Workgroup: Network Working Group
Internet-Draft: draft-ietf-lake-traces-00
Published: 25 November 2021
Intended Status: Standards Track
Expires: 29 May 2022
Authors: G. Selander J. Preuß Mattsson
Ericsson Ericsson

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 29 May 2022.

Table of Contents

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 traces in this draft are valid for versions -11 and -12 of [I-D.ietf-lake-edhoc]. A more extensive test vector suite and related code that was used to generate them can be found at: https://github.com/lake-wg/edhoc/tree/master/test-vectors-11.

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.

Both I and R are assumed to 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

External authorization data (EAD) is not used in these examples.

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 Section 3.4.1 and Appendix A.3 of [I-D.ietf-lake-edhoc]). Those bytes are not included in the traces in this document.

3. Authentication with static DH, CCS identified by 'kid'

In this example I and R are authenticated with ephemeral-static Diffie-Hellman (METHOD = 3). The public keys are represented as raw public keys (RPK), encoded in an CWT Claims Set (CCS) and identified by the COSE header parameter 'kid'.

3.1. message_1

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

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

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

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

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

```
X (Raw Value) (Initiator's ephemeral private key) (32 bytes)
b3 11 19 98 cb 3f 66 86 63 ed 42 51 c7 8b e6 e9 5a 4d a1 27 e4 f6 fe
e2 75 e8 55 d8 d9 df d8 ed

G_X (Raw Value) (Initiator's ephemeral public key) (32 bytes)
3a a9 eb 32 01 b3 36 7b 8c 8b e3 8d 91 e5 7a 2b 43 3e 67 88 8c 86 d2
ac 00 6a 52 08 42 ed 50 37

G_X (CBOR Data Item) (Initiator's ephemeral public key) (34 bytes)
58 20 3a a9 eb 32 01 b3 36 7b 8c 8b e3 8d 91 e5 7a 2b 43 3e 67 88 8c
86 d2 ac 00 6a 52 08 42 ed 50 37
```

I selects its connection identifier C_I to be the int 12:

```
C_I (Raw Value) (Connection identifier chosen by I) (int)
12

C_I (CBOR Data Item) (Connection identifier chosen by I) (1 bytes)
0c
```

No external authorization data:

```
EAD_1 (CBOR Sequence) (0 bytes)
```

I constructs message_1:

```
message_1 =
(
 3,
 0,
 h'3AA9EB3201B3367B8C8BE38D91E57A2B433E67888C86D2AC006A520842ED5037',
 12
)

message_1 (CBOR Sequence) (37 bytes)
03 00 58 20 3a a9 eb 32 01 b3 36 7b 8c 8b e3 8d 91 e5 7a 2b 43 3e 67
88 8c 86 d2 ac 00 6a 52 08 42 ed 50 37 0c
```

3.2. message_2

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

```
Y (Raw Value) (Responder's ephemeral private key) (32 bytes)
bd 86 ea f4 06 5a 83 6c d2 9d 0f 06 91 ca 2a 8e c1 3f 51 d1 c4 5e 1b
43 72 c0 cb e4 93 ce f6 bd
```

```
G_Y (Raw Value) (Responder's ephemeral public key) (32 bytes)
25 54 91 b0 5a 39 89 ff 2d 3f fe a6 20 98 aa b5 7c 16 0f 29 4e d9 48
01 8b 41 90 f7 d1 61 82 4e

G_Y (CBOR Data Item) (Responder's ephemeral public key) (34 bytes)
58 20 25 54 91 b0 5a 39 89 ff 2d 3f fe a6 20 98 aa b5 7c 16 0f 29 4e
d9 48 01 8b 41 90 f7 d1 61 82 4e
```

PRK_2e is specified in Section 4.1.1 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)
6d 26 60 ec 2b 30 15 d9 3f e6 5d ae a5 12 74 bd 5b 1e bb ad 9b 62 4e
67 0e 79 a6 55 e3 0e c3 4d
```

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 the zero-length byte string:

```
salt (Raw Value) (0 bytes)

PRK_2e (Raw Value) (32 bytes)
d1 d0 11 a5 9a 6d 10 57 5e b2 20 c7 65 2e 6f 98 c4 17 a5 65 e4 e4 5c
f5 b5 01 06 95 04 3b 0e b7
```

Since METHOD = 3, R authenticates using static DH.

R's static key pair for use with the EDHOC key exchange algorithm is based on the same curve as for the ephemeral keys, X25519:

```
R (Raw Value) (Responder's private authentication key) (32 bytes)
52 8b 49 c6 70 f8 fc 16 a2 ad 95 c1 88 5b 2e 24 fb 15 76 22 72 79 2a
a1 cf 05 1d f5 d9 3d 36 94

G_R (Raw Value) (Responder's public authentication key) (32 bytes)
e6 6f 35 59 90 22 3c 3f 6c af f8 62 e4 07 ed d1 17 4d 07 01 a0 9e cd
6a 15 ce e2 c6 ce 21 aa 50
```

PRK_3e2m is specified in Section 4.1.2 of [I-D.ietf-lake-edhoc].

Since R authenticates with static DH (METHOD = 3), PRK_3e2m is derived from G_RX using Extract() with the EDHOC hash algorithm:

```
PRK_3e2m = Extract(PRK_2e, G_RX) =
         = HMAC-SHA-256(PRK_2e, 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)
b5 8b 40 34 26 c0 3d b0 7b aa 93 44 d5 51 e6 7b 21 78 bf 05 ec 6f 52
c3 6a 2f a5 be 23 2d d4 78

PRK_3e2m (Raw Value) (32 bytes)
76 8e 13 75 27 2e 1e 68 b4 2c a3 24 84 80 d5 bb a8 8b cb 55 f6 60 ce
7f 94 1e 67 09 10 31 17 a1
```

R selects its connection identifier C_R to be the empty byte string "":

```
C_R (raw value) (Connection identifier chosen by R) (0 bytes)


C_R (CBOR Data Item) (Connection identifier chosen by R) (1 bytes)
40
```

The transcript hash TH_2 is calculated using the EDHOC hash algorithm:

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

H(message_1) (Raw Value) (32 bytes)
9b dd b0 cd 55 48 7f 82 a8 6f b7 2a 8b b3 58 52 68 91 a0 a6 c9 08 61
24 12 f5 af 29 9d af 01 96

H(message_1) (CBOR Data Item) (34 bytes)
58 20 9b dd b0 cd 55 48 7f 82 a8 6f b7 2a 8b b3 58 52 68 91 a0 a6 c9
08 61 24 12 f5 af 29 9d af 01 96
```

The input to calculate TH_2 is the CBOR sequence:

```
H(message_1), G_Y, C_R

Input to calculate TH_2 (CBOR Sequence) (69 bytes)
58 20 9b dd b0 cd 55 48 7f 82 a8 6f b7 2a 8b b3 58 52 68 91 a0 a6 c9
08 61 24 12 f5 af 29 9d af 01 96 58 20 25 54 91 b0 5a 39 89 ff 2d 3f
fe a6 20 98 aa b5 7c 16 0f 29 4e d9 48 01 8b 41 90 f7 d1 61 82 4e 40

TH_2 (Raw Value) (32 bytes)
71 a6 c7 c5 ba 9a d4 7f e7 2d a4 dc 35 9b f6 b2 76 d3 51 59 68 71 1b
9a 91 1c 71 fc 09 6a ee 0e
```

```
TH_2 (CBOR Data Item) (34 bytes)
58 20 71 a6 c7 c5 ba 9a d4 7f e7 2d a4 dc 35 9b f6 b2 76 d3 51 59 68
71 1b 9a 91 1c 71 fc 09 6a ee 0e
```

R constructs the remaining input needed to calculate MAC_2:

MAC_2 = EDHOC-KDF(PRK_3e2m, TH_2, "MAC_2", << ID_CRED_R, CRED_R, ? EAD_2 >>, mac_length_2)

CRED_R is identified by a 'kid' with integer value 5:

```
ID_CRED_R =
{
 4 : 5
}

ID_CRED_R (CBOR Data Item) (3 bytes)
a1 04 05
```

CRED_R is an RPK encoded as a CCS:

```
{                                                  /CCS/
  2 : "example.edu",                               /sub/
  8 : {                                            /cnf/
    1 : {                                          /COSE_Key/
      1 : 1,                                       /kty/
      2 : 5,                                       /kid/
     -1 : 4,                                       /crv/
     -2 : h'E66F355990223C3F6CAFF862E407EDD1       /x/
           174D0701A09ECD6A15CEE2C6CE21AA50'
    }
  }
}

CRED_R (CBOR Data Item) (59 bytes)
a2 02 6b 65 78 61 6d 70 6c 65 2e 65 64 75 08 a1 01 a4 01 01 02 05 20
04 21 58 20 e6 6f 35 59 90 22 3c 3f 6c af f8 62 e4 07 ed d1 17 4d 07
01 a0 9e cd 6a 15 ce e2 c6 ce 21 aa 50
```

