Stream:	Internet Engineering Task Force (IETF)				
RFC:	9529				
Category:	Informational				
Published:	March 2024				
ISSN:	2070-1721				
Authors:	G. Selander *Ericsson*	J. Preuß Mattsson *Ericsson*	M. Serafin *ASSA ABLOY*	M. Tiloca *RISE*	M. Vučinić *Inria*

RFC 9529
Traces of Ephemeral Diffie-Hellman Over COSE (EDHOC)

Abstract

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

Status of This Memo

This document is not an Internet Standards Track specification; it is published for informational purposes.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Not all documents approved by the IESG are candidates for any level of Internet Standard; see Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc9529.

Copyright Notice

Copyright (c) 2024 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction

EDHOC [RFC9528] is a lightweight authenticated key exchange protocol designed for highly constrained settings. This document contains annotated traces of EDHOC sessions with input, output, and intermediate processing results to simplify testing of implementations. The traces have been verified by two independent implementations.

1.1. Setup

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

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

The document contains two traces:

- Section 2 - Authentication with signature keys identified by the hash value of the X.509 certificates (provided in Section 2.8). The endpoints use Edwards-curve Digital Signature Algorithm (EdDSA) [RFC8032] for authentication and X25519 [RFC7748] for ephemeral-ephemeral Diffie-Hellman (DH) key exchange.
- Section 3 - Authentication with static Diffie-Hellman keys identified by short key identifiers labeling CWT Claims Sets (CCSs) [RFC8392]. The endpoints use NIST P-256 [SP-800-186] for both ephemeral-ephemeral and ephemeral-static DH key exchange. This trace also illustrates the cipher suite negotiation and provides an example of low protocol overhead with messages sizes of 39, 45, and 19 bytes.

Examples of invalid EDHOC messages are found in Section 4.

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

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

Note 3. When the protocol transporting EDHOC messages does not inherently provide correlation across all messages, like Constrained Application Protocol (CoAP) [RFC7252], then some messages are typically prepended with connection identifiers and potentially a message_1 indicator (see Section 3.4.1 and Appendix A.2 of [RFC9528]). Those bytes are not included in the traces in this document.

1.2. Requirements Language

The key words "**MUST**", "**MUST NOT**", "**REQUIRED**", "**SHALL**", "**SHALL NOT**", "**SHOULD**", "**SHOULD NOT**", "**RECOMMENDED**", "**NOT RECOMMENDED**", "**MAY**", and "**OPTIONAL**" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

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

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

- EDHOC AEAD algorithm = AES-CCM-16-64-128
- EDHOC hash algorithm = SHA-256
- EDHOC Message Authentication Code (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 CBOR Object Signing and Encryption (COSE) header parameter 'x5t'.

2.1. message_1

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

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

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

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

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

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

```
Initiator's ephemeral public key
G_X (Raw Value) (32 bytes)
31 f8 2c 7b 5b 9c bb f0 f1 94 d9 13 cc 12 ef 15 32 d3 28 ef 32 63 2a
48 81 a1 c0 70 1e 23 7f 04
```

```
Initiator's ephemeral public key
G_X (CBOR Data Item) (34 bytes)
58 20 31 f8 2c 7b 5b 9c bb f0 f1 94 d9 13 cc 12 ef 15 32 d3 28 ef 32
63 2a 48 81 a1 c0 70 1e 23 7f 04
```

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

```
Connection identifier chosen by Initiator
C_I (Raw Value) (1 byte)
2d
```

```
Connection identifier chosen by Initiator
C_I (CBOR Data Item) (1 byte)
2d
```

No external authorization data:

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

The Initiator constructs message_1:

```
message_1 =
(
 0,
 0,
 h'31f82c7b5b9cbbf0f194d913cc12ef1532d328ef32632a48
   81a1c0701e237f04',
 -14
)
```

```
message_1 (CBOR Sequence) (37 bytes)
00 00 58 20 31 f8 2c 7b 5b 9c bb f0 f1 94 d9 13 cc 12 ef 15 32 d3 28
ef 32 63 2a 48 81 a1 c0 70 1e 23 7f 04 2d
```

