Workgroup: Network Working Group
Internet-Draft: draft-ietf-lake-traces-01
Published: 10 July 2022
Intended Status: Informational
Expires: 11 January 2023
Authors: G. Selander
Ericsson
J. Preuß Mattsson
Ericsson
M. Serafin
ASSA ABLOY
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 11 January 2023.

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 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 -14 and -15 of [I-D.ietf-lake-edhoc].

Editor's note: update reference to test vectors below:

A more extensive test vector suite and related code that was used to generate trace 1 can be found at: https://github.com/lake-wg/edhoc/tree/master/test-vectors-11. The test vector for trace 2 can be found at: https://github.com/lake-wg/edhoc/tree/master/test-vectors-11/p256.

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 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 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 bytes)
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 bytes)
2d

C_I (CBOR Data Item) (Connection identifier chosen by I) (1 bytes)
2d
```

No external authorization data:

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

I constructs message_1:

```
message_1 =
(
 0,
 0,
 h'31F82C7B5B9CBBF0F194D913CC12EF1532D328EF32632A4881A1C0701E237F04',
 -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) (int)
18

C_R (CBOR Data Item) (Connection identifier chosen by R) (1 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 the zero-length byte string:

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

PRK_2e (Raw Value) (32 bytes)
c5 76 10 5d 95 b4 d8 c1 8e 8f 65 5f 54 68 80 a8 54 f2 da 10 6c e5 a3
a0 2d 8b 3e de 7b aa bc a6
```

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)
c5 76 10 5d 95 b4 d8 c1 8e 8f 65 5f 54 68 80 a8 54 f2 da 10 6c e5 a3
a0 2d 8b 3e de 7b aa bc a6
```

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 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 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)
3081EE3081A1A003020102020462319EC4300506032B6570301D311B301906035504
030C124544484F4320526F6F74204564323535313930 1E170D323230333136303832
3433365A170D323931323331323331323330303030 5A30223120301E06035504030C174544
484F4320526573706F6E6465722045643235353139302A300506032B6570032100A1
DB47B95184854AD12A0C1A354E418AACE33AA0F2C662C00B3AC55DE92F9359300506
032B6570034100B723BC01EAB0928E8B2B6C98DE19CC3823D46E7D6987B032478FEC
FAF14537A1AF14CC8BE829C6B73044101837EB4ABC949565D86DCE51CFAE52AB82C1
52CB02
```

```
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
   7420456432353139301e170d323230331363038323433
   365a170d323931323331323303030305a30223120301e06
   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)
02590123A11822822E4879F2A41B510C1F9B58203AB11700841FCE193C323911EDB3
17B046DCF24B9950FD624884F7F57CD98B0758F13081EE3081A1A003020102020462
319EC4300506032B6570301D311B301906035504030C124544484F4320526F6F7420
4564323535313930 1E170D3232303331363038323433365A170D3239313233313233
303030305A30223120301E06035504030C174544484F4320526573706F6E64657220
4564323535313930 2A300506032B6570032100A1DB47B95184854AD12A0C1A354E41
8AACE33AA0F2C662C00B3AC55DE92F9359300506032B6570034100B723BC01EAB092
8E8B2B6C98DE19CC3823D46E7D6987B032478FECFAF14537A1AF14CC8BE829C6B730
44101837EB4ABC949565D86DCE51CFAE52AB82C152CB021820

MAC_2 (Raw Value) (32 bytes)
7f 0f b4 2e c3 ed db 0d b7 94 b8 ef dd 14 1e 44 ca 03 9a a7 19 c9 e6 61
8f ea f1 e7 83 3e 58 9a

MAC_2 (CBOR Data Item) (34 bytes)
58 20 7f 0f b4 2e c3 ed db 0d b7 94 b8 ef dd 14 1e 44 ca 03 9a a7 19 c9
e6 61 8f ea f1 e7 83 3e 58 9a
```

   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'58203AB11700841FCE193C323911EDB317B046DCF24B9950FD624884F7F57CD98
 B0758F13081EE3081A1A003020102020462319EC4300506032B6570301D311B3019
 06035504030C124544484F4320526F6F74204564323535313930 1E170D323230333
 1363038323433365A170D32393132333132333033030305A30223120301E06035504
 030C174544484F4320526573706F6E6465722045643235353139302A300506032B6
 570032100A1DB47B95184854AD12A0C1A354E418AACE33AA0F2C662C00B3AC55DE9
 2F9359300506032B6570034100B723BC01EAB0928E8B2B6C98DE19CC3823D46E7D6
 987B032478FECFAF14537A1AF14CC8BE829C6B73044101837EB4ABC949565D86DCE
 51CFAE52AB82C152CB02',
 h'7F0FB42EC3EDDB0DB794B8EFDD141E44CA039AA719C9E6618FEAF1E7833E589A'
]
```

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

```
846A5369676E6174757265314EA11822822E4879F2A41B510C1F9B59011558203AB1
1700841FCE193C323911EDB317B046DCF24B9950FD624884F7F57CD98B0758F13081
EE3081A1A003020102020462319EC4300506032B6570301D311B301906035504030C
124544484F4320526F6F74204564323535313930 1E170D323230333131363038323433
365A170D32393132333132333033030305A30223120301E06035504030C174544484F
4320526573706F6E6465722045643235353139302A300506032B6570032100A1DB47
B95184854AD12A0C1A354E418AACE33AA0F2C662C00B3AC55DE92F9359300506032B
6570034100B723BC01EAB0928E8B2B6C98DE19CC3823D46E7D6987B032478FECFAF1
4537A1AF14CC8BE829C6B73044101837EB4ABC949565D86DCE51CFAE52AB82C152CB
0258207F0FB42EC3EDDB0DB794B8EFDD141E44CA039AA719C9E6618FEAF1E7833E58
9A
```