No external authorization data:

```
EAD_2 (CBOR Sequence) (0 bytes)
```

MAC_2 is computed through Expand() using the EDHOC hash algorithm, see Section 4.2 of [I-D.ietf-lake-edhoc]:

MAC_2 = HKDF-Expand(PRK_3e2m, info, mac_length_2)

Since METHOD = 3, mac_length_2 is given by the EDHOC MAC length.

   info for MAC_2 is:

```
info =
(
 h'71A6C7C5BA9AD47FE72DA4DC359BF6B276D3515968711B9A911C71FC096AEE0E',
 "MAC_2",
 h'A10405A2026B6578616D706C652E65647508A101A401010205200421582 0E6
   6F355990223C3F6CAFF862E407EDD1174D0701A09ECD6A15CEE2C6CE21AA50',
 8
)
```

   where the last value is the EDHOC MAC length.

```
info for MAC_2 (CBOR Sequence) (105 bytes)
58 20 71 a6 c7 c5 ba 9a d4 7f e7 2d a4 dc 35 9b f6 b2 76 d3 51 59 68
71 1b 9a 91 1c 71 fc 09 6a ee 0e 65 4d 41 43 5f 32 58 3e a1 04 05 a2
02 6b 65 78 61 6d 70 6c 65 2e 65 64 75 08 a1 01 a4 01 01 02 05 20 04
21 58 20 e6 6f 35 59 90 22 3c 3f 6c af f8 62 e4 07 ed d1 17 4d 07 01
a0 9e cd 6a 15 ce e2 c6 ce 21 aa 50 08

MAC_2 (Raw Value) (8 bytes)
8e 27 cb d4 94 f7 52 83

MAC_2 (CBOR Data Item) (9 bytes)
48 8e 27 cb d4 94 f7 52 83
```

   Since METHOD = 3, Signature_or_MAC_2 is MAC_2:

```
Signature_or_MAC_2 (Raw Value) (8 bytes)
8e 27 cb d4 94 f7 52 83

Signature_or_MAC_2 (CBOR Data Item) (9 bytes)
48 8e 27 cb d4 94 f7 52 83
```

   R constructs the plaintext:

```
PLAINTEXT_2 =
(
 ID_CRED_R / bstr / int,
 Signature_or_MAC_2,
 ? EAD_2
)
```

   Since ID_CRED_R contains a single 'kid' parameter, only the int 5 is
   included in the plaintext.

```
PLAINTEXT_2 (CBOR Sequence) (10 bytes)
05 48 8e 27 cb d4 94 f7 52 83
```

The input needed to calculate KEYSTREAM_2 is defined in Section 4.2 of [I-D.ietf-lake-edhoc], using Expand() with the EDHOC hash algorithm:

```
KEYSTREAM_2 = EDHOC-KDF(PRK_2e, TH_2, "KEYSTREAM_2", h'', length) =
            = HKDF-Expand(PRK_2e, info, length),
```

where length is the length of PLAINTEXT_2, and info for KEYSTREAM_2 is:

```
info =
(
 h'71A6C7C5BA9AD47FE72DA4DC359BF6B276D3515968711B9A911C71FC096AEE0E',
 "KEYSTREAM_2",
 h'',
 10
)
```

where last value is the length of PLAINTEXT_2.

```
info for KEYSTREAM_2 (CBOR Sequence) (48 bytes)
58 20 71 a6 c7 c5 ba 9a d4 7f e7 2d a4 dc 35 9b f6 b2 76 d3 51 59 68
71 1b 9a 91 1c 71 fc 09 6a ee 0e 6b 4b 45 59 53 54 52 45 41 4d 5f 32
40 0a

KEYSTREAM_2 (Raw Value) (10 bytes)
0a b8 c2 0e 84 9e 52 f5 9d fb
```

R calculates CIPHERTEXT_2 as XOR between PLAINTEXT_2 and KEYSTREAM_2:

```
CIPHERTEXT_2 (Raw Value) (10 bytes)
0f f0 4c 29 4f 4a c6 02 cf 78
```

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 25 54 91 b0 5a 39 89 ff 2d 3f fe a6 20 98 aa b5 7c 16 0f 29 4e
d9 48 01 8b 41 90 f7 d1 61 82 4e 0f f0 4c 29 4f 4a c6 02 cf 78 40
```

3.3. message_3

Since METHOD = 3, I authenticates using static DH.

I's static key pair for use with the EDHOC key exchange algorithm is based on the same curve as for the ephemeral keys, X25519:

```
I (Raw Value) (Initiator's private authentication key) (32 bytes)
cf c4 b6 ed 22 e7 00 a3 0d 5c 5b cd 61 f1 f0 20 49 de 23 54 62 33 48
93 d6 ff 9f 0c fe a3 fe 04

G_I (Raw Value) (Initiator's public authentication key) (32 bytes)
4a 49 d8 8c d5 d8 41 fa b7 ef 98 3e 91 1d 25 78 86 1f 95 88 4f 9f 5d
c4 2a 2e ed 33 de 79 ed 77
```

PRK_4x3m is derived as specified in Section 4.1.3 of [I-D.ietf-lake-edhoc]. Since I authenticates with static DH (METHOD = 3), PRK_4x3m is derived from G_IY using Extract() with the EDHOC hash algorithm:

```
PRK_4x3m = Extract(PRK_3e2m, G_IY) =
         = HMAC-SHA-256(PRK_3e2m, 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)
0a f4 2a d5 12 dc 3e 97 2b 3a c4 d4 7b a3 3f fc 21 f1 ae 6f 07 f2 f8
94 85 4a 5a 47 44 33 85 48

PRK_4x3m (Raw Value) (32 bytes)
b8 cc df 14 20 b5 b0 c8 2a 58 7e 7d 26 dd 7b 70 48 57 4c 3a 48 df 9f
6a 45 f7 21 c0 cf a4 b2 7c
```

The transcript hash TH_3 is calculated using the EDHOC hash algorithm:

TH_3 = H(TH_2, CIPHERTEXT_2)

```
Input to calculate TH_3 (CBOR Sequence) (45 bytes)
58 20 71 a6 c7 c5 ba 9a d4 7f e7 2d a4 dc 35 9b f6 b2 76 d3 51 59 68
71 1b 9a 91 1c 71 fc 09 6a ee 0e 4a 0f f0 4c 29 4f 4a c6 02 cf 78

TH_3 (Raw Value) (32 bytes)
a4 90 07 ce 54 76 2e 46 7c 4e 4a 44 69 2f 20 70 d3 e9 eb 00 f9 5a c2
62 9b 2b be f7 fb 24 a3 70

TH_3 (CBOR Data Item) (34 bytes)
58 20 a4 90 07 ce 54 76 2e 46 7c 4e 4a 44 69 2f 20 70 d3 e9 eb 00 f9
5a c2 62 9b 2b be f7 fb 24 a3 70
```

I constructs the remaining input needed to calculate MAC_3:

```
MAC_3 = EDHOC-KDF(PRK_4x3m, TH_3, "MAC_3",
        << ID_CRED_I, CRED_I, ? EAD_3 >>, mac_length_3)
```

CRED_I is identified by a 'kid' with integer value -10:

```
ID_CRED_I =
{
 4 : -10
}
```

ID_CRED_I (CBOR Data Item) (3 bytes) a1 04 29

CRED_I is an RPK encoded as a CCS:

```
{                                                 /CCS/
  2 : "42-50-31-FF-EF-37-32-39",                  /sub/
  8 : {                                           /cnf/
    1 : {                                         /COSE_Key/
      1 : 1,                                      /kty/
      2 : -10,                                    /kid/
     -1 : 4,                                      /crv/
     -2 : h'4A49D88CD5D841FAB7EF983E911D2578      /x/
             861F95884F9F5DC42A2EED33DE79ED77'
    }
  }
}
```

```
CRED_I (CBOR Data Item) (71 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 a4 01 01 02 29 20 04 21 58 20 4a 49 d8 8c d5 d8 41
fa b7 ef 98 3e 91 1d 25 78 86 1f 95 88 4f 9f 5d c4 2a 2e ed 33 de 79
ed 77
```

No external authorization data:

EAD_3 (CBOR Sequence) (0 bytes)

MAC_3 is computed through Expand() using the EDHOC hash algorithm, see Section 4.2 of [I-D.ietf-lake-edhoc]:

```
MAC_3 = HKDF-Expand(PRK_4x3m, info, mac_length_3)
```

Since METHOD = 3, mac_length_3 is given by the EDHOC MAC length.

info for MAC_3 is:

```
info =
(
 h'A49007CE54762E467C4E4A44692F2070D3E9EB00F95AC2629B2BBEF7FB24A370',
 "MAC_3",
 h'A10429A2027734322D35302D33312D46462D45462D33372D33322D333908A101
   A401010229200421582 04A49D88CD5D841FAB7EF983E911D2578861F95884F9F
   5DC42A2EED33DE79ED77',
 8
)
```

   where the last value is the EDHOC MAC length.

