1 byte)

18

C\_R (CBOR Data Item) (Connection identifier chosen by R) (2 bytes)

41 18

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

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

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, C\_R, H(message\_1)

Input to calculate TH\_2 (CBOR Sequence) (70 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 41 18 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)

3a b1 17 00 84 1f ce 19 3c 32 39 11 ed b3 17 b0 46 dc f2 4b 99 50 fd  
62 48 84 f7 f5 7c d9 8b 07

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

58 20 3a b1 17 00 84 1f ce 19 3c 32 39 11 ed b3 17 b0 46 dc f2 4b 99  
50 fd 62 48 84 f7 f5 7c d9 8b 07

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 Extract() determined by the EDHOC hash algorithm:

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

where salt is TH\_2:

```
salt (Raw Value) (32 bytes)  
3a b1 17 00 84 1f ce 19 3c 32 39 11 ed b3 17 b0 46 dc f2 4b 99 50 fd  
62 48 84 f7 f5 7c d9 8b 07  
PRK_2e (Raw Value) (32 bytes)  
2a e2 42 1d e9 a7 2a 7a e6 71 5f b5 18 f3 ed 30 05 8f d9 ca 58 b6 25  
68 ca fe 7c da a1 5a 41 f7
```

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

R'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  
Responders'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 R authenticates with signatures PRK\_3e2m = PRK\_2e.

```
PRK_3e2m (Raw Value) (32 bytes)  
2a e2 42 1d e9 a7 2a 7a e6 71 5f b5 18 f3 ed 30 05 8f d9 ca 58 b6 25  
68 ca fe 7c da a1 5a 41 f7
```

R 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 3.8.1](#):

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 3a b1 17 00 84 1f ce  
19 3c 32 39 11 ed b3 17 b0 46 dc f2 4b 99 50 fd 62 48 84 f7 f5 7c d9  
8b 07 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 3a b1 17 00
84 1f ce 19 3c 32 39 11 ed b3 17 b0 46 dc f2 4b 99 50 fd 62 48 84 f7
f5 7c d9 8b 07 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 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'a11822822e4879f2a41b510c1f9b58203ab11700841fce19
  3c323911edb317b046dcf24b9950fd624884f7f57cd98b07
  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.



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 3a b1 17
00 84 1f ce 19 3c 32 39 11 ed b3 17 b0 46 dc f2 4b 99 50 fd 62 48 84
f7 f5 7c d9 8b 07 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)

```
7a b9 61 ac 76 30 26 9a 99 5a 72 9a 0f ce ad 31 f5 cd 97 fb 51 5b c5
db 9c 11 19 83 3e 4c 3b 4a
```

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

```
58 20 7a b9 61 ac 76 30 26 9a 99 5a 72 9a 0f ce ad 31 f5 cd 97 fb 51
5b c5 db 9c 11 19 83 3e 4c 3b 4a
```

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

R constructs the message to be signed:

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

```
[
  "Signature1",
  h'a11822822e4879f2a41b510c1f9b',
  h'58203ab11700841fce193c323911edb317b046dcf24b9950
    fd624884f7f57cd98b0758f13081ee3081a1a00302010202
    0462319ec4300506032b6570301d311b301906035504030c
    124544484f4320526f6f742045643235353139301e170d32
    32303331363038323433365a170d32393132333132333030
    30305a30223120301e06035504030c174544484f43205265
    73706f6e6465722045643235353139302a300506032b6570
    032100a1db47b95184854ad12a0c1a354e418aace33aa0f2
    c662c00b3ac55de92f9359300506032b6570034100b723bc
    01eab0928e8b2b6c98de19cc3823d46e7d6987b032478fec
    faf14537a1af14cc8be829c6b73044101837eb4abc949565
    d86dce51cfae52ab82c152cb02',
  h'7ab961ac7630269a995a729a0fceed31f5cd97fb515bc5db
    9c1119833e4c3b4a'
]
```

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 3a b1 17 00 84 1f ce 19 3c 32 39 11 ed b3
17 b0 46 dc f2 4b 99 50 fd 62 48 84 f7 f5 7c d9 8b 07 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 7a b9 61 ac 76 30 26 9a 99 5a 72 9a 0f
ce ad 31 f5 cd 97 fb 51 5b c5 db 9c 11 19 83 3e 4c 3b 4a
```

R signs using the private authentication key SK\_R

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

```
af 73 81 f1 9a e1 fe 0f 53 89 5b 18 e5 81 8b 1f e3 e3 46 30 72 c0 2a
d3 9f 20 2d 38 28 aa 62 37 c1 0b 08 66 8f c4 76 96 41 24 03 1f ed 9f
94 4e 6a 78 79 7f 5c 08 49 58 db 0f 20 89 c2 1c 52 02
```

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

```
58 40 af 73 81 f1 9a e1 fe 0f 53 89 5b 18 e5 81 8b 1f e3 e3 46 30 72
c0 2a d3 9f 20 2d 38 28 aa 62 37 c1 0b 08 66 8f c4 76 96 41 24 03 1f
ed 9f 94 4e 6a 78 79 7f 5c 08 49 58 db 0f 20 89 c2 1c 52 02
```

R constructs the plaintext without padding:

PAD\_2 (CBOR sequence of simple type) (0 bytes)

PLAINTEXT\_2 =

```
(
  ? PAD_2,
  ID_CRED_R / bstr / -24..23,
  Signature_or_MAC_2,
  ? EAD_2
)
```

PLAINTEXT\_2 (CBOR Sequence) (80 bytes)