R signs using the private authentication key SK_R

Signature_or_MAC_2 (Raw Value) (64 bytes)

```
f3 73 a7 20 3e fa 7d f0 73 8c 36 0e e0 80 17 1a ca 67 fc b1 75 f2 6d 78
5d 09 5b 55 eb 14 84 65 5c 39 e0 3a 3f 5f 9b dd 87 ef 8c 5f 2e c3 df e6
fb ba 49 b7 b4 62 5b 12 6f 27 de 30 17 67 27 0c
```

Signature_or_MAC_2 (CBOR Data Item) (66 bytes)

```
58 40 f3 73 a7 20 3e fa 7d f0 73 8c 36 0e e0 80 17 1a ca 67 fc b1 75 f2
6d 78 5d 09 5b 55 eb 14 84 65 5c 39 e0 3a 3f 5f 9b dd 87 ef 8c 5f 2e c3
df e6 fb ba 49 b7 b4 62 5b 12 6f 27 de 30 17 67 27 0c
```

R constructs the plaintext without padding:

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

```
PLAINTEXT_2 =
(
 ? PAD,
 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 f3 73 a7 20 3e fa 7d f0
73 8c 36 0e e0 80 17 1a ca 67 fc b1 75 f2 6d 78 5d 09 5b 55 eb 14 84 65
5c 39 e0 3a 3f 5f 9b dd 87 ef 8c 5f 2e c3 df e6 fb ba 49 b7 b4 62 5b 12
6f 27 de 30 17 67 27 0c
```

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'3AB11700841FCE193C323911EDB317B046DCF24B9950FD624884F7F57CD98B07',
 80
)
```

where the last value is the length of PLAINTEXT_2.

```
info for KEYSTREAM_2 (CBOR Sequence) (37 bytes)
0058203AB11700841FCE193C323911EDB317B046DCF24B9950FD624884F7F57CD98B0718
50
```

```
KEYSTREAM_2 (Raw Value) (80 bytes)
3a 37 61 38 40 a9 65 bc c8 1b 0d 06 21 d7 ed 9d 38 87 bb b3 b0 50 43 ff
ec c3 ab 78 1e 00 d7 66 32 d7 67 99 14 c1 72 04 bc 11 1b 1f de 5f 0d 99
da 68 71 64 8d 78 56 db 47 9b ec 48 ad 3a af 8c e0 4e 0b 4c ed 60 9e d3
40 49 f7 b6 26 ee ac 31
```

R calculates CIPHERTEXT_2 as XOR between PLAINTEXT_2 and KEYSTREAM_2:

```
CIPHERTEXT_2 (Raw Value) (80 bytes)
9b 2f 43 ba 6e e1 1c 4e 6c 00 5c 0a 3e 4c b5 dd cb f4 1c 93 8e aa 3e 0f
9f 4f 9d 76 fe 80 c0 7c f8 b0 9b 28 61 33 1f 7c e1 18 40 4a 35 4b 89 fc
86 51 91 5e b2 27 cd 06 c0 74 60 17 83 f9 70 6a 1b f4 42 fb 59 02 c5 c1
2f 6e 29 86 31 89 8b 3d
```

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 9b 2f 43 ba 6e e1 1c 4e 6c 00 5c 0a 3e 4c
b5 dd cb f4 1c 93 8e aa 3e 0f 9f 4f 9d 76 fe 80 c0 7c f8 b0 9b 28 61 33
1f 7c e1 18 40 4a 35 4b 89 fc 86 51 91 5e b2 27 cd 06 c0 74 60 17 83 f9
70 6a 1b f4 42 fb 59 02 c5 c1 2f 6e 29 86 31 89 8b 3d 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)
c5 76 10 5d 95 b4 d8 c1 8e 8f 65 5f 54 68 80 a8 54 f2 da 10 6c e5 a3
a0 2d 8b 3e de 7b aa bc a6
```

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

   TH_3 = H(TH_2, PLAINTEXT_2)

```
Input to calculate TH_3 (CBOR Sequence) (114 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 f3 73 a7 20 3e fa 7d f0 73 8c 36 0e e0 80 17 1a ca 67 fc
b1 75 f2 6d 78 5d 09 5b 55 eb 14 84 65 5c 39 e0 3a 3f 5f 9b dd 87 ef
8c 5f 2e c3 df e6 fb ba 49 b7 b4 62 5b 12 6f 27 de 30 17 67 27 0c
```

```
TH_3 (Raw Value) (32 bytes)
77 a5 1a 1c f0 eb 60 e4 06 1b fc 27 08 87 52 9c 16 b6 f4 b3 4d 88 a2 26
e7 06 5b 60 f4 15 26 71

TH_3 (CBOR Data Item) (34 bytes)
58 20 77 a5 1a 1c f0 eb 60 e4 06 1b fc 27 08 87 52 9c 16 b6 f4 b3 4d 88
a2 26 e7 06 5b 60 f4 15 26 71
```