```
info for MAC_3 (CBOR Sequence) (117 bytes)
58 20 a4 90 07 ce 54 76 2e 46 7c 4e 4a 44 69 2f 20 70 d3 e9 eb 00 f9
5a c2 62 9b 2b be f7 fb 24 a3 70 65 4d 41 43 5f 33 58 4a a1 04 29 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 a4 01 01 02 29 20 04 21 58 20 4a 49 d8 8c d5 d8 41 fa
b7 ef 98 3e 91 1d 25 78 86 1f 95 88 4f 9f 5d c4 2a 2e ed 33 de 79 ed
77 08

MAC_3 (Raw Value) (8 bytes)
db 0b 8f 75 27 09 53 da

MAC_3 (CBOR Data Item) (9 bytes)
48 db 0b 8f 75 27 09 53 da
```

   Since METHOD = 3, Signature_or_MAC_3 is MAC_3:

```
Signature_or_MAC_3 (Raw Value) (8 bytes)
db 0b 8f 75 27 09 53 da

Signature_or_MAC_3 (CBOR Data Item) (9 bytes)
48 db 0b 8f 75 27 09 53 da
```

   I constructs the plaintext P_3:

```
P_3 =
(
 ID_CRED_I / bstr / int,
 Signature_or_MAC_3,
 ? EAD_3
)
```

   Since ID_CRED_I contains a single 'kid' parameter, only the int -10
   is included in the plaintext.

```
P_3 (CBOR Sequence) (10 bytes)
29 48 db 0b 8f 75 27 09 53 da
```

   I constructs the associated data for message_3:

```
A_3 =
(
 "Encrypt0",
 h'',
 TH_3
)
```

```
A_3 (CBOR Data Item) (45 bytes)
83 68 45 6e 63 72 79 70 74 30 40 58 20 a4 90 07 ce 54 76 2e 46 7c 4e
4a 44 69 2f 20 70 d3 e9 eb 00 f9 5a c2 62 9b 2b be f7 fb 24 a3 70
```

I constructs the input needed to derive the key K_3, see Section 4.2 of [I-D.ietf-lake-edhoc], using the EDHOC hash algorithm:

```
K_3 = EDHOC-KDF(PRK_3e2m, TH_3, "K_3", h'', length) =
            = HKDF-Expand(PRK_3e2m, info, length),
```

where length is the key length of EDHOC AEAD algorithm, and info for K_3 is:

```
info =
(
 h'A49007CE54762E467C4E4A44692F2070D3E9EB00F95AC2629B2BBEF7FB24A370',
 "K_3",
 h'',
 16
)
```

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

```
info for K_3 (CBOR Sequence) (40 bytes)
58 20 a4 90 07 ce 54 76 2e 46 7c 4e 4a 44 69 2f 20 70 d3 e9 eb 00 f9
5a c2 62 9b 2b be f7 fb 24 a3 70 63 4b 5f 33 40 10
```

```
K_3 (Raw Value) (16 bytes)
2a 30 e4 f6 bc 55 8d 0e 7a 8c 63 ee 7b b5 45 7f
```

I constructs the input needed to derive the nonce IV_3, see Section 4.2 of [I-D.ietf-lake-edhoc], using the EDHOC hash algorithm:

```
IV_3 = EDHOC-KDF(PRK_3e2m, TH_3, "IV_3", h'', length) =
       = HKDF-Expand(PRK_3e2m, info, length),
```

where length is the nonce length of EDHOC AEAD algorithm, and info for IV_3 is:

```
info =
(
 h'A49007CE54762E467C4E4A44692F2070D3E9EB00F95AC2629B2BBEF7FB24A370',
 "IV_3",
 h'',
 13
)
```

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

```
info for IV_3 (CBOR Sequence) (41 bytes)
58 20 a4 90 07 ce 54 76 2e 46 7c 4e 4a 44 69 2f 20 70 d3 e9 eb 00 f9
5a c2 62 9b 2b be f7 fb 24 a3 70 64 49 56 5f 33 40 0d

IV_3 (Raw Value) (13 bytes)
b3 8f b6 31 e3 44 a8 10 52 56 32 ed f8
```

   I calculates CIPHERTEXT_3 as 'ciphertext' of COSE_Encrypt0 applied using the EDHOC AEAD algorithm with plaintext P_3, additional data A_3, key K_3 and nonce IV_3.

```
CIPHERTEXT_3 (Raw Value) (18 bytes)
be 01 46 c1 36 ac 2e ff d4 53 a7 5e fa 90 89 6f 65 3b
```

   message_3 is the CBOR bstr encoding of CIPHERTEXT_3:

```
message_3 (CBOR Sequence) (19 bytes)
52 be 01 46 c1 36 ac 2e ff d4 53 a7 5e fa 90 89 6f 65 3b
```

   The transcript hash TH_4 is calculated using the EDHOC hash algorithm:

   TH_4 = H(TH_3, CIPHERTEXT_3)

```
Input to calculate TH_4 (CBOR Sequence) (53 bytes)
58 20 a4 90 07 ce 54 76 2e 46 7c 4e 4a 44 69 2f 20 70 d3 e9 eb 00 f9
5a c2 62 9b 2b be f7 fb 24 a3 70 52 be 01 46 c1 36 ac 2e ff d4 53 a7
5e fa 90 89 6f 65 3b

TH_4 (Raw Value) (32 bytes)
4b 9a dd 2a 9e eb 88 49 71 6c 79 68 78 4f 55 40 dd 64 a3 bb 07 f8 d0
00 ad ce 88 b6 30 d8 84 eb

TH_4 (CBOR Data Item) (34 bytes)
58 20 4b 9a dd 2a 9e eb 88 49 71 6c 79 68 78 4f 55 40 dd 64 a3 bb 07
f8 d0 00 ad ce 88 b6 30 d8 84 eb
```

3.4. message_4

   No external authorization data:

EAD_4 (CBOR Sequence) (0 bytes)

R constructs the plaintext P_4:

```
P_4 =
(
 ? EAD_4
)
```

P_4 (CBOR Sequence) (0 bytes)

R constructs the associated data for message_4:

```
A_4 =
(
 "Encrypt0",
 h'',
 TH_4
)
```

```
A_4 (CBOR Data Item) (45 bytes)
83 68 45 6e 63 72 79 70 74 30 40 58 20 4b 9a dd 2a 9e eb 88 49 71 6c
79 68 78 4f 55 40 dd 64 a3 bb 07 f8 d0 00 ad ce 88 b6 30 d8 84 eb
```

R constructs the input needed to derive the EDHOC message_4 key, see Section 4.2 of [I-D.ietf-lake-edhoc], using the EDHOC hash algorithm:

```
K_4 = EDHOC-Exporter("EDHOC_K_4", h'', length)
      = EDHOC-KDF(PRK_4x3m, TH_4, "EDHOC_K_4", h'', length)
      = HKDF-Expand(PRK_4x3m, info, length)
```

where length is the key length of the EDHOC AEAD algorithm, and info for EDHOC_K_4 is:

```
info =
(
 h'4B9ADD2A9EEB8849716C7968784F5540DD64A3BB07F8D000ADCE88B630D884EB',
 "EDHOC_K_4",
 h'',
 16
)
```

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

```
info for K_4 (CBOR Sequence) (46 bytes)
58 20 4b 9a dd 2a 9e eb 88 49 71 6c 79 68 78 4f 55 40 dd 64 a3 bb 07
f8 d0 00 ad ce 88 b6 30 d8 84 eb 69 45 44 48 4f 43 5f 4b 5f 34 40 10
```

```
K_4 (Raw Value) (16 bytes)
55 b5 7d 59 a8 26 f4 56 38 86 9b 75 07 0b 11 17
```

R constructs the input needed to derive the EDHOC message_4 nonce, see Section 4.2 of [I-D.ietf-lake-edhoc], using the EDHOC hash algorithm:

```
IV_4 =
= EDHOC-Exporter( "EDHOC_IV_4", h'', length )
= EDHOC-KDF(PRK_4x3m, TH_4, "EDHOC_IV_4", h'', length)
= HKDF-Expand(PRK_4x3m, info, length)
```

where length is the nonce length of EDHOC AEAD algorithm, and info for EDHOC_IV_4 is:

```
info =
(
 h'4B9ADD2A9EEB8849716C7968784F5540DD64A3BB07F8D000ADCE88B630D884EB',
 "EDHOC_IV_4",
 h'',
 13
)
```

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

```
info for IV_4 (CBOR Sequence) (47 bytes)
58 20 4b 9a dd 2a 9e eb 88 49 71 6c 79 68 78 4f 55 40 dd 64 a3 bb 07
f8 d0 00 ad ce 88 b6 30 d8 84 eb 6a 45 44 48 4f 43 5f 49 56 5f 34 40
0d

IV_4 (Raw Value) (13 bytes)
20 7a 4e fc 25 a6 58 96 45 11 f1 63 76
```

R calculates CIPHERTEXT_4 as 'ciphertext' of COSE_Encrypt0 applied using the EDHOC AEAD algorithm with plaintext P_4, additional data A_4, key K_4 and nonce IV_4.