```
a1 18 22 82 2e 48 79 f2 a4 1b 51 0c 1f 9b 58 40 af 73 81 f1 9a e1 fe
0f 53 89 5b 18 e5 81 8b 1f e3 e3 46 30 72 c0 2a d3 9f 20 2d 38 28 aa
62 37 c1 0b 08 66 8f c4 76 96 41 24 03 1f ed 9f 94 4e 6a 78 79 7f 5c
08 49 58 db 0f 20 89 c2 1c 52 02
```

The input needed to calculate KEYSTREAM\_2 is defined in [Section 4.1.2](#) of [[I-D.ietf-lake-edhoc](#)], using 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 of PLAINTEXT\_2, and info for KEYSTREAM\_2 is:

```
info =
(
0,
h'3ab11700841fce193c323911edb317b046dcf24b9950fd62
4884f7f57cd98b07',
80
)
```

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

```
info for KEYSTREAM_2 (CBOR Sequence) (37 bytes)
00 58 20 3a b1 17 00 84 1f ce 19 3c 32 39 11 ed b3 17 b0 46 dc f2 4b
99 50 fd 62 48 84 f7 f5 7c d9 8b 07 18 50
KEYSTREAM_2 (Raw Value) (80 bytes)
c6 a1 ed d7 c9 ff 34 20 38 c7 b7 82 43 e4 1a dc f0 84 6c 7e 80 22 05
4f 66 34 69 4c 57 ea e8 b7 b4 ca 1c cb 5d 1d 64 94 0e 14 0f 02 b4 73
fb 18 f1 64 a7 3a 04 13 57 4a 0e 96 d8 28 3e e9 2f aa 58 36 30 cf 47
ac 7d 9a 06 c3 83 cd f3 bb 4e 71
```

R calculates CIPHERTEXT\_2 as XOR between PLAINTEXT\_2 and KEYSTREAM\_2:

```
CIPHERTEXT_2 (Raw Value) (80 bytes)
67 b9 cf 55 e7 b7 4d d2 9c dc e6 8e 5c 7f 42 9c 5f f7 ed 8f 1a c3 fb
40 35 bd 32 54 b2 6b 63 a8 57 29 5a fb 2f dd 4e 47 91 34 22 3a 9c d9
99 2f 30 6f af 5c 8b d7 21 dc 4f b2 db 37 d3 76 bb e4 32 4e 49 b0 1b
a4 34 c2 dd cc a3 44 31 a7 1c 73
```

R constructs message\_2:

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

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 70 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 67 b9 cf 55 e7 b7 4d d2 9c dc e6 8e
5c 7f 42 9c 5f f7 ed 8f 1a c3 fb 40 35 bd 32 54 b2 6b 63 a8 57 29 5a
fb 2f dd 4e 47 91 34 22 3a 9c d9 99 2f 30 6f af 5c 8b d7 21 dc 4f b2
db 37 d3 76 bb e4 32 4e 49 b0 1b a4 34 c2 dd cc a3 44 31 a7 1c 73 41
18
```

### 3.3. message\_3

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

I'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 I authenticates with signatures PRK\_4e3m = PRK\_3e2m.

PRK\_4e3m (Raw Value) (32 bytes)

2a e2 42 1d e9 a7 2a 7a e6 71 5f b5 18 f3 ed 30 05 8f d9 ca 58 b6 25  
68 ca fe 7c da a1 5a 41 f7

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) (357 bytes)

58 20 3a b1 17 00 84 1f ce 19 3c 32 39 11 ed b3 17 b0 46 dc f2 4b 99  
50 fd 62 48 84 f7 f5 7c d9 8b 07 a1 18 22 82 2e 48 79 f2 a4 1b 51 0c  
1f 9b 58 40 af 73 81 f1 9a e1 fe 0f 53 89 5b 18 e5 81 8b 1f e3 e3 46  
30 72 c0 2a d3 9f 20 2d 38 28 aa 62 37 c1 0b 08 66 8f c4 76 96 41 24  
03 1f ed 9f 94 4e 6a 78 79 7f 5c 08 49 58 db 0f 20 89 c2 1c 52 02 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

TH\_3 (Raw Value) (32 bytes)

03 12 56 1b 73 43 ce af 65 9d f5 00 13 e0 64 e6 b4 6d cb 3f a8 40 d8  
55 04 5e 33 c0 21 d7 f6 91

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

58 20 03 12 56 1b 73 43 ce af 65 9d f5 00 13 e0 64 e6 b4 6d cb 3f a8  
40 d8 55 04 5e 33 c0 21 d7 f6 91

I 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 3.8.2](#):

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 03 12 56 1b 73 43 ce  
af 65 9d f5 00 13 e0 64 e6 b4 6d cb 3f a8 40 d8 55 04 5e 33 c0 21 d7  
f6 91 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 03 12 56 1b
73 43 ce af 65 9d f5 00 13 e0 64 e6 b4 6d cb 3f a8 40 d8 55 04 5e 33
c0 21 d7 f6 91 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 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'a11822822e48c24ab2fd7643c79f58200312561b7343ceaf
  659df50013e064e6b46dcb3fa840d855045e33c021d7f691
  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.

```

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 03 12 56
1b 73 43 ce af 65 9d f5 00 13 e0 64 e6 b4 6d cb 3f a8 40 d8 55 04 5e
33 c0 21 d7 f6 91 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)
cd d2 50 7b cf 66 2b 5d 9d 9c f2 3c 4b 31 a9 b6 66 c6 a1 9a 0a 44 dc
2a 7a 9c 90 45 22 b1 eb 3e
MAC_3 (CBOR Data Item) (34 bytes)
58 20 cd d2 50 7b cf 66 2b 5d 9d 9c f2 3c 4b 31 a9 b6 66 c6 a1 9a 0a
44 dc 2a 7a 9c 90 45 22 b1 eb 3e