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 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 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)
A11822822E48C24AB2FD7643C79F582077A51A1CF0EB60E4061BFC270887529C16B6
F4B34D88A226E7065B60F415267158F13081EE3081A1A00302010202046231 9EA030
0506032B6570301D311B301906035504030C124544484F4320526F6F742045643235
353139301E170D3232303331363038323430305A170D3239313233313233303030 30
5A30223120301E06035504030C174544484F4320496E69746961746F722045643235
353139302A300506032B6570032100ED06A8AE61A829BA5FA54525C9D07F48DD44A3
02F43E0F23D8CC20B73085141E300506032B657003410052124 1D8B3A770996BCFC9
B9EAD4E7E0A1C0DB353A3BDF2910B39275AE48B756015981850D27DB6734E37F6721
2267DD05EEFF27B9E7A813FA574B72A00B430B

context_3 (CBOR byte string) (294 bytes)
590123A11822822E48C24AB2FD7643C79F582077A51A1CF0EB60E4061BFC27088752
9C16B6F4B34D88A226E7065B60F415267158F13081EE3081A1A00302010202046231
9EA0300506032B6570301D311B301906035504030C124544484F4320526F6F742045
643235353139301E170D323230333136303832343030 5A170D323931323331323330
3030305A30223120301E06035504030C174544484F4320496E69746961746F722045
643235353139302A300506032B6570032100ED06A8AE61A829BA5FA54525C9D07F48
DD44A302F43E0F23D8CC20B73085141E300506032B657003410052124 1D8B3A77099
6BCFC9B9EAD4E7E0A1C0DB353A3BDF2910B39275AE48B756015981850D27DB6734E3
7F67212267DD05EEFF27B9E7A813FA574B72A00B430B
```

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'A11822822E48C24AB2FD7643C79F582077A51A1CF0EB60E4061BFC270887529C16B6
   F4B34D88A226E7065B60F415267158F13081EE3081A1A003020102020462319EA030
   0506032B6570301D311B301906035504030C124544484F4320526F6F742045643235
   353139301E170D3232303331363038323430305A170D3239313233313233303030
   5A30223120301E06035504030C174544484F4320496E69746961746F722045643235
   353139302A300506032B6570032100ED06A8AE61A829BA5FA54525C9D07F48DD44A3
   02F43E0F23D8CC20B73085141E300506032B6570034100521241D8B3A770996BCFC9
   B9EAD4E7E0A1C0DB353A3BDF2910B39275AE48B756015981850D27DB6734E37F6721
   2267DD05EEFF27B9E7A813FA574B72A00B430B',
 32
)
```

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

```
info for MAC_3 (CBOR Sequence) (297 bytes)
06590123A11822822E48C24AB2FD7643C79F582077A51A1CF0EB60E4061BFC270887
529C16B6F4B34D88A226E7065B60F415267158F13081EE3081A1A003020102020462
319EA0300506032B6570301D311B301906035504030C124544484F4320526F6F7420
4564323535313930 1E170D3232303331363038323430305A170D3239313233313233
303030305A30223120301E06035504030C174544484F4320496E69746961746F7220
4564323535313930 2A300506032B6570032100ED06A8AE61A829BA5FA54525C9D07F
48DD44A302F43E0F23D8CC20B73085141E300506032B6570034100521241D8B3A770
996BCFC9B9EAD4E7E0A1C0DB353A3BDF2910B39275AE48B756015981850D27DB6734
E37F67212267DD05EEFF27B9E7A813FA574B72A00B430B1820

MAC_3 (Raw Value) (32 bytes)
77 78 cc 72 7b f8 40 c9 bf c9 70 18 a4 d1 29 7c 51 74 43 0a 17 42 ae 97
6a c4 15 e9 7a 42 ce 55

MAC_3 (CBOR Data Item) (34 bytes)
58 20 77 78 cc 72 7b f8 40 c9 bf c9 70 18 a4 d1 29 7c 51 74 43 0a 17 42
ae 97 6a c4 15 e9 7a 42 ce 55
```

   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'582077A51A1CF0EB60E4061BFC270887529C16B6F4B34D88A226E7065B60F4
   15267158F13081EE3081A1A003020102020462319EA0300506032B6570301D
   311B301906035504030C124544484F4320526F6F74204564323535313930 1E
   170D323230333136303832343030 5A170D323933313233313233303030305A30
   223120301E06035504030C174544484F4320496E69746961746F7220456432
   35353139302A300506032B6570032100ED06A8AE61A829BA5FA54525C9D07F
   48DD44A302F43E0F23D8CC20B73085141E300506032B657003410052124 1D8
   B3A770996BCFC9B9EAD4E7E0A1C0DB353A3BDF2910B39275AE48B756015981
   850D27DB6734E37F67212267DD05EEFF27B9E7A813FA574B72A00B430B',
 h'7778CC727BF840C9BFC97018A4D1297C5174430A1742AE976AC415E97A42CE55'
]