```
CIPHERTEXT_4 (8 bytes)
e9 e6 c8 b6 37 6d b0 b1
```

message_4 is the CBOR bstr encoding of CIPHERTEXT_4:

```
message_4 (CBOR Sequence) (9 bytes)
48 e9 e6 c8 b6 37 6d b0 b1
```

3.5. OSCORE Parameters

The derivation of OSCORE parameters is specified in Appendix A.2 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 A.1of [I-D.ietf-lake-edhoc].

C_R is mapped to the Recipient ID of the server, i.e., the Sender ID of the client. Since C_R is byte valued it the OSCORE Sender/ Recipient ID equals the byte string (in this case the empty byte string).

```
Client's OSCORE Sender ID (Raw Value) (0 bytes)
```

C_I is mapped to the Recipient ID of the client, i.e., the Sender ID of the server. Since C_I is a numeric, it is converted to a byte string equal to its CBOR encoded form.

```
Server's OSCORE Sender ID (Raw Value) (1 bytes)
0c
```

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

```
OSCORE Master Secret =
= EDHOC-Exporter("OSCORE_Master_Secret", h'', key_length)
= EDHOC-KDF(PRK_4x3m, TH_4, "OSCORE_Master_Secret", h'', key_length)
= HKDF-Expand(PRK_4x3m, info, key_length)
```

where key_length is by default the key length of the Application AEAD algorithm, and info for the OSCORE Master Secret is:

```
info =
(
 h'4B9ADD2A9EEB8849716C7968784F5540DD64A3BB07F8D000ADCE88B630D884EB',
 "OSCORE_Master_Secret",
 h'',
 16
)
```

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

```
info for OSCORE Master Secret (CBOR Sequence) (57 bytes)
58 20 4b 9a dd 2a 9e eb 88 49 71 6c 79 68 78 4f 55 40 dd 64 a3 bb 07
f8 d0 00 ad ce 88 b6 30 d8 84 eb 74 4f 53 43 4f 52 45 5f 4d 61 73 74
65 72 5f 53 65 63 72 65 74 40 10


OSCORE Master Secret (Raw Value) (16 bytes)
c0 53 01 37 6c e9 5f 67 c4 14 d8 bb 5f 0f db 5e
```

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("OSCORE_Master_Salt", h'', salt_length)
= EDHOC-KDF(PRK_4x3m, TH_4, "OSCORE_Master_Salt", h'', salt_length)
= HKDF-Expand(PRK_4x3m, info, salt_length)
```

where salt_length is the length of the OSCORE Master Salt, and info for the OSCORE Master Salt is:

```
info =
(
 h'4B9ADD2A9EEB8849716C7968784F5540DD64A3BB07F8D000ADCE88B630D884EB',
 "OSCORE_Master_Salt",
 h'',
 8
)
```

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

```
info for OSCORE Master Salt (CBOR Sequence) (55 bytes)
58 20 4b 9a dd 2a 9e eb 88 49 71 6c 79 68 78 4f 55 40 dd 64 a3 bb 07
f8 d0 00 ad ce 88 b6 30 d8 84 eb 72 4f 53 43 4f 52 45 5f 4d 61 73 74
65 72 5f 53 61 6c 74 40 08

OSCORE Master Salt (Raw Value) (8 bytes)
74 01 b4 6f a8 2f 66 31
```

3.6. Key Update

Key update is defined in Section 4.4 of [I-D.ietf-lake-edhoc]:

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

KeyUpdate Nonce (Raw Value) (16 bytes)
d4 91 a2 04 ca a6 b8 02 54 c4 71 e0 de ee d1 60
```

PRK_4x3m after KeyUpdate (Raw Value) (32 bytes)
82 09 6e 3a e6 3d 93 c7 b6 f8 8b 7c 1b 5e 63 f4 9f 74 c8 0e f3 14 42
51 9f fb 20 e2 f8 87 3e b1

The OSCORE Master Secret is derived with the updated PRK_4x3m:

OSCORE Master Secret = HKDF-Expand(PRK_4x3m, info, key_length)

where info and key_length are unchanged.

OSCORE Master Secret after KeyUpdate (Raw Value) (16 bytes)
a5 15 23 1d 9e c5 88 74 82 22 6b f9 e0 da 05 ce

The OSCORE Master Salt is derived with the updated PRK_4x3m:

OSCORE Master Salt = HKDF-Expand(PRK_4x3m, info, salt_length)

where info and salt_length are unchanged.

OSCORE Master Salt after KeyUpdate (Raw Value) (8 bytes)
50 57 e5 92 ed 8b 11 28

4. Authentication with signatures, X.509 identified by 'x5t'

In this example the Initiator (I) and Responder (R) are authenticated with digital signatures (METHOD = 0). The public keys are represented with dummy X.509 certificates identified by the COSE header parameter 'x5t'.

4.1. message_1

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

METHOD (CBOR Data Item) (1 bytes)
00

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

SUITES_I (CBOR Data Item) (1 bytes)
00

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

X (Raw Value) (Initiator's ephemeral private key) (32 bytes)
b0 26 b1 68 42 9b 21 3d 6b 42 1d f6 ab d0 64 1c d6 6d ca 2e e7 fd 59
77 10 4b b2 38 18 2e 5e a6

```
G_X (Raw Value) (Initiator's ephemeral public key) (32 bytes)
e3 1e c1 5e e8 03 94 27 df c4 72 7e f1 7e 2e 0e 69 c5 44 37 f3 c5 82
80 19 ef 0a 63 88 c1 25 52

G_X (CBOR Data Item) (Initiator's ephemeral public key) (34 bytes)
58 20 e3 1e c1 5e e8 03 94 27 df c4 72 7e f1 7e 2e 0e 69 c5 44 37 f3
c5 82 80 19 ef 0a 63 88 c1 25 52
```

I selects its connection identifier C_I to be the int 14:

```
C_I (Raw Value) (Connection identifier chosen by I) (int)
14

C_I (CBOR Data Item) (Connection identifier chosen by I) (1 bytes)
0e
```

No external authorization data:

EAD_1 (CBOR Sequence) (0 bytes)

I constructs message_1:

```
message_1 =
(
 0,
 0,
 h'E31EC15EE8039427DFC4727EF17E2E0E69C54437F3C5828019EF0A6388C12552',
 14
)

message_1 (CBOR Sequence) (37 bytes)
00 00 58 20 e3 1e c1 5e e8 03 94 27 df c4 72 7e f1 7e 2e 0e 69 c5 44
37 f3 c5 82 80 19 ef 0a 63 88 c1 25 52 0e
```

4.2. message_2

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

```
Y (Raw Value) (Responder's ephemeral private key) (32 bytes)
db 06 84 a8 12 54 66 41 3e 59 8d c2 67 73 7f 5f ef 0c 5a a2 29 fa a1
55 43 9f 60 08 5f d2 53 6d

G_Y (Raw Value) (Responder's ephemeral public key) (32 bytes)
e1 73 90 96 c5 c9 58 2c 12 98 91 81 66 d6 95 48 c7 8f 74 97 b2 58 c0
85 6a a2 01 98 93 a3 94 25

G_Y (CBOR Data Item) (Responder's ephemeral public key) (34 bytes)
58 20 e1 73 90 96 c5 c9 58 2c 12 98 91 81 66 d6 95 48 c7 8f 74 97 b2
58 c0 85 6a a2 01 98 93 a3 94 25
```

PRK_2e is specified in Section 4.1.1 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)
0b eb 98 d8 8f 49 67 7c 17 47 88 f8 87 bd cc d2 28 a1 88 39 2c cd 10
12 bd 31 70 d7 c8 85 65 66
```

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 the zero-length byte string:

salt (Raw Value) (0 bytes)

```
PRK_2e (Raw Value) (32 bytes)
4e 57 dc e2 58 75 77 c4 34 69 7c 03 93 5c c6 a2 82 16 5a 88 76 05 11
fc 70 a8 c0 02 20 a5 ba 1a
```

Since METHOD = 0, R authenticates using signatures with the EDHOC signature algorithm. R's signature key pair using Ed25519 is (note that Ed448 would also be compatible with EdDSA):

```
SK_R (Raw Value) (Responders's private authentication key) (32 bytes)
bc 4d 4f 98 82 61 22 33 b4 02 db 75 e6 c4 cf 30 32 a7 0a 0d 2e 3e e6
d0 1b 11 dd de 5f 41 9c fc

PK_R (Raw Value) (Responders's public authentication key) (32 bytes)
27 ee f2 b0 8a 6f 49 6f ae da a6 c7 f9 ec 6a e3 b9 d5 24 24 58 0d 52
e4 9d a6 93 5e df 53 cd c5
```

PRK_3e2m is specified in Section 4.1.2 of [I-D.ietf-lake-edhoc].