```

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

I constructs the message to be signed:

```

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

[
  "Signature1",
  h'a11822822e48c24ab2fd7643c79f',
  h'58200312561b7343ceaf659df50013e064e6b46dcb3fa840
    d855045e33c021d7f69158f13081ee3081a1a00302010202
    0462319ea0300506032b6570301d311b301906035504030c
    124544484f4320526f6f742045643235353139301e170d32
    32303331363038323430305a170d32393132333132333030
    30305a30223120301e06035504030c174544484f4320496e
    69746961746f722045643235353139302a300506032b6570
    032100ed06a8ae61a829ba5fa54525c9d07f48dd44a302f4
    3e0f23d8cc20b73085141e300506032b6570034100521241
    d8b3a770996bcfc9b9ead4e7e0a1c0db353a3bdf2910b392
    75ae48b756015981850d27db6734e37f67212267dd05eef
    27b9e7a813fa574b72a00b430b',
  h'cdd2507bcf662b5d9d9cf23c4b31a9b666c6a19a0a44dc2a
    7a9c904522b1eb3e'
]

```

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 03 12 56 1b 73 43 ce af 65 9d f5 00 13 e0
64 e6 b4 6d cb 3f a8 40 d8 55 04 5e 33 c0 21 d7 f6 91 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 cd d2 50 7b cf 66 2b 5d 9d 9c f2 3c 4b
31 a9 b6 66 c6 a1 9a 0a 44 dc 2a 7a 9c 90 45 22 b1 eb 3e
```

I signs using the private authentication key SK\_I:

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

```
4f 99 22 77 bc be 5f ec 00 9f be 0b 31 34 91 65 2b d4 c8 02 18 07 32
75 c3 f1 66 99 af 9c d3 f4 c6 b6 61 ff 11 da 12 b3 fe 03 c5 df d2 ce
ee c0 6a dc ff 6a 76 a7 0e 31 56 a6 00 fb 61 ac d6 02
```

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

```
58 40 4f 99 22 77 bc be 5f ec 00 9f be 0b 31 34 91 65 2b d4 c8 02 18
07 32 75 c3 f1 66 99 af 9c d3 f4 c6 b6 61 ff 11 da 12 b3 fe 03 c5 df
d2 ce ee c0 6a dc ff 6a 76 a7 0e 31 56 a6 00 fb 61 ac d6 02
```

I constructs the plaintext without padding:

PAD\_3 (CBOR sequence of simple type) (0 bytes)

PLAINTEXT\_3 =

```
(
  ? PAD_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 4f 99 22 77 bc be 5f
ec 00 9f be 0b 31 34 91 65 2b d4 c8 02 18 07 32 75 c3 f1 66 99 af 9c
d3 f4 c6 b6 61 ff 11 da 12 b3 fe 03 c5 df d2 ce ee c0 6a dc ff 6a 76
a7 0e 31 56 a6 00 fb 61 ac d6 02
```

I constructs the associated data for message\_3:

A\_3 =

```
[
  "Encrypt0",
  h'',
  h'0312561b7343ceaf659df50013e064e6b46dcb3fa840d855
  045e33c021d7f691'
]
```

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

```
83 68 45 6e 63 72 79 70 74 30 40 58 20 03 12 56 1b 73 43 ce af 65 9d
f5 00 13 e0 64 e6 b4 6d cb 3f a8 40 d8 55 04 5e 33 c0 21 d7 f6 91
```



I constructs the input needed to derive the key K<sub>3</sub>, see [Section 4.1.2](#) of [[I-D.ietf-lake-edhoc](#)], using the EDHOC hash algorithm:

```
K3 = EDHOC-KDF( PRK3e2m, 3, TH3, key_length )
      = HKDF-Expand( PRK3e2m, info, key_length ),
```

where key\_length is the key length of EDHOC AEAD algorithm, and info for K<sub>3</sub> is:

```
info =
(
  3,
  h'0312561b7343ceaf659df50013e064e6b46dcb3fa840d855
    045e33c021d7f691',
  16
)
```

where the last value is the key length of EDHOC AEAD algorithm.

```
info for K3 (CBOR Sequence) (36 bytes)
03 58 20 03 12 56 1b 73 43 ce af 65 9d f5 00 13 e0 64 e6 b4 6d cb 3f
a8 40 d8 55 04 5e 33 c0 21 d7 f6 91 10
K3 (Raw Value) (16 bytes)
50 b9 cb 0b ba 0c 75 88 0b 54 27 86 be 62 77 fa
```

I constructs the input needed to derive the nonce IV<sub>3</sub>, see [Section 4.1.2](#) of [[I-D.ietf-lake-edhoc](#)], using the EDHOC hash algorithm:

```
IV3 = EDHOC-KDF( PRK3e2m, 4, TH3, iv_length )
      = HKDF-Expand( PRK3e2m, info, iv_length ),
```

where iv\_length is the nonce length of EDHOC AEAD algorithm, and info for IV<sub>3</sub> is:

```
info =
(
  4,
  h'0312561b7343ceaf659df50013e064e6b46dcb3fa840d855
    045e33c021d7f691',
  13
)
```

where the last value is the nonce length of EDHOC AEAD algorithm.

```
info for IV3 (CBOR Sequence) (36 bytes)
04 58 20 03 12 56 1b 73 43 ce af 65 9d f5 00 13 e0 64 e6 b4 6d cb 3f
a8 40 d8 55 04 5e 33 c0 21 d7 f6 91 0d
IV3 (Raw Value) (13 bytes)
27 a3 b3 ba 30 14 ab 62 d9 a2 69 45 a3
```

I calculates CIPHERTEXT<sub>3</sub> as 'ciphertext' of COSE\_Encrypt0 applied using the EDHOC AEAD algorithm with plaintext PLAINTEXT<sub>3</sub>, additional data A<sub>3</sub>, key K<sub>3</sub> and nonce IV<sub>3</sub>.

CIPHERTEXT\_3 (Raw Value) (88 bytes)