Message to be signed 3 (CBOR Data Item) (341 bytes)
846A5369676E6174757265314EA11822822E48C24AB2FD7643C79F590115582077A5
1A1CF0EB60E4061BFC270887529C16B6F4B34D88A226E7065B60F415267158F13081
EE3081A1A003020102020462319EA0300506032B6570301D311B301906035504030C
124544484F4320526F6F74204564323535313930 1E170D323230333136303832343030
305A170D323933313233313233303030305A30223120301E06035504030C174544484F
4320496E69746961746F7220456432353531393 02A300506032B6570032100ED06A8
AE61A829BA5FA54525C9D07F48DD44A302F43E0F23D8CC20B73085141E300506032B
657003410052124 1D8B3A770996BCFC9B9EAD4E7E0A1C0DB353A3BDF2910B39275AE
48B756015981850D27DB6734E37F67212267DD05EEFF27B9E7A813FA574B72A00B43
0B58207778CC727BF840C9BFC97018A4D1297C5174430A1742AE976AC415E97A42CE
55

   I signs using the private authentication key SK_I:

Signature_or_MAC_3 (Raw Value) (64 bytes)
ab 7e fd 2d 2f 2d 2b 7f f4 e1 f1 6b f5 51 a5 09 f1 f7 6d 37 e0 28 b7 0c
9c 98 32 02 6c 52 c2 37 21 73 81 91 91 c5 95 59 07 b1 63 c8 86 0f bf 7e
62 51 b6 71 9f 76 1c b9 44 9e 1d 47 57 41 c6 0b

Signature_or_MAC_3 (CBOR Data Item) (66 bytes)
58 40 ab 7e fd 2d 2f 2d 2b 7f f4 e1 f1 6b f5 51 a5 09 f1 f7 6d 37 e0 28
b7 0c 9c 98 32 02 6c 52 c2 37 21 73 81 91 91 c5 95 59 07 b1 63 c8 86 0f
bf 7e 62 51 b6 71 9f 76 1c b9 44 9e 1d 47 57 41 c6 0b

   I constructs the plaintext without padding:

PAD (CBOR sequence of simple type) (0 bytes)
```

```
PLAINTEXT_3 =
(
 ? PAD,
 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 ab 7e fd 2d 2f 2d 2b 7f
f4 e1 f1 6b f5 51 a5 09 f1 f7 6d 37 e0 28 b7 0c 9c 98 32 02 6c 52 c2 37
21 73 81 91 91 c5 95 59 07 b1 63 c8 86 0f bf 7e 62 51 b6 71 9f 76 1c b9
44 9e 1d 47 57 41 c6 0b
```

I constructs the associated data for message_3:

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

```
A_3 (CBOR Data Item) (45 bytes)
8368456E6372797074304058 2077A51A1CF0EB60E4061BFC270887529C16B6F4B34D
88A226E7065B60F4152671
```

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'77a51a1cf0eb60e4061bfc270887529c16b6f4b34d88a226e7065b60f4152671',
 16
)
```

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

```
info for K_3 (CBOR Sequence) (36 bytes)
03582077A51A1CF0EB60E4061BFC270887529C16B6F4B34D88A226E7065B60F41526
7110
```

```
K_3 (Raw Value) (16 bytes)
8f cd 39 25 44 41 fc 8d 54 b9 cd 97 a6 43 82 4d
```

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'77a51a1cf0eb60e4061bfc270887529c16b6f4b34d88a226e7065b60f4152671',
 13
)
```

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

info for IV_3 (CBOR Sequence) (36 bytes)

```
04582077A51A1CF0EB60E4061BFC270887529C16B6F4B34D88A226E7065B60F41526
710D
```

IV_3 (Raw Value) (13 bytes)

```
5a 5d b1 7d 7b 83 b0 95 14 5c 77 42 a8
```

   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) (88 bytes)

```
2f c1 95 56 f6 7c 92 9a 97 34 78 9e ce c6 0a af 8f 50 32 4f c3 1b d0 78
01 d5 7c ec 00 d3 bf 1e a4 db af ac 34 2f e1 68 6c bc ea 0c ec 45 02 95
98 05 b1 53 81 9e 27 46 38 cc 23 bb c4 f4 e3 94 77 f8 a8 ff f5 ca af 5e
32 d4 ca 4e 7c 74 8d 2b 51 9d 21 b8 d7 57 72 48
```

   message_3 is the CBOR bstr encoding of CIPHERTEXT_3:

message_3 (CBOR Sequence) (90 bytes)

```
58 58 2f c1 95 56 f6 7c 92 9a 97 34 78 9e ce c6 0a af 8f 50 32 4f c3 1b
d0 78 01 d5 7c ec 00 d3 bf 1e a4 db af ac 34 2f e1 68 6c bc ea 0c ec 45
02 95 98 05 b1 53 81 9e 27 46 38 cc 23 bb c4 f4 e3 94 77 f8 a8 ff f5 ca
af 5e 32 d4 ca 4e 7c 74 8d 2b 51 9d 21 b8 d7 57 72 48
```

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

   TH_4 = H(TH_3, PLAINTEXT_3)

Input to calculate TH_4 (CBOR Sequence) (114 bytes)

```
58 20 77 a5 1a 1c f0 eb 60 e4 06 1b fc 27 08 87 52 9c 16 b6 f4 b3 4d 88
a2 26 e7 06 5b 60 f4 15 26 71 a1 18 22 82 2e 48 c2 4a b2 fd 76 43 c7 9f
58 40 ab 7e fd 2d 2f 2d 2b 7f f4 e1 f1 6b f5 51 a5 09 f1 f7 6d 37 e0 28
b7 0c 9c 98 32 02 6c 52 c2 37 21 73 81 91 91 c5 95 59 07 b1 63 c8 86 0f
bf 7e 62 51 b6 71 9f 76 1c b9 44 9e 1d 47 57 41 c6 0b
```