2.2. message_2

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

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

```
Responder's ephemeral private key
Y (Raw Value) (32 bytes)
e6 9c 23 fb f8 1b c4 35 94 24 46 83 7f e8 27 bf 20 6c 8f a1 0a 39 db
47 44 9e 5a 81 34 21 e1 e8
```

```
Responder's ephemeral public key
G_Y (Raw Value) (32 bytes)
dc 88 d2 d5 1d a5 ed 67 fc 46 16 35 6b c8 ca 74 ef 9e be 8b 38 7e 62
3a 36 0b a4 80 b9 b2 9d 1c
```

```
Responder's ephemeral public key
G_Y (CBOR Data Item) (34 bytes)
58 20 dc 88 d2 d5 1d a5 ed 67 fc 46 16 35 6b c8 ca 74 ef 9e be 8b 38
7e 62 3a 36 0b a4 80 b9 b2 9d 1c
```

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

```
Connection identifier chosen by Responder
C_R (Raw Value) (1 byte)
18
```

```
Connection identifier chosen by Responder
C_R (CBOR Data Item) (2 bytes)
41 18
```

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

TH_2 = H(G_Y, H(message_1))

```
H(message_1) (Raw Value) (32 bytes)
c1 65 d6 a9 9d 1b ca fa ac 8d bf 2b 35 2a 6f 7d 71 a3 0b 43 9c 9d 64
d3 49 a2 38 48 03 8e d1 6b
```

```
H(message_1) (CBOR Data Item) (34 bytes)
58 20 c1 65 d6 a9 9d 1b ca fa ac 8d bf 2b 35 2a 6f 7d 71 a3 0b 43 9c
9d 64 d3 49 a2 38 48 03 8e d1 6b
```

The input to calculate TH_2 is the CBOR sequence:

G_Y, H(message_1)

```
Input to calculate TH_2 (CBOR Sequence) (68 bytes)
58 20 dc 88 d2 d5 1d a5 ed 67 fc 46 16 35 6b c8 ca 74 ef 9e be 8b 38
7e 62 3a 36 0b a4 80 b9 b2 9d 1c 58 20 c1 65 d6 a9 9d 1b ca fa ac 8d
bf 2b 35 2a 6f 7d 71 a3 0b 43 9c 9d 64 d3 49 a2 38 48 03 8e d1 6b
```

```
TH_2 (Raw Value) (32 bytes)
c6 40 5c 15 4c 56 74 66 ab 1d f2 03 69 50 0e 54 0e 9f 14 bd 3a 79 6a
06 52 ca e6 6c 90 61 68 8d
```

```
TH_2 (CBOR Data Item) (34 bytes)
58 20 c6 40 5c 15 4c 56 74 66 ab 1d f2 03 69 50 0e 54 0e 9f 14 bd 3a
79 6a 06 52 ca e6 6c 90 61 68 8d
```

PRK_2e is specified in Section 4.1.1.1 of [RFC9528].

First, the Elliptic Curve Diffie-Hellman (ECDH) shared secret G_XY is computed from G_X and Y or G_Y and X:

```
G_XY (Raw Value) (ECDH shared secret) (32 bytes)
e5 cd f3 a9 86 cd ac 5b 7b f0 46 91 e2 b0 7c 08 e7 1f 53 99 8d 8f 84
2b 7c 3f b4 d8 39 cf 7b 28
```

Then, PRK_2e is calculated using EDHOC_Extract(), which is determined by the EDHOC hash algorithm:

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

where salt is TH_2:

```
salt (Raw Value) (32 bytes)
c6 40 5c 15 4c 56 74 66 ab 1d f2 03 69 50 0e 54 0e 9f 14 bd 3a 79 6a
06 52 ca e6 6c 90 61 68 8d
```

```
PRK_2e (Raw Value) (32 bytes)
d5 84 ac 2e 5d ad 5a 77 d1 4b 53 eb e7 2e f1 d5 da a8 86 0d 39 93 73
bf 2c 24 0a fa 7b a8 04 da
```

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

The Responder's signature key pair uses EdDSA:

```
Responder's private authentication key
SK_R (Raw Value) (32 bytes)
ef 14 0f f9 00 b0 ab 03 f0 c0 8d 87 9c bb d4 b3 1e a7 1e 6e 7e e7 ff
cb 7e 79 55 77 7a 33 27 99
```

```
Responder's public authentication key
PK_R (Raw Value) (32 bytes)
a1 db 47 b9 51 84 85 4a d1 2a 0c 1a 35 4e 41 8a ac e3 3a a0 f2 c6 62
c0 0b 3a c5 5d e9 2f 93 59
```

PRK_3e2m is specified in Section 4.1.1.2 of [RFC9528].