```
ba 5e 0e 74 5b fa 2a 87 1d 20 cb 02 c8 00 20 07 71 43 4b 6e 1a c9 89
77 ec 73 3e c9 4c 06 33 cb 3e c0 20 78 98 59 7f 2c 49 d3 a4 0f 4c 14
51 b4 3d 0b ca e4 84 7a 0d 6c d3 2d 5e 8a 35 54 f4 3f 7a 98 29 04 b0
77 c5 02 9b 3d c7 f0 5e ed ed e3 b0 21 57 c3 24 c0 db 3e
```

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

message\_3 (CBOR Sequence) (90 bytes)

```
58 58 ba 5e 0e 74 5b fa 2a 87 1d 20 cb 02 c8 00 20 07 71 43 4b 6e 1a
c9 89 77 ec 73 3e c9 4c 06 33 cb 3e c0 20 78 98 59 7f 2c 49 d3 a4 0f
4c 14 51 b4 3d 0b ca e4 84 7a 0d 6c d3 2d 5e 8a 35 54 f4 3f 7a 98 29
04 b0 77 c5 02 9b 3d c7 f0 5e ed ed e3 b0 21 57 c3 24 c0 db 3e
```

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 03 12 56 1b 73 43 ce af 65 9d f5 00 13 e0 64 e6 b4 6d cb 3f a8
40 d8 55 04 5e 33 c0 21 d7 f6 91 a1 18 22 82 2e 48 c2 4a b2 fd 76 43
c7 9f 58 40 4f 99 22 77 bc be 5f ec 00 9f be 0b 31 34 91 65 2b d4 c8
02 18 07 32 75 c3 f1 66 99 af 9c d3 f4 c6 b6 61 ff 11 da 12 b3 fe 03
c5 df d2 ce ee c0 6a dc ff 6a 76 a7 0e 31 56 a6 00 fb 61 ac d6 02 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)

```
38 e2 e6 f4 64 1e 81 4b 72 18 14 c0 5b 51 ef 0a a3 8b db 36 07 4f 98
12 39 e6 47 4d 9c cc dd c8
```

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

```
58 20 38 e2 e6 f4 64 1e 81 4b 72 18 14 c0 5b 51 ef 0a a3 8b db 36 07
4f 98 12 39 e6 47 4d 9c cc dd c8
```

### 3.4. message\_4

No external authorization data:

EAD\_4 (CBOR Sequence) (0 bytes)

R constructs the plaintext PLAINTEXT\_4:

PLAINTEXT\_4 =

```
(
  ? EAD_4
)
```

PLAINTEXT\_4 (CBOR Sequence) (0 bytes)

R constructs the associated data for message\_4:

```
A_4 =
[
  "Encrypt0",
  h'',
  h'38e2e6f4641e814b721814c05b51ef0aa38bdb36074f9812
    39e6474d9ccddc8'
]
```

```
A_4 (CBOR Data Item) (45 bytes)
83 68 45 6e 63 72 79 70 74 30 40 58 20 38 e2 e6 f4 64 1e 81 4b 72 18
14 c0 5b 51 ef 0a a3 8b db 36 07 4f 98 12 39 e6 47 4d 9c cc dd c8
```

R 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 of the EDHOC AEAD algorithm, and info for EDHOC\_K\_4 is:

```
info =
(
  8,
  h'38e2e6f4641e814b721814c05b51ef0aa38bdb36074f9812
    39e6474d9ccddc8',
  16
)
```

where the last value is the key length of EDHOC AEAD algorithm.

```
info for K_4 (CBOR Sequence) (36 bytes)
08 58 20 38 e2 e6 f4 64 1e 81 4b 72 18 14 c0 5b 51 ef 0a a3 8b db 36
07 4f 98 12 39 e6 47 4d 9c cc dd c8 10
K_4 (Raw Value) (16 bytes)
3d e5 c1 6f 9f 7e f0 0c 46 4b e8 d7 7b de f7 30
```

R 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 of EDHOC AEAD algorithm, and info for EDHOC\_IV\_4 is:

```
info =
(
  9,
  h'38e2e6f4641e814b721814c05b51ef0aa38bdb36074f9812
    39e6474d9ccddc8',
  13
)
```

where the last value is the nonce length of EDHOC AEAD algorithm.

```
info for IV_4 (CBOR Sequence) (36 bytes)
09 58 20 38 e2 e6 f4 64 1e 81 4b 72 18 14 c0 5b 51 ef 0a a3 8b db 36
07 4f 98 12 39 e6 47 4d 9c cc dd c8 0d
IV_4 (Raw Value) (13 bytes)
26 35 c2 b3 6d 2b f8 af b6 c8 9b 0f af
```

R 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)
d5 41 7c 47 4c b4 a3 02
```

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

```
message_4 (CBOR Sequence) (9 bytes)
48 d5 41 7c 47 4c b4 a3 02
```

### 3.5. 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 of the output of the EDHOC hash algorithm, and info for PRK\_out is:

```
info =
(
  7,
  h'38e2e6f4641e814b721814c05b51ef0aa38bdb36074f9812
  39e6474d9ccccddc8',
  32
)
```

where the last value is the length of EDHOC hash algorithm.

```
info for PRK_out (CBOR Sequence) (37 bytes)
07 58 20 38 e2 e6 f4 64 1e 81 4b 72 18 14 c0 5b 51 ef 0a a3 8b db 36
07 4f 98 12 39 e6 47 4d 9c cc dd c8 18 20
PRK_out (Raw Value) (32 bytes)
cf aa 94 87 37 c8 c7 5f 54 2a fb 6a 07 df da 67 3e 78 a1 04 ca cb d9
3f dc a3 c2 b0 e6 63 e9 44
```

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 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 of EDHOC hash algorithm.