Since R authenticates with signatures PRK_3e2m = PRK_2e.

```
PRK_3e2m (Raw Value) (32 bytes)
4e 57 dc e2 58 75 77 c4 34 69 7c 03 93 5c c6 a2 82 16 5a 88 76 05 11
fc 70 a8 c0 02 20 a5 ba 1a
```

R selects its connection identifier C_R to be the int -19

```
C_R (Raw Value) (Connection identifier chosen by R) (int)
-19

C_R (CBOR Data Item) (Connection identifier chosen by R) (1 bytes)
32
```

The transcript hash TH_2 is calculated using the EDHOC hash algorithm:

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

```
H(message_1) (Raw Value) (32 bytes)
ce ba 8d 4d a2 80 b1 61 c8 5a 19 47 81 a9 31 88 35 41 50 b4 9c 4f 93
2e 4a a0 8f f3 ed 11 04 65

H(message_1) (CBOR Data Item) (34 bytes)
58 20 ce ba 8d 4d a2 80 b1 61 c8 5a 19 47 81 a9 31 88 35 41 50 b4 9c
4f 93 2e 4a a0 8f f3 ed 11 04 65
```

The input to calculate TH_2 is the CBOR sequence:

H(message_1), G_Y, C_R

```
Input to calculate TH_2 (CBOR Sequence) (69 bytes)
58 20 ce ba 8d 4d a2 80 b1 61 c8 5a 19 47 81 a9 31 88 35 41 50 b4 9c
4f 93 2e 4a a0 8f f3 ed 11 04 65 58 20 e1 73 90 96 c5 c9 58 2c 12 98
91 81 66 d6 95 48 c7 8f 74 97 b2 58 c0 85 6a a2 01 98 93 a3 94 25 32

TH_2 (Raw Value) (32 bytes)
07 82 db b6 87 c3 02 88 a3 0b 70 6b 07 4b ed 78 95 74 57 3f 24 44 3e
91 83 3d 68 cd dd 7f 9b 39

TH_2 (CBOR Data Item) (34 bytes)
58 20 07 82 db b6 87 c3 02 88 a3 0b 70 6b 07 4b ed 78 95 74 57 3f 24
44 3e 91 83 3d 68 cd dd 7f 9b 39
```

R constructs the remaining input needed to calculate MAC_2:

MAC_2 = EDHOC-KDF(PRK_3e2m, TH_2, "MAC_2", << ID_CRED_R, CRED_R, ? EAD_2 >>, mac_length_2)

CRED_R is identified by a 64-bit hash:

```
ID_CRED_R =
{
  34 : [-15, h'60780E9451BDC43C']
}
```

where the COSE header value 34 ('x5t') indicates a hash of an X.509 certficate, 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 60 78 0e 94 51 bd c4 3c

CRED_R is a byte string acting as a dummy X.509 certificate:

```
CRED_R (CBOR Data Item) (113 bytes)
58 6f 00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 14
15 16 17 18 19 1a 1b 1c 1d 1e 1f 20 21 22 23 24 25 26 27 28 29 2a 2b
2c 2d 2e 2f 30 31 32 33 34 35 36 37 38 39 3a 3b 3c 3d 3e 3f 40 41 42
43 44 45 46 47 48 49 4a 4b 4c 4d 4e 4f 50 51 52 53 54 55 56 57 58 59
5a 5b 5c 5d 5e 5f 60 61 62 63 64 65 66 67 68 69 6a 6b 6c 6d 6e
```

No external authorization data:

```
EAD_2 (CBOR Sequence) (0 bytes)
```

MAC_2 is computed through Expand() using the EDHOC hash algorithm, Section 4.2 of [I-D.ietf-lake-edhoc]:

MAC_2 = HKDF-Expand(PRK_3e2m, info, mac_length_2)

Since METHOD = 0, mac_length_2 is given by the EDHOC hash algorithm.

info for MAC_2 is:

```
info =
(
 h'0782DBB687C30288A30B706B074BED789574573F24443E91833D68CDDD7F9B39',
 "MAC_2",
 h'A11822822E4860780E9451BDC43C586F000102030405060708090A0B0C0D0E0F10
   111213141516171819 1A1B1C1D1E1F202122232425262728292A2B2C2D2E2F3031
   323334353637383 93A3B3C3D3E3F404142434445464748494A4B4C4D4E4F505152
   535455565758595A5B5C5D5E5F606162636465666768696A6B6C6D6E',
 32
)
```

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

```
info for MAC_2 (CBOR Sequence) (171 bytes)
58 20 07 82 db b6 87 c3 02 88 a3 0b 70 6b 07 4b ed 78 95 74 57 3f 24
44 3e 91 83 3d 68 cd dd 7f 9b 39 65 4d 41 43 5f 32 58 7f a1 18 22 82
2e 48 60 78 0e 94 51 bd c4 3c 58 6f 00 01 02 03 04 05 06 07 08 09 0a
0b 0c 0d 0e 0f 10 11 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f 20 21
22 23 24 25 26 27 28 29 2a 2b 2c 2d 2e 2f 30 31 32 33 34 35 36 37 38
39 3a 3b 3c 3d 3e 3f 40 41 42 43 44 45 46 47 48 49 4a 4b 4c 4d 4e 4f
50 51 52 53 54 55 56 57 58 59 5a 5b 5c 5d 5e 5f 60 61 62 63 64 65 66
67 68 69 6a 6b 6c 6d 6e 18 20

MAC_2 (Raw Value) (32 bytes)
27 c8 f1 e4 a7 af f2 a0 f0 bc 0f 91 83 93 ee f1 8b 69 0c 4d 4c 3d 81
bd fe 22 95 42 40 bc c4 cc

MAC_2 (CBOR Data Item) (34 bytes)
58 20 27 c8 f1 e4 a7 af f2 a0 f0 bc 0f 91 83 93 ee f1 8b 69 0c 4d 4c
3d 81 bd fe 22 95 42 40 bc c4 cc
```

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'A11822822E4860780E9451BDC43C',
 h'58200782DBB687C30288A30B706B074BED789574573F24443E91833D68CDDD7F
   9B39586F000102030405060708090A0B0C0D0E0F101112131415161718191A1B
   1C1D1E1F202122232425262728292A2B2C2D2E2F303132333435363738393A3B
   3C3D3E3F404142434445464748494A4B4C4D4E4F505152535455565758595A5B
   5C5D5E5F606162636465666768696A6B6C6D6E',
 h'27C8F1E4A7AFF2A0F0BC0F918393EEF18B690C4D4C3D81BDFE22954240BCC4CC'
]
```

```
Message to be signed 2 (CBOR Data Item) (210 bytes)
84 6a 53 69 67 6e 61 74 75 72 65 31 4e a1 18 22 82 2e 48 60 78 0e 94
51 bd c4 3c 58 93 58 20 07 82 db b6 87 c3 02 88 a3 0b 70 6b 07 4b ed
78 95 74 57 3f 24 44 3e 91 83 3d 68 cd dd 7f 9b 39 58 6f 00 01 02 03
04 05 06 07 08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 14 15 16 17 18 19 1a
1b 1c 1d 1e 1f 20 21 22 23 24 25 26 27 28 29 2a 2b 2c 2d 2e 2f 30 31
32 33 34 35 36 37 38 39 3a 3b 3c 3d 3e 3f 40 41 42 43 44 45 46 47 48
49 4a 4b 4c 4d 4e 4f 50 51 52 53 54 55 56 57 58 59 5a 5b 5c 5d 5e 5f
60 61 62 63 64 65 66 67 68 69 6a 6b 6c 6d 6e 58 20 27 c8 f1 e4 a7 af
f2 a0 f0 bc 0f 91 83 93 ee f1 8b 69 0c 4d 4c 3d 81 bd fe 22 95 42 40
bc c4 cc
```

R signs using the private authentication key SK_R

```
Signature_or_MAC_2 (Raw Value) (64 bytes)
3c e5 20 75 db 55 89 2d f1 25 8f a6 9e 86 ab 5b 59 33 ea dc 07 ea 82
41 1f 17 9a 5f de f1 c9 43 23 63 f6 58 f9 a2 04 fa 81 54 d1 4f fd 87
b5 01 0c 4f d0 a0 c7 7e 2a ca 77 5f 67 cb 5e 8b be 08
```

```
Signature_or_MAC_2 (CBOR Data Item) (66 bytes)
58 40 3c e5 20 75 db 55 89 2d f1 25 8f a6 9e 86 ab 5b 59 33 ea dc 07
ea 82 41 1f 17 9a 5f de f1 c9 43 23 63 f6 58 f9 a2 04 fa 81 54 d1 4f
fd 87 b5 01 0c 4f d0 a0 c7 7e 2a ca 77 5f 67 cb 5e 8b be 08
```

R constructs the plaintext:

```
PLAINTEXT_2 =
(
 ID_CRED_R / bstr / int,
 Signature_or_MAC_2,
 ? EAD_2
)
```

```
PLAINTEXT_2 (CBOR Sequence) (80 bytes)
a1 18 22 82 2e 48 60 78 0e 94 51 bd c4 3c 58 40 3c e5 20 75 db 55 89
2d f1 25 8f a6 9e 86 ab 5b 59 33 ea dc 07 ea 82 41 1f 17 9a 5f de f1
c9 43 23 63 f6 58 f9 a2 04 fa 81 54 d1 4f fd 87 b5 01 0c 4f d0 a0 c7
7e 2a ca 77 5f 67 cb 5e 8b be 08
```

The input needed to calculate KEYSTREAM_2 is defined in Section 4.2 of [I-D.ietf-lake-edhoc], using Expand() with the EDHOC hash algorithm:

```
KEYSTREAM_2 = EDHOC-KDF(PRK_2e, TH_2, "KEYSTREAM_2", h'', length) =
            = HKDF-Expand(PRK_2e, info, length)
```

where length is the length of PLAINTEXT_2, and info for KEYSTREAM_2 is:

```
info =
(
 h'0782DBB687C30288A30B706B074BED789574573F24443E91833D68CDDD7F9B39',
 "KEYSTREAM_2",
 h'',
 80
)
```

where the last value is the length of PLAINTEXT_2.