Since the Responder authenticates with signatures, PRK_3e2m = PRK_2e.

```
PRK_3e2m (Raw Value) (32 bytes)
d5 84 ac 2e 5d ad 5a 77 d1 4b 53 eb e7 2e f1 d5 da a8 86 0d 39 93 73
bf 2c 24 0a fa 7b a8 04 da
```

The Responder constructs the remaining input needed to calculate MAC_2:

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

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

CRED_R is identified by a 64-bit hash:

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

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

```
ID_CRED_R (CBOR Data Item) (14 bytes)
a1 18 22 82 2e 48 79 f2 a4 1b 51 0c 1f 9b
```

CRED_R is a CBOR byte string of the DER encoding of the X.509 certificate in Section 2.8.1:

```
CRED_R (Raw Value) (241 bytes)
30 81 ee 30 81 a1 a0 03 02 01 02 02 04 62 31 9e c4 30 05 06 03 2b 65
70 30 1d 31 1b 30 19 06 03 55 04 03 0c 12 45 44 48 4f 43 20 52 6f 6f
74 20 45 64 32 35 35 31 39 30 1e 17 0d 32 32 30 33 31 36 30 38 32 34
33 36 5a 17 0d 32 39 31 32 33 31 32 33 30 30 30 30 5a 30 22 31 20 30
1e 06 03 55 04 03 0c 17 45 44 48 4f 43 20 52 65 73 70 6f 6e 64 65 72
20 45 64 32 35 35 31 39 30 2a 30 05 06 03 2b 65 70 03 21 00 a1 db 47
b9 51 84 85 4a d1 2a 0c 1a 35 4e 41 8a ac e3 3a a0 f2 c6 62 c0 0b 3a
c5 5d e9 2f 93 59 30 05 06 03 2b 65 70 03 41 00 b7 23 bc 01 ea b0 92
8e 8b 2b 6c 98 de 19 cc 38 23 d4 6e 7d 69 87 b0 32 47 8f ec fa f1 45
37 a1 af 14 cc 8b e8 29 c6 b7 30 44 10 18 37 eb 4a bc 94 95 65 d8 6d
ce 51 cf ae 52 ab 82 c1 52 cb 02
```

```
CRED_R (CBOR Data Item) (243 bytes)
58 f1 30 81 ee 30 81 a1 a0 03 02 01 02 02 04 62 31 9e c4 30 05 06 03
2b 65 70 30 1d 31 1b 30 19 06 03 55 04 03 0c 12 45 44 48 4f 43 20 52
6f 6f 74 20 45 64 32 35 35 31 39 30 1e 17 0d 32 32 30 33 31 36 30 38
32 34 33 36 5a 17 0d 32 39 31 32 33 31 32 33 30 30 30 30 5a 30 22 31
20 30 1e 06 03 55 04 03 0c 17 45 44 48 4f 43 20 52 65 73 70 6f 6e 64
65 72 20 45 64 32 35 35 31 39 30 2a 30 05 06 03 2b 65 70 03 21 00 a1
db 47 b9 51 84 85 4a d1 2a 0c 1a 35 4e 41 8a ac e3 3a a0 f2 c6 62 c0
0b 3a c5 5d e9 2f 93 59 30 05 06 03 2b 65 70 03 41 00 b7 23 bc 01 ea
b0 92 8e 8b 2b 6c 98 de 19 cc 38 23 d4 6e 7d 69 87 b0 32 47 8f ec fa
f1 45 37 a1 af 14 cc 8b e8 29 c6 b7 30 44 10 18 37 eb 4a bc 94 95 65
d8 6d ce 51 cf ae 52 ab 82 c1 52 cb 02
```

No external authorization data:

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

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

```
context_2 (CBOR Sequence) (293 bytes)
41 18 a1 18 22 82 2e 48 79 f2 a4 1b 51 0c 1f 9b 58 20 c6 40 5c 15 4c
56 74 66 ab 1d f2 03 69 50 0e 54 0e 9f 14 bd 3a 79 6a 06 52 ca e6 6c
90 61 68 8d 58 f1 30 81 ee 30 81 a1 a0 03 02 01 02 02 04 62 31 9e c4
30 05 06 03 2b 65 70 30 1d 31 1b 30 19 06 03 55 04 03 0c 12 45 44 48
4f 43 20 52 6f 6f 74 20 45 64 32 35 35 31 39 30 1e 17 0d 32 32 30 33
31 36 30 38 32 34 33 36 5a 17 0d 32 39 31 32 33 31 32 33 30 30 30 30
5a 30 22 31 20 30 1e 06 03 55 04 03 0c 17 45 44 48 4f 43 20 52 65 73
70 6f 6e 64 65 72 20 45 64 32 35 35 31 39 30 2a 30 05 06 03 2b 65 70
03 21 00 a1 db 47 b9 51 84 85 4a d1 2a 0c 1a 35 4e 41 8a ac e3 3a a0
f2 c6 62 c0 0b 3a c5 5d e9 2f 93 59 30 05 06 03 2b 65 70 03 41 00 b7
23 bc 01 ea b0 92 8e 8b 2b 6c 98 de 19 cc 38 23 d4 6e 7d 69 87 b0 32
47 8f ec fa f1 45 37 a1 af 14 cc 8b e8 29 c6 b7 30 44 10 18 37 eb 4a
bc 94 95 65 d8 6d ce 51 cf ae 52 ab 82 c1 52 cb 02
```

```
context_2 (CBOR byte string) (296 bytes)
59 01 25 41 18 a1 18 22 82 2e 48 79 f2 a4 1b 51 0c 1f 9b 58 20 c6 40
5c 15 4c 56 74 66 ab 1d f2 03 69 50 0e 54 0e 9f 14 bd 3a 79 6a 06 52
ca e6 6c 90 61 68 8d 58 f1 30 81 ee 30 81 a1 a0 03 02 01 02 02 04 62
31 9e c4 30 05 06 03 2b 65 70 30 1d 31 1b 30 19 06 03 55 04 03 0c 12
45 44 48 4f 43 20 52 6f 6f 74 20 45 64 32 35 35 31 39 30 1e 17 0d 32
32 30 33 31 36 30 38 32 34 33 36 5a 17 0d 32 39 31 32 33 31 32 33 30
30 30 30 5a 30 22 31 20 30 1e 06 03 55 04 03 0c 17 45 44 48 4f 43 20
52 65 73 70 6f 6e 64 65 72 20 45 64 32 35 35 31 39 30 2a 30 05 06 03
2b 65 70 03 21 00 a1 db 47 b9 51 84 85 4a d1 2a 0c 1a 35 4e 41 8a ac
e3 3a a0 f2 c6 62 c0 0b 3a c5 5d e9 2f 93 59 30 05 06 03 2b 65 70 03
41 00 b7 23 bc 01 ea b0 92 8e 8b 2b 6c 98 de 19 cc 38 23 d4 6e 7d 69
87 b0 32 47 8f ec fa f1 45 37 a1 af 14 cc 8b e8 29 c6 b7 30 44 10 18
37 eb 4a bc 94 95 65 d8 6d ce 51 cf ae 52 ab 82 c1 52 cb 02
```

MAC_2 is computed through EDHOC_Expand() using the EDHOC hash algorithm (see Section 4.1.2 of [RFC9528]):

```
[
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'4118a11822822e4879f2a41b510c1f9b5820c6405c154c56
   7466ab1df20369500e540e9f14bd3a796a0652cae66c9061
   688d58f13081ee3081a1a003020102020462319ec4300506
   032b6570301d311b30190603550403 0c124544484f432052
   6f6f74204564323535313930 1e170d323230333136303832
   3433365a170d323931323331323330303030 5a3022312030
   1e06035504030c174544484f4320526573706f6e64657220
   4564323535313930 2a300506032b6570032100a1db47b951
   84854ad12a0c1a354e418aace33aa0f2c662c00b3ac55de9
   2f9359300506032b6570034100b723bc01eab0928e8b2b6c
   98de19cc3823d46e7d6987b032478fecfaf14537a1af14cc
   8be829c6b73044101837eb4abc949565d86dce51cfae52ab
   82c152cb02',
 32
)
```