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

```
0a 40 18 20
```

PRK\_exporter (Raw Value) (32 bytes)

```
55 15 9b 06 37 4e 4b 2b c2 a9 f5 82 4b 56 1f e1 66 d6 26 4a a6 da e8
97 7d 2e d5 37 90 b4 2b 2f
```

### 3.6. 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 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 of the Application AEAD algorithm, and info for the OSCORE Master Secret is:

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

where the last value is the key length of Application AEAD algorithm.

```
info for OSCORE Master Secret (CBOR Sequence) (3 bytes)
00 40 10
OSCORE Master Secret (Raw Value) (16 bytes)
09 c3 66 61 cf 68 f8 c3 ad 21 64 43 cf 62 91 e6
```

The OSCORE Master Salt is computed through 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 of the OSCORE Master Salt, and info for the OSCORE Master Salt is:

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

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

```
info for OSCORE Master Salt (CBOR Sequence) (3 bytes)
01 40 08
OSCORE Master Salt (Raw Value) (8 bytes)
13 82 bf 71 9e e6 5c 32
```

### 3.7. Key Update

Key update is defined in [Appendix J](#) 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 of the output of the EDHOC hash function, 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)

```
2b 31 bf cf 9b 0b b2 a6 92 65 3a 08 40 02 73 59 c4 e6 7c c5 04 ff 65
7a 30 af d7 67 c5 a4 1e f9
```

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 as unchanged as in [Section 3.5](#).

PRK\_exporter (Raw Value) (32 bytes)

```
2c 62 c4 ac 76 c8 e1 e8 48 38 5b 07 fe 2a 58 ad 2a f7 4c ee 38 70 d5
2b 4d a1 ec 63 39 3d 0f ec
```

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 3.6](#).

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

```
f0 05 28 0c 94 8a 64 c4 6e 33 e9 ea 8d e9 31 15
```

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 3.6](#).

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

```
0b 0a f3 2a a4 9b 3c e3
```

### 3.8. Certificates

#### 3.8.1. Responder Certificate

Version: 3 (0x2)  
Serial Number: 1647419076 (0x62319ec4)  
Signature Algorithm: ED25519  
Issuer: CN = EDHOC Root Ed25519  
Validity  
    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

#### 3.8.2. 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



### 3.8.3. 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
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
```

## 4. Authentication with static DH, CCS identified by 'kid'

In this example I and R are authenticated with ephemeral-static Diffie-Hellman (METHOD = 3). I supports cipher suites 6 and 2 (in order of preference) and R 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'.

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

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

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

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

SUITES\_I (CBOR Data Item) (1 byte)  
06

I 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, 'x'-coordinate

G\_X (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

I 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:

C\_I (Raw Value) (Connection identifier chosen by I) (1 byte)

0e

C\_I (CBOR Data Item) (Connection identifier chosen by I) (1 byte)

0e

No external authorization data:

EAD\_1 (CBOR Sequence) (0 bytes)

I 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

#### 4.2. error

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

SUITES\_R

02

error (CBOR Sequence) (2 bytes)  
02 02

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

Same steps are performed as message\_1 first time, [Section 4.1](#), but with updated SUITES\_I.

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

METHOD (CBOR Data Item) (1 byte)  
03

I 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

I 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, '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

I 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:

C\_I (Raw Value) (Connection identifier chosen by I) (1 byte)

37

C\_I (CBOR Data Item) (Connection identifier chosen by I) (1 byte)

37

No external authorization data:

EAD\_1 (CBOR Sequence) (0 bytes)

I 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

```

#### 4.4. message\_2

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

R 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, '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

```

R 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:

```

C_R (raw value) (Connection identifier chosen by R) (1 byte)
27
C_R (CBOR Data Item) (Connection identifier chosen by R) (1 byte)
27

```

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

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

Input to calculate TH\_2 (CBOR Sequence) (69 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 27 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)

```
9d 2a f3 a3 d3 fc 06 ae a8 11 0f 14 ba 12 ad 0b 4f b7 e5 cd f5 9c 7d
f1 cf 2d fe 9c 20 24 43 9c
```

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

```
58 20 9d 2a f3 a3 d3 fc 06 ae a8 11 0f 14 ba 12 ad 0b 4f b7 e5 cd f5
9c 7d f1 cf 2d fe 9c 20 24 43 9c
```

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 Extract() determined by the EDHOC hash algorithm:

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

where salt is TH\_2:

salt (Raw Value) (32 bytes)

```
9d 2a f3 a3 d3 fc 06 ae a8 11 0f 14 ba 12 ad 0b 4f b7 e5 cd f5 9c 7d
f1 cf 2d fe 9c 20 24 43 9c
```

PRK\_2e (Raw Value) (32 bytes)

```
e0 1f a1 4d d5 6e 30 82 67 a1 a8 12 a9 d0 b9 53 41 e3 94 ab c7 c5 c3
9d d7 18 85 f7 d4 cd 5b f3
```

Since METHOD = 3, R 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.

R's static Diffie-Hellman key pair for use with P-256:

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

PK\_R (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 R 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 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 of the output of the EDHOC hash algorithm, and info for SALT\_3e2m is:

```
info =  
(  
  1,  
  h'9d2af3a3d3fc06aea8110f14ba12ad0b4fb7e5cdf59c7df1  
    cf2dfe9c2024439c',  
  32  
)  
info for SALT_3e2m (CBOR Sequence) (37 bytes)  
01 58 20 9d 2a f3 a3 d3 fc 06 ae a8 11 0f 14 ba 12 ad 0b 4f b7 e5 cd  
f5 9c 7d f1 cf 2d fe 9c 20 24 43 9c 18 20  
SALT_3e2m (Raw Value) (32 bytes)  
a4 f7 67 b3 46 9a 6e 6a e5 fc bf 27 38 39 fa 87 c4 1f 46 2b 03 ad 1c  
a7 ce 8f 37 c9 53 66 d8 d1
```

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

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