```
info for KEYSTREAM_2 (CBOR Sequence) (49 bytes)
58 20 07 82 db b6 87 c3 02 88 a3 0b 70 6b 07 4b ed 78 95 74 57 3f 24
44 3e 91 83 3d 68 cd dd 7f 9b 39 6b 4b 45 59 53 54 52 45 41 4d 5f 32
40 18 50
```

```
KEYSTREAM_2 (Raw Value) (80 bytes)
c8 13 ff 19 3b c0 31 40 47 99 6a 37 03 09 ba ed 45 f7 f5 f8 d5 6c 1c
df 44 6b 01 c5 77 8d 68 9f 7f 13 da 50 17 ba 0f 4e 5f df 6e d0 59 55
cd 8c e4 ec 43 7a 22 fa 8e e8 72 8c 36 2b cb 7b 93 a9 11 e1 67 95 04
31 c1 d5 05 0b da 69 e9 5b aa fb
```

R calculates CIPHERTEXT_2 as XOR between PLAINTEXT_2 and KEYSTREAM_2:

```
CIPHERTEXT_2 (Raw Value) (80 bytes)
69 0b dd 9b 15 88 51 38 49 0d 3b 8a c7 35 e2 ad 79 12 d5 8d 0e 39 95
f2 b5 4e 8e 63 e9 0b c3 c4 26 20 30 8c 10 50 8d 0f 40 c8 f4 8f 87 a4
04 cf c7 8f b5 22 db 58 8a 12 f3 d8 e7 64 36 fc 26 a8 1d ae b7 35 c3
4f eb 1f 72 54 bd a2 b7 d0 14 f3
```

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) (115 bytes)
58 70 e1 73 90 96 c5 c9 58 2c 12 98 91 81 66 d6 95 48 c7 8f 74 97 b2
58 c0 85 6a a2 01 98 93 a3 94 25 69 0b dd 9b 15 88 51 38 49 0d 3b 8a
c7 35 e2 ad 79 12 d5 8d 0e 39 95 f2 b5 4e 8e 63 e9 0b c3 c4 26 20 30
8c 10 50 8d 0f 40 c8 f4 8f 87 a4 04 cf c7 8f b5 22 db 58 8a 12 f3 d8
e7 64 36 fc 26 a8 1d ae b7 35 c3 4f eb 1f 72 54 bd a2 b7 d0 14 f3 32
```

4.3. message_3

Since METHOD = 0, I authenticates using signatures with the EDHOC signature algorithm. I's signature key pair using Ed25519 is (note that Ed448 would also be compatible with EdDSA):

```
SK_I (Raw Value) (Initiator's private authentication key) (32 bytes)
36 6a 58 59 a4 cd 65 cf ae af 05 66 c9 fc 7e 1a 93 30 6f de c1 77 63
e0 58 13 a7 0f 21 ff 59 db

PK_I (Raw Value) (Responders's public authentication key) (32 bytes)
ec 2c 2e b6 cd d9 57 82 a8 cd 0b 2e 9c 44 27 07 74 dc bd 31 bf be 23
13 ce 80 13 2e 8a 26 1c 04
```

PRK_4x3m is specified in Section 4.1.3 of [I-D.ietf-lake-edhoc].

Since R authenticates with signatures PRK_4x3m = PRK_3e2m.

```
PRK_4x3m (Raw Value) (32 bytes)
4e 57 dc e2 58 75 77 c4 34 69 7c 03 93 5c c6 a2 82 16 5a 88 76 05 11
fc 70 a8 c0 02 20 a5 ba 1a
```

The transcript hash TH_3 is calculated using the EDHOC hash algorithm:

TH_3 = H(TH_2, CIPHERTEXT_2)

```
Input to calculate TH_3 (CBOR Sequence) (116 bytes)
58 20 07 82 db b6 87 c3 02 88 a3 0b 70 6b 07 4b ed 78 95 74 57 3f 24
44 3e 91 83 3d 68 cd dd 7f 9b 39 58 50 69 0b dd 9b 15 88 51 38 49 0d
3b 8a c7 35 e2 ad 79 12 d5 8d 0e 39 95 f2 b5 4e 8e 63 e9 0b c3 c4 26
20 30 8c 10 50 8d 0f 40 c8 f4 8f 87 a4 04 cf c7 8f b5 22 db 58 8a 12
f3 d8 e7 64 36 fc 26 a8 1d ae b7 35 c3 4f eb 1f 72 54 bd a2 b7 d0 14
f3

TH_3 (Raw Value) (32 bytes)
23 ce 42 96 fc 64 ab 04 8a 59 3b 67 11 e4 82 20 11 bb 58 d8 5d 37 98
b0 81 a9 bd 12 a3 31 7a 82

TH_3 (CBOR Data Item) (34 bytes)
58 20 23 ce 42 96 fc 64 ab 04 8a 59 3b 67 11 e4 82 20 11 bb 58 d8 5d
37 98 b0 81 a9 bd 12 a3 31 7a 82
```

I constructs the remaining input needed to calculate MAC_3:

```
MAC_3 = EDHOC-KDF(PRK_4x3m, TH_3, "MAC_3",
        << ID_CRED_I, CRED_I, ? EAD_3 >>, mac_length_3)
```

CRED_I is identified by a 64-bit hash:

```
ID_CRED_I =
{
  34 : [-15, h'81D45BE06329D63A']
}
```

where the COSE header value 34 ('x5t') indicates a hash of an X.509 certficate, 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 81 d4 5b e0 63 29 d6 3a
```

CRED_I is a byte string acting as a dummy X.509 certificate:

```
CRED_I (CBOR Data Item) (139 bytes)
58 89 00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 14
15 16 17 18 19 1a 1b 1c 1d 1e 1f 20 21 22 23 24 25 26 27 28 29 2a 2b
2c 2d 2e 2f 30 31 32 33 34 35 36 37 38 39 3a 3b 3c 3d 3e 3f 40 41 42
43 44 45 46 47 48 49 4a 4b 4c 4d 4e 4f 50 51 52 53 54 55 56 57 58 59
5a 5b 5c 5d 5e 5f 60 61 62 63 64 65 66 67 68 69 6a 6b 6c 6d 6e 6f 70
71 72 73 74 75 76 77 78 79 7a 7b 7c 7d 7e 7f 80 81 82 83 84 85 86 87
88
```

No external authorization data:

```
EAD_3 (CBOR Sequence) (0 bytes)
```

MAC_3 is computed through Expand() using the EDHOC hash algorithm, see Section 4.2 of [I-D.ietf-lake-edhoc]:

```
MAC_3 = HKDF-Expand(PRK_4x3m, info, mac_length_3)
```

Since METHOD = 0, mac_length_3 is given by the EDHOC hash algorithm.

info for MAC_3 is:

```
info =
(
 h'23CE4296FC64AB048A593B6711E4822011BB58D85D3798B081A9BD12A3317A82',
 "MAC_3",
 h'A11822822E4881D45BE06329D63A588900010203040506070809 0A0B0C0D0E0F
   101112131415161718191A1B1C1D1E1F202122232425262728292A2B2C2D2E2F
   303132333435363738393A3B3C3D3E3F404142434445464748494A4B4C4D4E4F
   505152535455565758595A5B5C5D5E5F606162636465666768696A6B6C6D6E6F
   707172737475767778797A7B7C7D7E7F808182838485868788',
 32
)
```

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

```
info for MAC_3 (CBOR Sequence) (197 bytes)
58 20 23 ce 42 96 fc 64 ab 04 8a 59 3b 67 11 e4 82 20 11 bb 58 d8 5d
37 98 b0 81 a9 bd 12 a3 31 7a 82 65 4d 41 43 5f 33 58 99 a1 18 22 82
2e 48 81 d4 5b e0 63 29 d6 3a 58 89 00 01 02 03 04 05 06 07 08 09 0a
0b 0c 0d 0e 0f 10 11 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f 20 21
22 23 24 25 26 27 28 29 2a 2b 2c 2d 2e 2f 30 31 32 33 34 35 36 37 38
39 3a 3b 3c 3d 3e 3f 40 41 42 43 44 45 46 47 48 49 4a 4b 4c 4d 4e 4f
50 51 52 53 54 55 56 57 58 59 5a 5b 5c 5d 5e 5f 60 61 62 63 64 65 66
67 68 69 6a 6b 6c 6d 6e 6f 70 71 72 73 74 75 76 77 78 79 7a 7b 7c 7d
7e 7f 80 81 82 83 84 85 86 87 88 18 20