```
TH_4 (Raw Value) (32 bytes)
dd cc c1 4e 32 33 0d 2d a9 3d 13 4b 0c 57 1e 2e 22 a2 8e 62 08 13 5e 7c
da 45 23 1d 85 0d ce c3

TH_4 (CBOR Data Item) (34 bytes)
58 20 dd cc c1 4e 32 33 0d 2d a9 3d 13 4b 0c 57 1e 2e 22 a2 8e 62 08 13
5e 7c da 45 23 1d 85 0d ce c3
```

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'',
 TH_4
)

A_4 (CBOR Data Item) (45 bytes)
8368456E637279707430405820DDCCC14E32330D2DA93D134B0C571E2E22A28E6208
135E7CDA45231D850DCEC3
```

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'ddccc14e32330d2da93d134b0c571e2e22a28e6208135e7cda45231d850dcec3',
 16
)
```

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

```
info for K_4 (CBOR Sequence) (36 bytes)
085820DDCCC14E32330D2DA93D134B0C571E2E22A28E6208135E7CDA45231D850DCE
C310

K_4 (Raw Value) (16 bytes)
e5 5a ba d2 4c 2d 5b 1a a4 a7 e9 98 5c 56 99 ce
```

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'ddccc14e32330d2da93d134b0c571e2e22a28e6208135e7cda45231d850dcec3',
 13
)
```

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

```
info for IV_4 (CBOR Sequence) (36 bytes)
095820DDCCC14E32330D2DA93D134B0C571E2E22A28E6208135E7CDA45231D850DCE
C30D

IV_4 (Raw Value) (13 bytes)
f2 4b 5c da 64 f7 a5 dc 08 dd 6f ef d5
```

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)
52 3b 02 82 a1 3c 89 23
```

message_4 is the CBOR bstr encoding of CIPHERTEXT_4:

```
message_4 (CBOR Sequence) (9 bytes)
48 52 3b 02 82 a1 3c 89 23
```

3.5. PRK_out and PRK_exporter

PRK_out is specified in Section 4.1.2 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'ddccc14e32330d2da93d134b0c571e2e22a28e6208135e7cda45231d850dcec3',
 32
)
```

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

```
info for PRK_out (CBOR Sequence) (37 bytes)
075820DDCCC14E32330D2DA93D134B0C571E2E22A28E6208135E7CDA45231D850DCEC3
1820

PRK_out (Raw Value) (32 bytes)
2d 6d b7 af a2 e8 72 e2 31 65 19 63 03 e2 f4 56 71 40 81 ab 90 81 1e 29
26 d7 28 3a 3f 49 f1 ac
```

   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)
ea f9 12 d6 9a 96 68 59 f0 e6 9b c0 fd 16 a4 b9 5b bd 9b 6c be 2c 52 5b
d2 3f d4 7f 34 05 9e 10
```

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 bytes)
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 bytes)
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)
d6 dd 09 b1 37 35 9f 0a d2 15 dd 02 19 62 c0 5c

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)
67 ec d7 d5 bb 49 46 17
```

3.7. Key Update

Key update is defined in Section 4.2.2 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)
6f a6 ee 2f fb f3 80 5b d1 d8 ab 59 0b 89 32 a2 05 af f6 a5 fa 79 4b 1d
e9 3c 57 af a9 63 8f b3
```

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)
a2 70 ad da a5 39 d9 44 9c ae 79 e9 f5 7b 18 53 28 aa ac e2 bb b4 25 01
d9 d0 dc 30 7f 10 fb 28
```

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)
66 ec b0 db 0a 9e 49 6f 67 c0 b5 55 54 79 6e 3e
```

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)
18 f5 7d c3 4e d5 49 17
```

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 bytes)
03
```

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

```
SUITES_I (CBOR Data Item) (1 bytes)
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
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
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 bytes)
0e

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 =
(
 3,
 6,
 h'741A13D7BA048FBB615E94386AA3B61BEA5B3D8F65F32620B749BEE8D278
 EFA9',
 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 bytes)
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 bytes)
37

C_I (CBOR Data Item) (Connection identifier chosen by I) (1 bytes)
37
```

No external authorization data:

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

I constructs message_1:

```
message_1 =
(
 3,
 [6, 2],
 h'8AF6F430EBE18D34184017A9A11BF511C8DFF8F834730B96C1B7C8DBCA2F
 C3B6',
 -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) (0 bytes)
27

C_R (CBOR Data Item) (Connection identifier chosen by R) (1 bytes)
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 the zero-length byte string:

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

PRK_2e (Raw Value) (32 bytes)
fd 9e ef 62 74 87 e4 03 90 ca e9 22 51 2d b5 a6 47 c0 8d c9 0d eb 22
b7 2e ce 6f 15 6f f1 c3 96
```

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:

```
==>MT This should be called SK_R
Responder's private authentication key
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
G_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'9d2af3a3d3fc06aea8110f14ba12ad0b4fb7e5cdf59c7df1cf2dfe9c2024439c',
 32
)

info for SALT_3e2m (CBOR Sequence) (37 bytes)
0158209D2AF3A3D3FC06AEA8110F14BA12AD0B4FB7E5CDF59C7DF1CF2DFE9C2024439C
1820

SALT_3e2m (Raw Value) (32 bytes)
39 92 a4 4f 33 0f ac fc 25 6a 00 ba 32 0d 77 8a 69 f9 99 70 db 39 8a 61
3f 9c 25 06 8e 0a bd 03
```

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)
7e 23 0e 62 b9 09 ca 74 92 36 7a aa 8a 22 9f 63 06 c5 ac 67 48 21 84 b3
33 62 d2 8d 17 7a 56 e9
```

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:

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

```
CRED_R (CBOR Data Item) (95 bytes)
A2026B6578616D706C652E65647508A101A501020241322001215820BBC34960526EA4D
32E940CAD2A234148DDC21791A12AFBCBAC93622046DD44F02258204519E257236B2A0C
E2023F0931F1F386CA7AFDA64FCDE0108C224C51EABF6072
```

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'a104413258209d2af3a3d3fc06aea8110f14ba12ad0b4fb7e5cdf59c7df1cf2d
 fe9c2024439ca2026b6578616d706c652e65647508a101a5010202413220012158
 20bbc34960526ea4d32e940cad2a234148ddc21791a12afbcbac93622046dd44f0
 2258204519e257236b2a0ce2023f0931f1f386ca7afda64fcde0108c224c51eabf
 6072',
 8
)
```

   where the last value is the EDHOC MAC length.

```
info for MAC_2 (CBOR Sequence) (137 bytes)
025885A104413258209D2AF3A3D3FC06AEA8110F14BA12AD0B4FB7E5CDF59C7DF1CF2D
FE9C2024439CA2026B6578616D706C652E65647508A101A5010202413220012158 20BB
C34960526EA4D32E940CAD2A234148DDC21791A12AFBCBAC93622046DD44F022582045
19E257236B2A0CE2023F0931F1F386CA7AFDA64FCDE0108C224C51EABF607208

MAC_2 (Raw Value) (8 bytes)
ad 01 bc 30 c6 91 11 76

MAC_2 (CBOR Data Item) (9 bytes)
48 ad 01 bc 30 c6 91 11 76
```

   Since METHOD = 3, Signature_or_MAC_2 is MAC_2:

```
Signature_or_MAC_2 (Raw Value) (8 bytes)
ad 01 bc 30 c6 91 11 76

Signature_or_MAC_2 (CBOR Data Item) (9 bytes)
48 ad 01 bc 30 c6 91 11 76
```

   R constructs PLAINTEXT_2 without padding:

```
PAD (CBOR sequence of simple type) (0 bytes)

PLAINTEXT_2 =
(
 ? PAD,
 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 ad 01 bc 30 c6 91 11 76
```

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'9d2af3a3d3fc06aea8110f14ba12ad0b4fb7e5cdf59c7df1cf2dfe9c2024439c',
 10
)
```

where last value is the length of PLAINTEXT_2.

```
info for KEYSTREAM_2 (CBOR Sequence) (36 bytes)
0058209D2AF3A3D3FC06AEA8110F14BA12AD0B4FB7E5CDF59C7DF1CF2DFE9C2024439C
0A

KEYSTREAM_2 (Raw Value) (10 bytes)
b9 c7 41 6a a3 35 46 54 15 4f
```

R calculates CIPHERTEXT_2 as XOR between PLAINTEXT_2 and KEYSTREAM_2:

```
CIPHERTEXT_2 (Raw Value) (10 bytes)
8b 8f ec 6b 1f 05 80 c5 04 39
```

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 8b 8f ec 6b 1f 05 80 c5 04 39 27
```

4.5. message_3

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

```
TH_3 = H( TH_2, PLAINTEXT_2 )
```

```
Input to calculate TH_3 (CBOR Sequence) (44 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 ad 01 bc 30 c6 91 11 76

TH_3 (Raw Value) (32 bytes)
08 5d e1 6d 9c 82 35 cb f5 7c 46 d0 6d 16 d4 56 a6 c0 ad 81 aa 4b 44 8b
6a bc 98 dc ba 61 25 eb

TH_3 (CBOR Data Item) (34 bytes)
58 20 08 5d e1 6d 9c 82 35 cb f5 7c 46 d0 6d 16 d4 56 a6 c0 ad 81 aa 4b
44 8b 6a bc 98 dc ba 61 25 eb
```

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
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
G_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'085de16d9c8235cbf57c46d06d16d456a6c0ad81aa4b448b6abc98dcba61
 25eb',
 32
)
```

```
info for SALT_4e3m (CBOR Sequence) (37 bytes)
055820085DE16D9C8235CBF57C46D06D16D456A6C0AD81AA4B448B6ABC98DCBA6125EB
1820

SALT_4e3m (Raw Value) (32 bytes)
b8 42 a7 11 41 6d 16 f6 93 24 96 9f 68 bd a7 46 62 9d 71 b9 9d 0a 88 16
0e 12 b9 4f 45 58 ec fd
```

PRK_4e3m is specified in Section 4.1.1.3 of [I-D.ietf-lake-edhoc].