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

```
info for MAC_2 (CBOR Sequence) (299 bytes)
02 59 01 25 41 18 a1 18 22 82 2e 48 79 f2 a4 1b 51 0c 1f 9b 58 20 c6
40 5c 15 4c 56 74 66 ab 1d f2 03 69 50 0e 54 0e 9f 14 bd 3a 79 6a 06
52 ca e6 6c 90 61 68 8d 58 f1 30 81 ee 30 81 a1 a0 03 02 01 02 02 04
62 31 9e c4 30 05 06 03 2b 65 70 30 1d 31 1b 30 19 06 03 55 04 03 0c
12 45 44 48 4f 43 20 52 6f 6f 74 20 45 64 32 35 35 31 39 30 1e 17 0d
32 32 30 33 31 36 30 38 32 34 33 36 5a 17 0d 32 39 31 32 33 31 32 33
30 30 30 30 5a 30 22 31 20 30 1e 06 03 55 04 03 0c 17 45 44 48 4f 43
20 52 65 73 70 6f 6e 64 65 72 20 45 64 32 35 35 31 39 30 2a 30 05 06
03 2b 65 70 03 21 00 a1 db 47 b9 51 84 85 4a d1 2a 0c 1a 35 4e 41 8a
ac e3 3a a0 f2 c6 62 c0 0b 3a c5 5d e9 2f 93 59 30 05 06 03 2b 65 70
03 41 00 b7 23 bc 01 ea b0 92 8e 8b 2b 6c 98 de 19 cc 38 23 d4 6e 7d
69 87 b0 32 47 8f ec fa f1 45 37 a1 af 14 cc 8b e8 29 c6 b7 30 44 10
18 37 eb 4a bc 94 95 65 d8 6d ce 51 cf ae 52 ab 82 c1 52 cb 02 18 20
```

```
MAC_2 (Raw Value) (32 bytes)
86 2a 7e 5e f1 47 f9 a5 f4 c5 12 e1 b6 62 3c d6 6c d1 7a 72 72 07 2b
fe 5b 60 2f fe 30 7e e0 e9
```

```
MAC_2 (CBOR Data Item) (34 bytes)
58 20 86 2a 7e 5e f1 47 f9 a5 f4 c5 12 e1 b6 62 3c d6 6c d1 7a 72 72
07 2b fe 5b 60 2f fe 30 7e e0 e9
```

Since METHOD = 0, Signature_or_MAC_2 is the 'signature' of the COSE_Sign1 object.

The Responder constructs the message to be signed:

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

[
 "Signature1",
 h'a11822822e4879f2a41b510c1f9b',
 h'5820c6405c154c567466ab1df20369500e540e9f14bd3a79
   6a0652cae66c9061688d58f13081ee3081a1a00302010202
   0462319ec4300506032b6570301d311b301906035504030c
   124544484f4320526f6f7420456432353139301e170d32
   3230333136303823433365a170d32393132333132333030
   30305a30223120301e06035504030c174544484f43205265
   73706f6e646572204564323535139302a300506032b6570
   032100a1db47b95184854ad12a0c1a354e418aace33aa0f2
   c662c00b3ac55de92f9359300506032b6570034100b723bc
   01eab0928e8b2b6c98de19cc3823d46e7d6987b032478fec
   faf14537a1af14cc8be829c6b73044101837eb4abc949565
   d86dce51cfae52ab82c152cb02',
 h'862a7e5ef147f9a5f4c512e1b6623cd66cd17a7272072bfe
   5b602ffe307ee0e9'
]
```