```
PRK_3e2m = 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)  
41 2d 60 cd f9 9d c7 49 07 54 c9 69 ad 4c 46 b1 35 0b 90 84 33 eb f3  
fe 06 3b e8 62 7f b3 5b 3b
```

R 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",
  8 : {
    1 : {
      1 : 2,
      2 : h'32',
      -1 : 1,
      -2 : h'BBC34960526EA4D32E940CAD2A234148
          DDC21791A12AFBCBAC93622046DD44F0',
      -3 : h'4519E257236B2A0CE2023F0931F1F386
          CA7AFDA64FCDE0108C224C51EABF6072'
    }
  }
}
```

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 9d 2a f3 a3 d3 fc 06 ae a8 11 0f 14 ba 12 ad 0b 4f  
b7 e5 cd f5 9c 7d f1 cf 2d fe 9c 20 24 43 9c 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 9d 2a f3 a3 d3 fc 06 ae a8 11 0f 14 ba 12 ad  
0b 4f b7 e5 cd f5 9c 7d f1 cf 2d fe 9c 20 24 43 9c 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 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'a104413258209d2af3a3d3fc06aea8110f14ba12ad0b4fb7
  e5cdf59c7df1cf2dfe9c2024439ca2026b6578616d706c65
  2e65647508a101a501020241322001215820bbc34960526e
  a4d32e940cad2a234148ddc21791a12afbcbac93622046dd
  44f02258204519e257236b2a0ce2023f0931f1f386ca7afd
  a64fcde0108c224c51eabf6072',
  8
)

```

where the last value is the EDHOC MAC length.

```

info for MAC_2 (CBOR Sequence) (137 bytes)
02 58 85 a1 04 41 32 58 20 9d 2a f3 a3 d3 fc 06 ae a8 11 0f 14 ba 12
ad 0b 4f b7 e5 cd f5 9c 7d f1 cf 2d fe 9c 20 24 43 9c 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)
d0 d1 a5 94 79 7d 0a af
MAC_2 (CBOR Data Item) (9 bytes)
48 d0 d1 a5 94 79 7d 0a af

```

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

```

Signature_or_MAC_2 (Raw Value) (8 bytes)
d0 d1 a5 94 79 7d 0a af
Signature_or_MAC_2 (CBOR Data Item) (9 bytes)
48 d0 d1 a5 94 79 7d 0a af

```

R constructs PLAINTEXT\_2 without padding:

```

PAD_2 (CBOR sequence of simple type) (0 bytes)
PLAINTEXT_2 =
(
  ? PAD_2,
  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) (10 bytes)
32 48 d0 d1 a5 94 79 7d 0a af

```

The input needed to calculate KEYSTREAM\_2 is defined in [Section 4.1.2](#) of [[I-D.ietf-lake-edhoc](#)], using 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 of PLAINTEXT\_2, and info for  
KEYSTREAM\_2 is:

```
info =  
(  
  0,  
  h'9d2af3a3d3fc06aea8110f14ba12ad0b4fb7e5cdf59c7df1  
    cf2dfe9c2024439c',  
  10  
)
```

where last value is the length of PLAINTEXT\_2.

```
info for KEYSTREAM_2 (CBOR Sequence) (36 bytes)  
00 58 20 9d 2a f3 a3 d3 fc 06 ae a8 11 0f 14 ba 12 ad 0b 4f b7 e5 cd  
f5 9c 7d f1 cf 2d fe 9c 20 24 43 9c 0a  
KEYSTREAM_2 (Raw Value) (10 bytes)  
36 6c 89 33 7f f8 0c 69 35 9a
```

R calculates CIPHERTEXT\_2 as XOR between PLAINTEXT\_2 and  
KEYSTREAM\_2:

```
CIPHERTEXT_2 (Raw Value) (10 bytes)  
04 24 59 e2 da 6c 75 14 3f 35
```

R constructs message\_2:

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

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 2a 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 04 24 59 e2 da 6c 75 14 3f 35 27
```

#### 4.5. 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) (139 bytes)  
58 20 9d 2a f3 a3 d3 fc 06 ae a8 11 0f 14 ba 12 ad 0b 4f b7 e5 cd f5  
9c 7d f1 cf 2d fe 9c 20 24 43 9c 32 48 d0 d1 a5 94 79 7d 0a af 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)
b7 78 f6 02 33 1f f6 8a c4 02 a6 51 1b 9d e2 85 be df 6e ab 3e 9e d1
2d fe 22 a5 3e ed a7 de 48
TH_3 (CBOR Data Item) (34 bytes)
58 20 b7 78 f6 02 33 1f f6 8a c4 02 a6 51 1b 9d e2 85 be df 6e ab 3e
9e d1 2d fe 22 a5 3e ed a7 de 48
```

Since METHOD = 3, I 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.

I's static Diffie-Hellman key pair for use with P-256:

```
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
PK_I (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 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 of the output of the EDHOC hash algorithm, and info for SALT\_4e3m is:

```
info =
(
  5,
  h'b778f602331ff68ac402a6511b9de285bedf6eab3e9ed12d
  fe22a53eeda7de48',
  32
)
info for SALT_4e3m (CBOR Sequence) (37 bytes)
05 58 20 b7 78 f6 02 33 1f f6 8a c4 02 a6 51 1b 9d e2 85 be df 6e ab
3e 9e d1 2d fe 22 a5 3e ed a7 de 48 18 20
SALT_4e3m (Raw Value) (32 bytes)
8c 60 d4 35 7f ba 5f 69 4a 81 48 2c 4d 38 a1 00 0b c3 e3 e2 a2 94 06
d1 81 53 ff c3 59 5c 17 ba
```

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 Extract() with the EDHOC hash algorithm:

```
PRK_4e3m = 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)