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(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_4x3m (Raw Value) (32 bytes)
9e da 8c d7 55 ae 3b 80 b4 7e 8d db b8 d7 c5 fe 2b 62 b4 62 e4 bc ba 2c
6c 8e a3 6e e5 fb 60 4d
```

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:

```
{                                                 /CCS/
  2 : "42-50-31-FF-EF-37-32-39",                  /sub/
  8 : {                                           /cnf/
    1 : {                                         /COSE_Key/
      1 : 1,                                      /kty/
      2 : h'2b',                                  /kid/
     -1 : 4,                                      /crv/
     -2 : h'AC75E9ECE3E50BFC8ED6039988952240
             5C47BF16DF96660A41298CB4307F7EB6'    /x/
     -3 : h'6E5DE611388A4B8A8211334AC7D37ECB
             52A387D257E6DB3C2A93DF21FF3AFFC8'    /y/
    }
  }
}
```

CRED_I (CBOR Data Item) (107 bytes)

```
A2027734322D35302D33312D46462D45462D33372D33322D333908A101A5010202412B
2001215820AC75E9ECE3E50BFC8ED60399889522405C47BF16DF96660A41298CB4307F
7EB62258206E5DE611388A4B8A8211334AC7D37ECB52A387D257E6DB3C2A93DF21FF3A
FFC8
```

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)

```
A104412B5820085DE16D9C8235CBF57C46D06D16D456A6C0AD81AA4B448B6ABC98DCBA
6125EBA2027734322D35302D33312D46462D45462D33372D33322D333908A101A50102
02412B2001215820AC75E9ECE3E50BFC8ED60399889522405C47BF16DF96660A41298C
B4307F7EB62258206E5DE611388A4B8A8211334AC7D37ECB52A387D257E6DB3C2A93DF
21FF3AFFC8
```

context_3 (CBOR byte string) (147 bytes)

```
5891A104412B5820085DE16D9C8235CBF57C46D06D16D456A6C0AD81AA4B448B6ABC98
DCBA6125EBA2027734322D35302D33312D46462D45462D33372D33322D333908A101A5
010202412B2001215820AC75E9ECE3E50BFC8ED60399889522405C47BF16DF96660A41
298CB4307F7EB62258206E5DE611388A4B8A8211334AC7D37ECB52A387D257E6DB3C2A
93DF21FF3AFFC8
```

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'A104412B5820085DE16D9C8235CBF57C46D06D16D456A6C0AD81AA4B448B6ABC
 98DCBA6125EBA2027734322D35302D33312D46462D45462D33372D33322D333908
 A101A5010202412B2001215820AC75E9ECE3E50BFC8ED60399889522405C47BF16
 DF96660A41298CB4307F7EB62258206E5DE611388A4B8A8211334AC7D37ECB52A3
 87D257E6DB3C2A93DF21FF3AFFC8',
 8
)
```

   where the last value is the EDHOC MAC length.

```
info for MAC_3 (CBOR Sequence) (149 bytes)
065891A104412B5820085DE16D9C8235CBF57C46D06D16D456A6C0AD81AA4B448B6ABC
98DCBA6125EBA2027734322D35302D33312D46462D45462D33372D33322D333908A101
A5010202412B2001215820AC75E9ECE3E50BFC8ED60399889522405C47BF16DF96660A
41298CB4307F7EB62258206E5DE611388A4B8A8211334AC7D37ECB52A387D257E6DB3C
2A93DF21FF3AFFC808

MAC_3 (Raw Value) (8 bytes)
35 4f 0b c2 74 1e ea c6

MAC_3 (CBOR Data Item) (9 bytes)
48 35 4f 0b c2 74 1e ea c6
```

   Since METHOD = 3, Signature_or_MAC_3 is MAC_3:

```
Signature_or_MAC_3 (Raw Value) (8 bytes)
35 4f 0b c2 74 1e ea c6

Signature_or_MAC_3 (CBOR Data Item) (9 bytes)
48 35 4f 0b c2 74 1e ea c6b
```

   I constructs PLAINTEXT_3 without padding:

```
PAD (CBOR sequence of simple type) (0 bytes)

PLAINTEXT_3 =
(
 ? PAD,
 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 35 4f 0b c2 74 1e ea c6
```

I constructs the associated data for message_3:

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

A_3 (CBOR Data Item) (45 bytes)
8368456E6372797074304058 20085DE16D9C8235CBF57C46D06D16D456A6C0AD81AA4B
448B6ABC98DCBA6125EB
```

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'085de16d9c8235cbf57c46d06d16d456a6c0ad81aa4b448b6abc98dcba6125eb',
 16
)
```

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

```
info for K_3 (CBOR Sequence) (36 bytes)
035820085DE16D9C8235CBF57C46D06D16D456A6C0AD81AA4B448B6ABC98DCBA6125EB
10

K_3 (Raw Value) (16 bytes)
20 68 72 4b 29 0c db 72 c3 55 99 cd 5e b9 87 97
```

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'085de16d9c8235cbf57c46d06d16d456a6c0ad81aa4b448b6abc98dcba6125eb',
 13
)
```

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

```
info for IV_3 (CBOR Sequence) (36 bytes)
045820085DE16D9C8235CBF57C46D06D16D456A6C0AD81AA4B448B6ABC98DCBA6125EB
0D