```
Message to be signed 2 (CBOR Data Item) (341 bytes)
84 6a 53 69 67 6e 61 74 75 72 65 31 4e a1 18 22 82 2e 48 79 f2 a4 1b
51 0c 1f 9b 59 01 15 58 20 c6 40 5c 15 4c 56 74 66 ab 1d f2 03 69 50
0e 54 0e 9f 14 bd 3a 79 6a 06 52 ca e6 6c 90 61 68 8d 58 f1 30 81 ee
30 81 a1 a0 03 02 01 02 02 04 62 31 9e c4 30 05 06 03 2b 65 70 30 1d
31 1b 30 19 06 03 55 04 03 0c 12 45 44 48 4f 43 20 52 6f 6f 74 20 45
64 32 35 35 31 39 30 1e 17 0d 32 32 30 33 31 36 30 38 32 34 33 36 5a
17 0d 32 39 31 32 33 31 32 33 30 30 30 30 5a 30 22 31 20 30 1e 06 03
55 04 03 0c 17 45 44 48 4f 43 20 52 65 73 70 6f 6e 64 65 72 20 45 64
32 35 35 31 39 30 2a 30 05 06 03 2b 65 70 03 21 00 a1 db 47 b9 51 84
85 4a d1 2a 0c 1a 35 4e 41 8a ac e3 3a a0 f2 c6 62 c0 0b 3a c5 5d e9
2f 93 59 30 05 06 03 2b 65 70 03 41 00 b7 23 bc 01 ea b0 92 8e 8b 2b
6c 98 de 19 cc 38 23 d4 6e 7d 69 87 b0 32 47 8f ec fa f1 45 37 a1 af
14 cc 8b e8 29 c6 b7 30 44 10 18 37 eb 4a bc 94 95 65 d8 6d ce 51 cf
ae 52 ab 82 c1 52 cb 02 58 20 86 2a 7e 5e f1 47 f9 a5 f4 c5 12 e1 b6
62 3c d6 6c d1 7a 72 72 07 2b fe 5b 60 2f fe 30 7e e0 e9
```

The Responder signs using the private authentication key SK_R.

```
Signature_or_MAC_2 (Raw Value) (64 bytes)
c3 b5 bd 44 d1 e4 4a 08 5c 03 d3 ae de 4e 1e 6c 11 c5 72 a1 96 8c c3
62 9b 50 5f 98 c6 81 60 8d 3d 1d e7 93 d1 c4 0e b5 dd 5d 89 ac f1 96
6a ea 07 02 2b 48 cd c9 98 70 eb c4 03 74 e8 fa 6e 09
```

```
Signature_or_MAC_2 (CBOR Data Item) (66 bytes)
58 40 c3 b5 bd 44 d1 e4 4a 08 5c 03 d3 ae de 4e 1e 6c 11 c5 72 a1 96
8c c3 62 9b 50 5f 98 c6 81 60 8d 3d 1d e7 93 d1 c4 0e b5 dd 5d 89 ac
f1 96 6a ea 07 02 2b 48 cd c9 98 70 eb c4 03 74 e8 fa 6e 09
```

The Responder constructs PLAINTEXT_2:

```
PLAINTEXT_2 =
(
 C_R,
 ID_CRED_R / bstr / -24..23,
 Signature_or_MAC_2,
 ? EAD_2
)
```

```
PLAINTEXT_2 (CBOR Sequence) (82 bytes)
41 18 a1 18 22 82 2e 48 79 f2 a4 1b 51 0c 1f 9b 58 40 c3 b5 bd 44 d1
e4 4a 08 5c 03 d3 ae de 4e 1e 6c 11 c5 72 a1 96 8c c3 62 9b 50 5f 98
c6 81 60 8d 3d 1d e7 93 d1 c4 0e b5 dd 5d 89 ac f1 96 6a ea 07 02 2b
48 cd c9 98 70 eb c4 03 74 e8 fa 6e 09
```

The input needed to calculate KEYSTREAM_2 is defined in Section 4.1.2 of [RFC9528], using EDHOC_Expand() with the EDHOC hash algorithm:

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

where plaintext_length is the length in bytes of PLAINTEXT_2 in bytes and info for KEYSTREAM_2 is:

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

where the last value is the length in bytes of PLAINTEXT_2.

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

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

The Responder calculates CIPHERTEXT_2 as XOR between PLAINTEXT_2 and KEYSTREAM_2:

```
CIPHERTEXT_2 (Raw Value) (82 bytes)
bc 26 dd 27 0f e9 c0 2c 44 ce 39 34 79 4b 1c c6 2b a2 2f 05 45 9f 8d
35 8c 8d 12 27 5a c4 2c 5f 96 de d5 f1 3c c9 08 4e 5b 20 18 89 a4 5e
5a 60 a5 56 2d c1 18 61 9c 3d aa 2f d9 f4 c9 f4 d6 ed ad 10 9d d4 ed
f9 59 62 aa fb af 9a b3 f4 a1 f6 b9 8f
```

The Responder constructs message_2:

```
message_2 =
(
 G_Y_CIPHERTEXT_2
)
```

where G_Y_CIPHERTEXT_2 is the bstr encoding of the concatenation of the raw values of G_Y and CIPHERTEXT_2.