```
7d 01 59 bb e4 54 73 c9 40 2e 0d 42 db ce b4 5d ca 05 b7 44 ca e1 e0
83 e5 83 15 b8 aa 47 ce ec
```

I 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",           /CCS/
  8 : {                                     /sub/
    1 : {                                   /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 b7 78 f6 02 33 1f f6 8a c4 02 a6 51 1b 9d e2 85 be
df 6e ab 3e 9e d1 2d fe 22 a5 3e ed a7 de 48 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 b7 78 f6 02 33 1f f6 8a c4 02 a6 51 1b 9d e2
85 be df 6e ab 3e 9e d1 2d fe 22 a5 3e ed a7 de 48 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 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'a104412b5820b778f602331ff68ac402a6511b9de285bedf
  6eab3e9ed12dfe22a53eeda7de48a2027734322d35302d33
  312d46462d45462d33372d33322d333908a101a501020241
  2b2001215820ac75e9ece3e50bfc8ed60399889522405c47
  bf16df96660a41298cb4307f7eb62258206e5de611388a4b
  8a8211334ac7d37ecb52a387d257e6db3c2a93df21ff3aff
  c8',
  8
)
```

where the last value is the EDHOC MAC length.

```
info for MAC_3 (CBOR Sequence) (149 bytes)
06 58 91 a1 04 41 2b 58 20 b7 78 f6 02 33 1f f6 8a c4 02 a6 51 1b 9d
e2 85 be df 6e ab 3e 9e d1 2d fe 22 a5 3e ed a7 de 48 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)

```
dd f1 06 b8 6f d2 2f e4
```

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

```
48 dd f1 06 b8 6f d2 2f e4
```

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

Signature\_or\_MAC\_3 (Raw Value) (8 bytes)  
dd f1 06 b8 6f d2 2f e4  
Signature\_or\_MAC\_3 (CBOR Data Item) (9 bytes)  
48 dd f1 06 b8 6f d2 2f e4

I constructs PLAINTEXT\_3 without padding:

PAD\_3 (CBOR sequence of simple type) (0 bytes)  
PLAINTEXT\_3 =  
(  
  ? PAD\_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 dd f1 06 b8 6f d2 2f e4

I constructs the associated data for message\_3:

A\_3 =  
[  
  "Encrypt0",  
  h',  
  h'b778f602331ff68ac402a6511b9de285bedf6eab3e9ed12d  
  fe22a53eeda7de48'  
]  
A\_3 (CBOR Data Item) (45 bytes)  
83 68 45 6e 63 72 79 70 74 30 40 58 20 b7 78 f6 02 33 1f f6 8a c4 02  
a6 51 1b 9d e2 85 be df 6e ab 3e 9e d1 2d fe 22 a5 3e ed a7 de 48

I 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 of EDHOC AEAD algorithm, and info for K\_3 is:

info =  
(  
  3,  
  h'b778f602331ff68ac402a6511b9de285bedf6eab3e9ed12d  
  fe22a53eeda7de48',  
  16  
)

where the last value is the key length of EDHOC AEAD algorithm.

```
info for K_3 (CBOR Sequence) (36 bytes)
03 58 20 b7 78 f6 02 33 1f f6 8a c4 02 a6 51 1b 9d e2 85 be df 6e ab
3e 9e d1 2d fe 22 a5 3e ed a7 de 48 10
K_3 (Raw Value) (16 bytes)
2f 10 8b ef ff 80 6f 5f c8 1b f0 a2 d5 f4 24 1f
```

I 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 of EDHOC AEAD algorithm, and info for IV\_3 is:

```
info =
(
  4,
  h'b778f602331ff68ac402a6511b9de285bedf6eab3e9ed12d
  fe22a53eeda7de48',
  13
)
```

where the last value is the nonce length of EDHOC AEAD algorithm.

```
info for IV_3 (CBOR Sequence) (36 bytes)
04 58 20 b7 78 f6 02 33 1f f6 8a c4 02 a6 51 1b 9d e2 85 be df 6e ab
3e 9e d1 2d fe 22 a5 3e ed a7 de 48 0d
IV_3 (Raw Value) (13 bytes)
e3 ff 26 46 33 25 8e 49 46 2d 35 56 6d
```

I 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)
c2 b6 28 35 dc 9b 1f 53 41 9c 1d 3a 22 61 ee ed 35 05
```

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

```
message_3 (CBOR Sequence) (19 bytes)
52 c2 b6 28 35 dc 9b 1f 53 41 9c 1d 3a 22 61 ee ed 35 05
```

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 b7 78 f6 02 33 1f f6 8a c4 02 a6 51 1b 9d e2 85 be df 6e ab 3e
9e d1 2d fe 22 a5 3e ed a7 de 48 2b 48 dd f1 06 b8 6f d2 2f e4 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)
1f 57 da bf 8f 26 da 06 57 d9 84 0c 9b 10 77 c1 d4 c4 7d b2 43 a8 b4
13 60 a9 8e c4 cb 70 6b 70
TH_4 (CBOR Data Item) (34 bytes)
58 20 1f 57 da bf 8f 26 da 06 57 d9 84 0c 9b 10 77 c1 d4 c4 7d b2 43
a8 b4 13 60 a9 8e c4 cb 70 6b 70
```

#### 4.6. message\_4

No external authorization data:

EAD\_4 (CBOR Sequence) (0 bytes)

R constructs the plaintext PLAINTEXT\_4:

PLAINTEXT\_4 =

```
(
 ? EAD_4
)
```

PLAINTEXT\_4 (CBOR Sequence) (0 bytes)

R constructs the associated data for message\_4:

A\_4 =

```
[
 "Encrypt0",
 h'',
 h'1f57dabf8f26da0657d9840c9b1077c1d4c47db243a8b413
 60a98ec4cb706b70'
]
```

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