MAC_3 (Raw Value) (32 bytes)
fc 86 e7 d4 f1 8b 34 8c 29 7c 2f a3 eb 19 52 9a cc 3e 0a 4c b1 ba 99
b6 9d 16 aa b1 9d 33 3c 12

MAC_3 (CBOR Data Item) (34 bytes)
58 20 fc 86 e7 d4 f1 8b 34 8c 29 7c 2f a3 eb 19 52 9a cc 3e 0a 4c b1
ba 99 b6 9d 16 aa b1 9d 33 3c 12
```

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'A11822822E4881D45BE06329D63A',
 h'58205AA25B46397C2F145EB792ED0D17EA2B078C73E4EE148780C3C2E7341372
   CBAD5889000102030405060708090A0B0C0D0E0F101112131415161718191A1B
   1C1D1E1F202122232425262728292A2B2C2D2E2F303132333435363738393A3B
   3C3D3E3F404142434445464748494A4B4C4D4E4F505152535455565758595A5B
   5C5D5E5F606162636465666768696A6B6C6D6E6F707172737475767778797A7B
   7C7D7E7F808182838485868788',
 h'FC86E7D4F18B348C297C2FA3EB19529ACC3E0A4CB1BA99B69D16AAB19D333C12'
]
```

Message to be signed 3 (CBOR Data Item) (236 bytes)

```
84 6a 53 69 67 6e 61 74 75 72 65 31 4e a1 18 22 82 2e 48 81 d4 5b e0
63 29 d6 3a 58 ad 58 20 23 ce 42 96 fc 64 ab 04 8a 59 3b 67 11 e4 82
20 11 bb 58 d8 5d 37 98 b0 81 a9 bd 12 a3 31 7a 82 58 89 00 01 02 03
04 05 06 07 08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 14 15 16 17 18 19 1a
1b 1c 1d 1e 1f 20 21 22 23 24 25 26 27 28 29 2a 2b 2c 2d 2e 2f 30 31
32 33 34 35 36 37 38 39 3a 3b 3c 3d 3e 3f 40 41 42 43 44 45 46 47 48
49 4a 4b 4c 4d 4e 4f 50 51 52 53 54 55 56 57 58 59 5a 5b 5c 5d 5e 5f
60 61 62 63 64 65 66 67 68 69 6a 6b 6c 6d 6e 6f 70 71 72 73 74 75 76
77 78 79 7a 7b 7c 7d 7e 7f 80 81 82 83 84 85 86 87 88 58 20 fc 86 e7
d4 f1 8b 34 8c 29 7c 2f a3 eb 19 52 9a cc 3e 0a 4c b1 ba 99 b6 9d 16
aa b1 9d 33 3c 12
```

R signs using the private authentication key SK_R:

Signature_or_MAC_3 (Raw Value) (64 bytes)

```
3d d3 74 07 a1 d9 f1 2a 5b a6 4d f0 5f a0 d9 46 25 bf 74 0c 29 5f e1
88 58 d6 8e 04 5c 84 90 27 54 88 03 56 3e de 8c 5b 39 11 4f 13 fe 29
78 8a 83 b7 42 28 8e ab 8a 94 52 2c b1 d3 03 f2 62 04
```

Signature_or_MAC_3 (CBOR Data Item) (66 bytes)

```
58 40 3d d3 74 07 a1 d9 f1 2a 5b a6 4d f0 5f a0 d9 46 25 bf 74 0c 29
5f e1 88 58 d6 8e 04 5c 84 90 27 54 88 03 56 3e de 8c 5b 39 11 4f 13
fe 29 78 8a 83 b7 42 28 8e ab 8a 94 52 2c b1 d3 03 f2 62 04
```

R constructs the plaintext:

```
P_3 =
(
 ID_CRED_I / bstr / int,
 Signature_or_MAC_3,
 ? EAD_3
)
```

```
P_3 (CBOR Sequence) (80 bytes)
a1 18 22 82 2e 48 81 d4 5b e0 63 29 d6 3a 58 40 3d d3 74 07 a1 d9 f1
2a 5b a6 4d f0 5f a0 d9 46 25 bf 74 0c 29 5f e1 88 58 d6 8e 04 5c 84
90 27 54 88 03 56 3e de 8c 5b 39 11 4f 13 fe 29 78 8a 83 b7 42 28 8e
ab 8a 94 52 2c b1 d3 03 f2 62 04
```

I constructs the associated data for message_3:

```
A_3 =
(
 "Encrypt0",
 h'',
 TH_3
)

A_3 (CBOR Data Item) (45 bytes)
83 68 45 6e 63 72 79 70 74 30 40 58 20 23 ce 42 96 fc 64 ab 04 8a 59
3b 67 11 e4 82 20 11 bb 58 d8 5d 37 98 b0 81 a9 bd 12 a3 31 7a 82
```

I constructs the input needed to derive the key K_3, see Section 4.2 of [I-D.ietf-lake-edhoc], using the EDHOC hash algorithm:

```
K_3 = EDHOC-KDF(PRK_3e2m, TH_3, "K_3", h'', length) =
            = HKDF-Expand(PRK_3e2m, info, length),
```

where length is the key length of EDHOC AEAD algorithm, and info for K_3 is:

```
info =
(
 h'23CE4296FC64AB048A593B6711E4822011BB58D85D3798B081A9BD12A3317A82',
 "K_3",
 h'',
 16
)
```

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

```
info for K_3 (CBOR Sequence) (40 bytes)
58 20 23 ce 42 96 fc 64 ab 04 8a 59 3b 67 11 e4 82 20 11 bb 58 d8 5d
37 98 b0 81 a9 bd 12 a3 31 7a 82 63 4b 5f 33 40 10

K_3 (Raw Value) (16 bytes)
7a 40 e4 b6 75 9c 72 7e 8a ef f1 08 9e e7 69 af
```

I constructs the input needed to derive the nonce IV_3, see Section 4.2 of [I-D.ietf-lake-edhoc], using the EDHOC hash algorithm:

```
IV_3 = EDHOC-KDF(PRK_3e2m, TH_3, "IV_3", h'', length) =
       = HKDF-Expand(PRK_3e2m, info, length),
```

   where length is the nonce length of EDHOC AEAD algorithm, and info
   for IV_3 is:

```
info =
(
 h'23CE4296FC64AB048A593B6711E4822011BB58D85D3798B081A9BD12A3317A82',
 "IV_3",
 h'',
 13
)
```

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

```
info for IV_3 (CBOR Sequence) (41 bytes)
58 20 23 ce 42 96 fc 64 ab 04 8a 59 3b 67 11 e4 82 20 11 bb 58 d8 5d
37 98 b0 81 a9 bd 12 a3 31 7a 82 64 49 56 5f 33 40 0d
```

```
IV_3 (Raw Value) (13 bytes)
d3 98 90 65 7e ef 37 8f 36 52 0c b3 44
```

   I calculates CIPHERTEXT_3 as 'ciphertext' of COSE_Encrypt0 applied
   using the EDHOC AEAD algorithm with plaintext P_3, additional data
   A_3, key K_3 and nonce IV_3.

```
CIPHERTEXT_3 (Raw Value) (88 bytes)
4c 53 ed 22 c4 5f b0 0c ad 88 9b 4c 06 f2 a2 6c f4 91 54 cb 8b df 4e
ee 44 e2 b5 02 21 ab 1f 02 9d 3d 3e 05 23 dd f9 d7 61 0c 37 6c 72 8a
1e 90 16 92 f1 da 07 82 a3 47 2f f6 eb 1b b6 81 0c 6f 68 68 79 c9 a5
59 4f 8f 17 0c a5 a2 b5 bf 05 a7 4f 42 cd d9 c8 54 e0 1e
```

   message_3 is the CBOR bstr encoding of CIPHERTEXT_3:

```
message_3 (CBOR Sequence) (90 bytes)
58 58 4c 53 ed 22 c4 5f b0 0c ad 88 9b 4c 06 f2 a2 6c f4 91 54 cb 8b
df 4e ee 44 e2 b5 02 21 ab 1f 02 9d 3d 3e 05 23 dd f9 d7 61 0c 37 6c
72 8a 1e 90 16 92 f1 da 07 82 a3 47 2f f6 eb 1b b6 81 0c 6f 68 68 79
c9 a5 59 4f 8f 17 0c a5 a2 b5 bf 05 a7 4f 42 cd d9 c8 54 e0 1e
```

   The transcript hash TH_4 is calculated using the EDHOC hash
   algorithm:

   TH_4 = H(TH_3, CIPHERTEXT_3)

```
Input to calculate TH_4 (CBOR Sequence) (124 bytes)
58 20 23 ce 42 96 fc 64 ab 04 8a 59 3b 67 11 e4 82 20 11 bb 58 d8 5d
37 98 b0 81 a9 bd 12 a3 31 7a 82 58 58 4c 53 ed 22 c4 5f b0 0c ad 88
9b 4c 06 f2 a2 6c f4 91 54 cb 8b df 4e ee 44 e2 b5 02 21 ab 1f 02 9d
3d 3e 05 23 dd f9 d7 61 0c 37 6c 72 8a 1e 90 16 92 f1 da 07 82 a3 47
2f f6 eb 1b b6 81 0c 6f 68 68 79 c9 a5 59 4f 8f 17 0c a5 a2 b5 bf 05
a7 4f 42 cd d9 c8 54 e0 1e
```

```
TH_4 (Raw Value) (32 bytes)
63 ff 46 ad b9 eb 2f 89 ac ed 66 f7 c9 23 e6 6c 36 02 e2 56 57 b2 0a
8b 67 07 6d cc 92 aa d4 0b

TH_4 (CBOR Data Item) (34 bytes)
58 20 63 ff 46 ad b9 eb 2f 89 ac ed 66 f7 c9 23 e6 6c 36 02 e2 56 57
b2 0a 8b 67 07 6d cc 92 aa d4 0b
```

4.4. message_4

No external authorization data:

```
EAD_4 (CBOR Sequence) (0 bytes)
```

R constructs the plaintext P_4:

```
P_4 =
(
 ? EAD_4
)

P_4 (CBOR Sequence) (0 bytes)
```

R constructs the associated data for message_4:

```
A_4 =
(
 "Encrypt0",
 h'',
 TH_4
)

A_4 (CBOR Data Item) (45 bytes)
83 68 45 6e 63 72 79 70 74 30 40 58 20 63 ff 46 ad b9 eb 2f 89 ac ed
66 f7 c9 23 e6 6c 36 02 e2 56 57 b2 0a 8b 67 07 6d cc 92 aa d4 0b
```

R constructs the input needed to derive the EDHOC message_4 key, see Section 4.2 of [I-D.ietf-lake-edhoc], using the EDHOC hash algorithm:

```
K_4 = EDHOC-Exporter("EDHOC_K_4", h'', length)
      = EDHOC-KDF(PRK_4x3m, TH_4, "EDHOC_K_4", h'', length)
      = HKDF-Expand(PRK_4x3m, info, length)
```

where length is the key length of the EDHOC AEAD algorithm, and info for EDHOC_K_4 is:

```
info =
(
 h'63FF46ADB9EB2F89ACED66F7C923E66C3602E25657B20A8B67076DCC92AAD40B',
 "EDHOC_K_4",
 h'',
 16
)
```

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

```
info for K_4 (CBOR Sequence) (46 bytes)
58 20 63 ff 46 ad b9 eb 2f 89 ac ed 66 f7 c9 23 e6 6c 36 02 e2 56 57
b2 0a 8b 67 07 6d cc 92 aa d4 0b 69 45 44 48 4f 43 5f 4b 5f 34 40 10

K_4 (Raw Value) (16 bytes)
ee 55 a5 46 1b 2c 41 82 1b 1a be dc 03 b4 ef 50
```

   R constructs the input needed to derive the EDHOC message_4 nonce, see Section 4.2 of [I-D.ietf-lake-edhoc], using the EDHOC hash algorithm:

```
    IV_4 =
    = EDHOC-Exporter( "EDHOC_IV_4", h'', length )
    = EDHOC-KDF(PRK_4x3m, TH_4, "EDHOC_IV_4", h'', length)
    = HKDF-Expand(PRK_4x3m, info, length)
```

   where length is the nonce length of EDHOC AEAD algorithm, and info for EDHOC_IV_4 is:

```
info =
(
 h'63FF46ADB9EB2F89ACED66F7C923E66C3602E25657B20A8B67076DCC92AAD40B',
 "EDHOC_IV_4",
 h'',
 13
)
```

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

```
info for IV_4 (CBOR Sequence) (47 bytes)
58 20 63 ff 46 ad b9 eb 2f 89 ac ed 66 f7 c9 23 e6 6c 36 02 e2 56 57
b2 0a 8b 67 07 6d cc 92 aa d4 0b 6a 45 44 48 4f 43 5f 49 56 5f 34 40
0d

IV_4 (Raw Value) (13 bytes)
cb 14 8d 0f 30 c5 ce 4a 6d 80 eb f3 6c
```

   R calculates CIPHERTEXT_4 as 'ciphertext' of COSE_Encrypt0 applied using the EDHOC AEAD algorithm with plaintext P_4, additional data A_4, key K_4 and nonce IV_4.

```
CIPHERTEXT_4 (8 bytes)
fc 4f 5e 2f 54 c2 d4 08
```

message_4 is the CBOR bstr encoding of CIPHERTEXT_4:

```
message_4 (CBOR Sequence) (9 bytes)
48 fc 4f 5e 2f 54 c2 d4 08
```

4.5. OSCORE Parameters

The derivation of OSCORE parameters is specified in Appendix A.2 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 Appendix A.1 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. Since C_R is a numeric, it is converted to a byte string equal to its CBOR encoded form.

```
Client's OSCORE Sender ID (Raw Value) (1 bytes)
32
```

C_I is mapped to the Recipient ID of the client, i.e., the Sender ID of the server. Since C_I is a numeric, it is converted to a byte string equal to its CBOR encoded form.

```
Server's OSCORE Sender ID (Raw Value) (1 bytes)
0e
```

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

```
OSCORE Master Secret =
= EDHOC-Exporter("OSCORE_Master_Secret", h'', key_length)
= EDHOC-KDF(PRK_4x3m, TH_4, "OSCORE_Master_Secret", h'', key_length)
= HKDF-Expand(PRK_4x3m, info, key_length)
```

where key_length is by default the key length of the Application AEAD algorithm, and info for the OSCORE Master Secret is:

```
info =
(
 h'63FF46ADB9EB2F89ACED66F7C923E66C3602E25657B20A8B67076DCC92AAD40B',
 "OSCORE_Master_Secret",
 h'',
 16
)
```

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

```
info for OSCORE Master Secret (CBOR Sequence) (57 bytes)
58 20 63 ff 46 ad b9 eb 2f 89 ac ed 66 f7 c9 23 e6 6c 36 02 e2 56 57
b2 0a 8b 67 07 6d cc 92 aa d4 0b 74 4f 53 43 4f 52 45 5f 4d 61 73 74
65 72 5f 53 65 63 72 65 74 40 10

OSCORE Master Secret (Raw Value) (16 bytes)
01 4f df 73 06 7d fe fd 97 e6 b0 59 72 f9 0d 85
```

   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("OSCORE_Master_Salt", h'', salt_length)
= EDHOC-KDF(PRK_4x3m, TH_4, "OSCORE_Master_Salt", h'', salt_length)
= HKDF-Expand(PRK_4x3m, info, salt_length)
```

   where salt_length is the length of the OSCORE Master Salt, and info
   for the OSCORE Master Salt is:

```
info =
(
 h'63FF46ADB9EB2F89ACED66F7C923E66C3602E25657B20A8B67076DCC92AAD40B',
 "OSCORE_Master_Salt",
 h'',
 8
)
```

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

```
info for OSCORE Master Salt (CBOR Sequence) (55 bytes)
58 20 63 ff 46 ad b9 eb 2f 89 ac ed 66 f7 c9 23 e6 6c 36 02 e2 56 57
b2 0a 8b 67 07 6d cc 92 aa d4 0b 72 4f 53 43 4f 52 45 5f 4d 61 73 74
65 72 5f 53 61 6c 74 40 08

OSCORE Master Salt (Raw Value) (8 bytes)
cb 47 b6 ec d3 86 72 dd
```

4.6. Key Update

Key update is defined in Section 4.4 of [I-D.ietf-lake-edhoc].

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

KeyUpdate Nonce (Raw Value) (16 bytes)
e6 f5 49 b8 58 1a a2 92 53 cf ce 68 07 53 a4 00

PRK_4x3m after KeyUpdate (Raw Value) (32 bytes)
26 78 00 73 f8 ce 0b eb 71 03 e0 c7 17 d1 6d db bb f6 7b b1 f0 77 53
ca 97 df ec 34 73 23 47 4d
```

The OSCORE Master Secret is derived with the updated PRK_4x3m:

```
OSCORE Master Secret = HKDF-Expand(PRK_4x3m, info, key_length)
```

where info and key_length are unchanged.

```
OSCORE Master Secret after KeyUpdate (Raw Value) (16 bytes)
8f 7c 42 12 d7 e4 2a 1c 5f bb 5d c6 2f d7 b7 f3
```

The OSCORE Master Salt is derived with the updated PRK_4x3m:

```
OSCORE Master Salt = HKDF-Expand(PRK_4x3m, info, salt_length)
```

where info and salt_length are unchanged.

```
OSCORE Master Salt after KeyUpdate (Raw Value) (8 bytes)
87 eb 7d b2 fd cf a8 9c
```

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-12, 20 October 2021, <https://www.ietf.org/archive/id/draft-ietf-lake-edhoc-12.txt>.

[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

Authors' Addresses

Göran Selander
Ericsson AB
SE-164 80 Stockholm
Sweden

Email: goran.selander@ericsson.com

John Preuß Mattsson
Ericsson AB
SE-164 80 Stockholm
Sweden

Email: john.mattsson@ericsson.com