```
message_2 (CBOR Sequence) (116 bytes)
58 72 dc 88 d2 d5 1d a5 ed 67 fc 46 16 35 6b c8 ca 74 ef 9e be 8b 38
7e 62 3a 36 0b a4 80 b9 b2 9d 1c bc 26 dd 27 0f e9 c0 2c 44 ce 39 34
79 4b 1c c6 2b a2 2f 05 45 9f 8d 35 8c 8d 12 27 5a c4 2c 5f 96 de d5
f1 3c c9 08 4e 5b 20 18 89 a4 5e 5a 60 a5 56 2d c1 18 61 9c 3d aa 2f
d9 f4 c9 f4 d6 ed ad 10 9d d4 ed f9 59 62 aa fb af 9a b3 f4 a1 f6 b9
8f
```

2.3. message_3

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

The Initiator's signature key pair uses 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 [RFC9528].

Since the Initiator authenticates with signatures, PRK_4e3m = PRK_3e2m.

```
PRK_4e3m (Raw Value) (32 bytes)
d5 84 ac 2e 5d ad 5a 77 d1 4b 53 eb e7 2e f1 d5 da a8 86 0d 39 93 73
bf 2c 24 0a fa 7b a8 04 da
```

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

TH_3 = H(TH_2, PLAINTEXT_2, CRED_R)

```
Input to calculate TH_3 (CBOR Sequence) (359 bytes)
58 20 c6 40 5c 15 4c 56 74 66 ab 1d f2 03 69 50 0e 54 0e 9f 14 bd 3a
79 6a 06 52 ca e6 6c 90 61 68 8d 41 18 a1 18 22 82 2e 48 79 f2 a4 1b
51 0c 1f 9b 58 40 c3 b5 bd 44 d1 e4 4a 08 5c 03 d3 ae de 4e 1e 6c 11
c5 72 a1 96 8c c3 62 9b 50 5f 98 c6 81 60 8d 3d 1d e7 93 d1 c4 0e b5
dd 5d 89 ac f1 96 6a ea 07 02 2b 48 cd c9 98 70 eb c4 03 74 e8 fa 6e
09 58 f1 30 81 ee 30 81 a1 a0 03 02 01 02 02 04 62 31 9e c4 30 05 06
03 2b 65 70 30 1d 31 1b 30 19 06 03 55 04 03 0c 12 45 44 48 4f 43 20
52 6f 6f 74 20 45 64 32 35 35 31 39 30 1e 17 0d 32 32 30 33 31 36 30
38 32 34 33 36 5a 17 0d 32 39 31 32 33 31 32 33 30 30 30 30 5a 30 22
31 20 30 1e 06 03 55 04 03 0c 17 45 44 48 4f 43 20 52 65 73 70 6f 6e
64 65 72 20 45 64 32 35 35 31 39 30 2a 30 05 06 03 2b 65 70 03 21 00
a1 db 47 b9 51 84 85 4a d1 2a 0c 1a 35 4e 41 8a ac e3 3a a0 f2 c6 62
c0 0b 3a c5 5d e9 2f 93 59 30 05 06 03 2b 65 70 03 41 00 b7 23 bc 01
ea b0 92 8e 8b 2b 6c 98 de 19 cc 38 23 d4 6e 7d 69 87 b0 32 47 8f ec
fa f1 45 37 a1 af 14 cc 8b e8 29 c6 b7 30 44 10 18 37 eb 4a bc 94 95
65 d8 6d ce 51 cf ae 52 ab 82 c1 52 cb 02
```

```
TH_3 (Raw Value) (32 bytes)
5b 7d f9 b4 f5 8f 24 0c e0 41 8e 48 19 1b 5f ff 3a 22 b5 ca 57 f6 69
b1 67 77 99 65 92 e9 28 bc
```

```
TH_3 (CBOR Data Item) (34 bytes)
58 20 5b 7d f9 b4 f5 8f 24 0c e0 41 8e 48 19 1b 5f ff 3a 22 b5 ca 57
f6 69 b1 67 77 99 65 92 e9 28 bc
```

The Initiator constructs the remaining input needed to calculate MAC_3:

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

where

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

CRED_I is identified by a 64-bit hash:

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

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