```
83 68 45 6e 63 72 79 70 74 30 40 58 20 1f 57 da bf 8f 26 da 06 57 d9
84 0c 9b 10 77 c1 d4 c4 7d b2 43 a8 b4 13 60 a9 8e c4 cb 70 6b 70
```

R 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 of the EDHOC AEAD algorithm, and info for EDHOC\_K\_4 is:

info =

```
(
 8,
 h'1f57dabf8f26da0657d9840c9b1077c1d4c47db243a8b413
 60a98ec4cb706b70',
 16
)
```

where the last value is the key length of EDHOC AEAD algorithm.

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

```
08 58 20 1f 57 da bf 8f 26 da 06 57 d9 84 0c 9b 10 77 c1 d4 c4 7d b2
43 a8 b4 13 60 a9 8e c4 cb 70 6b 70 10
```

K\_4 (Raw Value) (16 bytes)  
de 02 dc 03 6c b6 81 cd 53 80 d7 83 e8 53 14 2f

R 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 of EDHOC AEAD algorithm, and info for EDHOC\_IV\_4 is:

```
info =
(
  9,
  h'1f57dabf8f26da0657d9840c9b1077c1d4c47db243a8b413
    60a98ec4cb706b70',
  13
)
```

where the last value is the nonce length of EDHOC AEAD algorithm.

```
info for IV_4 (CBOR Sequence) (36 bytes)
09 58 20 1f 57 da bf 8f 26 da 06 57 d9 84 0c 9b 10 77 c1 d4 c4 7d b2
43 a8 b4 13 60 a9 8e c4 cb 70 6b 70 0d
IV_4 (Raw Value) (13 bytes)
c2 93 2c 74 55 f5 6c 82 57 59 23 39 59
```

R 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)
63 59 ad 21 f0 77 a9 d1
```

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

```
message_4 (CBOR Sequence) (9 bytes)
48 63 59 ad 21 f0 77 a9 d1
```

#### 4.7. 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 of the output of the EDHOC hash algorithm, and info for PRK\_out is:

```
info =
(
  7,
  h'1f57dabf8f26da0657d9840c9b1077c1d4c47db243a8b413
    60a98ec4cb706b70',
  32
)
```



where the last value is the length of EDHOC hash algorithm.

```
info for PRK_out (CBOR Sequence) (37 bytes)
07 58 20 1f 57 da bf 8f 26 da 06 57 d9 84 0c 9b 10 77 c1 d4 c4 7d b2
43 a8 b4 13 60 a9 8e c4 cb 70 6b 70 18 20
PRK_out (Raw Value) (32 bytes)
7d 0a 64 61 d8 38 48 ed d5 23 4c 5f 97 f4 b7 7c 1d 24 a7 12 09 29 29
20 cb 49 74 e5 59 f5 41 3d
```

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 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 of EDHOC hash algorithm.

```
info for PRK_exporter (CBOR Sequence) (4 bytes)
0a 40 18 20
PRK_exporter (Raw Value) (32 bytes)
52 d1 2a 79 52 00 96 b6 c4 be 60 cf a9 9e ad 2f d6 2a ba 58 aa fb 5c
c2 df 2e 04 52 ef 6c 0d d9
```

#### 4.8. 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 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 of the Application AEAD algorithm, and info for the OSCORE Master Secret is:

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

where the last value is the key length of Application AEAD algorithm.

info for OSCORE Master Secret (CBOR Sequence) (3 bytes)

00 40 10

OSCORE Master Secret (Raw Value) (16 bytes)

07 ce 22 f2 63 8f ca 40 4d de d7 2a 25 fa 45 f4

The OSCORE Master Salt is computed through 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 of the OSCORE Master Salt, and info for the OSCORE Master Salt is:

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

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

info for OSCORE Master Salt (CBOR Sequence) (3 bytes)  
01 40 08  
OSCORE Master Salt (Raw Value) (8 bytes)  
5b e3 82 5f 5a 52 84 b7

#### 4.9. Key Update

Key update is defined in [Appendix J](#) 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 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)  
cb ae fc 6c fe 8c 9d 65 09 0c 34 2e 4e 4f cd d6 07 98 19 85 db 6f 57  
67 e9 06 55 14 0e 3a 09 b1
```

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` as unchanged as in [Section 4.7](#).

```
PRK_exporter (Raw Value) (32 bytes)  
10 c3 69 11 e0 8a e5 25 13 b9 a8 a2 84 85 bf 3c eb 79 18 e4 c8 4e 5b  
ca ad 7a 21 1c 42 f0 13 3a
```

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 3.6](#).

```
OSCORE Master Secret after KeyUpdate (Raw Value) (16 bytes)  
4c 75 69 6c ba 17 9c a9 f6 87 07 ee dc de 76 e0
```

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 3.6](#).

OSCORE Master Salt after KeyUpdate (Raw Value) (8 bytes)  
9d 95 4f c2 e7 ab b4 d0

## 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. Informative References

- [[CborMe](#)] Bormann, C., "CBOR Playground", May 2018, <<http://cbor.me/>>.
- [[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-17, 12 October 2022, <<https://www.ietf.org/archive/id/draft-ietf-lake-edhoc-17.txt>>.
- [[RFC7748](#)] Langley, A., Hamburg, M., and S. Turner, "Elliptic Curves for Security", RFC 7748, DOI 10.17487/RFC7748, January 2016, <<https://www.rfc-editor.org/info/rfc7748>>.
- [[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/info/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/info/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/info/rfc8949>>.

## 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, Michel Veillette and Mališa Vučinić.

## Authors' Addresses

Göran Selander  
Ericsson  
SE-164 40 Stockholm

Sweden

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

John Preuß Mattsson  
Ericsson  
SE-164 40 Stockholm  
Sweden

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

Marek Serafin  
ASSA ABLOY  
32-080 Zabierzów  
Poland

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

Marco Tiloca  
RISE  
SE-164 40 Stockholm  
Sweden

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