IV_3 (Raw Value) (13 bytes)
cb db 34 e5 93 8e ef 39 fa 9c 28 f5 82
```

   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 5c 84 20 03 67 64 46 2f 57 35 79 86 61 6c 8d 21 b0
```

   message_3 is the CBOR bstr encoding of CIPHERTEXT_3:

```
message_3 (CBOR Sequence) (20 bytes)
27 52 c2 5c 84 20 03 67 64 46 2f 57 35 79 86 61 6c 8d 21 b0
```

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

   TH_4 = H(TH_3, PLAINTEXT_3)

```
Input to calculate TH_4 (CBOR Sequence) (44 bytes)
58 20 08 5d e1 6d 9c 82 35 cb f5 7c 46 d0 6d 16 d4 56 a6 c0 ad 81 aa 4b
44 8b 6a bc 98 dc ba 61 25 eb 2b 48 35 4f 0b c2 74 1e ea c6

TH_4 (Raw Value) (32 bytes)
a4 09 7a 6b 9e 39 f7 d3 dc 4f 8a f2 c4 a8 64 5b 37 3d 7a f5 86 f4 15 df
62 6e 16 b6 ac 27 55 d3

TH_4 (CBOR Data Item) (34 bytes)
58 20 a4 09 7a 6b 9e 39 f7 d3 dc 4f 8a f2 c4 a8 64 5b 37 3d 7a f5 86 f4
15 df 62 6e 16 b6 ac 27 55 d3
```

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'',
 TH_4
)
```

```
A_4 (CBOR Data Item) (45 bytes)
8368456E637279707430405820A4097A6B9E39F7D3DC4F8AF2C4A8645B373D7AF586F4
15DF626E16B6AC2755D3
```

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'a4097a6b9e39f7d3dc4f8af2c4a8645b373d7af586f415df626e16b6ac2755d3',
 16
)
```

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

```
info for K_4 (CBOR Sequence) (36 bytes)
085820A4097A6B9E39F7D3DC4F8AF2C4A8645B373D7AF586F415DF626E16B6AC2755D3
10
```

```
K_4 (Raw Value) (16 bytes)
d2 33 fd c7 6e 45 c4 59 48 af 81 f0 72 34 1d b6
```

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

```
info =
 (
 9,
 h'a4097a6b9e39f7d3dc4f8af2c4a8645b373d7af586f415df626e16b6ac2755d3',
 13
)
```

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

```
info for IV_4 (CBOR Sequence) (36 bytes)
095820A4097A6B9E39F7D3DC4F8AF2C4A8645B373D7AF586F415DF626E16B6AC2755D3
0D

IV_4 (Raw Value) (13 bytes)
aa 33 90 cd 64 93 1a e0 40 10 d8 a8 38
```

   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)
dd f9 77 df 1c ac 7f c3
```

   message_4 is the CBOR bstr encoding of CIPHERTEXT_4:

```
message_4 (CBOR Sequence) (9 bytes)
48 dd f9 77 df 1c ac 7f c3
```

4.7. PRK_out and PRK_exporter

   PRK_out is specified in Section 4.1.2 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'a4097a6b9e39f7d3dc4f8af2c4a8645b373d7af586f415df626e16b6ac2755d3',
 32
)
```

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

```
info for PRK_out (CBOR Sequence) (37 bytes)
075820A4097A6B9E39F7D3DC4F8AF2C4A8645B373D7AF586F415DF626E16B6AC2755D3
1820
```

```
PRK_out (Raw Value) (32 bytes)
c9 3a db 8f cd a3 ec ba 4b 8b 7a e3 32 db 1d 17 71 e0 a2 fb 9d da 3c 0e
06 fe fa bc 14 cc 17 f2
```

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)
86 78 e6 a9 ea 3f a1 d2 ee 2d be 99 bd a9 76 5c ee f5 c4 47 be 6f f4 ea
0b 21 fb 63 cc 82 2f 48
```

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 bytes)
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 bytes)
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)
f3 fb 21 49 fd 08 c1 8a 40 93 8a e5 70 3b fb 38
```

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)
25 54 9b 72 55 ca fa 1d
```

4.9. Key Update

   ey update is defined in Section 4.2.2 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)
83 de 3f a7 eb 2a a7 b0 0a 71 10 9f 41 5f a2 86 52 fc 15 4e 37 69 56 c5
c3 74 bd a8 a4 3d 45 07
```

   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)
a8 05 45 f5 51 94 8b 71 b2 b6 b0 04 88 ed 47 e8 74 3a 68 b1 ca a0 e8 20
cb 3f 94 ad 45 41 e6 a6
```

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)
5d 96 c2 fb 5c 1a d1 2e 32 4a 59 f4 a4 cd e2 5b
```

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)
3a ef 12 c4 27 00 cb 60
```

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-14, 18 May 2022, <https://www.ietf.org/archive/id/draft-ietf-lake-edhoc-14.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

Authors' Addresses

Göran Selander
Ericsson
SE-164 40 Stockholm
Sweden

Email: goran.selander@ericsson.com

John Preuß Mattsson
Ericsson
SE-164 40 Stockholm
Sweden

Email: john.mattsson@ericsson.com

Marek Serafin
ASSA ABLOY
32-080 Zabierzów
Poland

Email: marek.serafin@assaabloy.com

Marco Tiloca
RISE
SE-164 40 Stockholm
Sweden

Email: marco.tiloca@ri.se