```
ID_CRED_I (CBOR Data Item) (14 bytes)
a1 18 22 82 2e 48 c2 4a b2 fd 76 43 c7 9f
```

CRED_I is a CBOR byte string of the DER encoding of the X.509 certificate in Section 2.8.2:

```
CRED_I (Raw Value) (241 bytes)
30 81 ee 30 81 a1 a0 03 02 01 02 02 04 62 31 9e a0 30 05 06 03 2b 65
70 30 1d 31 1b 30 19 06 03 55 04 03 0c 12 45 44 48 4f 43 20 52 6f 6f
74 20 45 64 32 35 35 31 39 30 1e 17 0d 32 32 30 33 31 36 30 38 32 34
30 30 5a 17 0d 32 39 31 32 33 31 32 33 30 30 30 30 5a 30 22 31 20 30
1e 06 03 55 04 03 0c 17 45 44 48 4f 43 20 49 6e 69 74 69 61 74 6f 72
20 45 64 32 35 35 31 39 30 2a 30 05 06 03 2b 65 70 03 21 00 ed 06 a8
ae 61 a8 29 ba 5f a5 45 25 c9 d0 7f 48 dd 44 a3 02 f4 3e 0f 23 d8 cc
20 b7 30 85 14 1e 30 05 06 03 2b 65 70 03 41 00 52 12 41 d8 b3 a7 70
99 6b cf c9 b9 ea d4 e7 e0 a1 c0 db 35 3a 3b df 29 10 b3 92 75 ae 48
b7 56 01 59 81 85 0d 27 db 67 34 e3 7f 67 21 22 67 dd 05 ee ff 27 b9
e7 a8 13 fa 57 4b 72 a0 0b 43 0b
```

```
CRED_I (CBOR Data Item) (243 bytes)
58 f1 30 81 ee 30 81 a1 a0 03 02 01 02 02 04 62 31 9e a0 30 05 06 03
2b 65 70 30 1d 31 1b 30 19 06 03 55 04 03 0c 12 45 44 48 4f 43 20 52
6f 6f 74 20 45 64 32 35 35 31 39 30 1e 17 0d 32 32 30 33 31 36 30 38
32 34 30 30 5a 17 0d 32 39 31 32 33 31 32 33 30 30 30 30 5a 30 22 31
20 30 1e 06 03 55 04 03 0c 17 45 44 48 4f 43 20 49 6e 69 74 69 61 74
6f 72 20 45 64 32 35 35 31 39 30 2a 30 05 06 03 2b 65 70 03 21 00 ed
06 a8 ae 61 a8 29 ba 5f a5 45 25 c9 d0 7f 48 dd 44 a3 02 f4 3e 0f 23
d8 cc 20 b7 30 85 14 1e 30 05 06 03 2b 65 70 03 41 00 52 12 41 d8 b3
a7 70 99 6b cf c9 b9 ea d4 e7 e0 a1 c0 db 35 3a 3b df 29 10 b3 92 75
ae 48 b7 56 01 59 81 85 0d 27 db 67 34 e3 7f 67 21 22 67 dd 05 ee ff
27 b9 e7 a8 13 fa 57 4b 72 a0 0b 43 0b
```

No external authorization data:

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

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

```
context_3 (CBOR Sequence) (291 bytes)
a1 18 22 82 2e 48 c2 4a b2 fd 76 43 c7 9f 58 20 5b 7d f9 b4 f5 8f 24
0c e0 41 8e 48 19 1b 5f ff 3a 22 b5 ca 57 f6 69 b1 67 77 99 65 92 e9
28 bc 58 f1 30 81 ee 30 81 a1 a0 03 02 01 02 02 04 62 31 9e a0 30 05
06 03 2b 65 70 30 1d 31 1b 30 19 06 03 55 04 03 0c 12 45 44 48 4f 43
20 52 6f 6f 74 20 45 64 32 35 35 31 39 30 1e 17 0d 32 32 30 33 31 36
30 38 32 34 30 30 5a 17 0d 32 39 31 32 33 31 32 33 30 30 30 30 5a 30
22 31 20 30 1e 06 03 55 04 03 0c 17 45 44 48 4f 43 20 49 6e 69 74 69
61 74 6f 72 20 45 64 32 35 35 31 39 30 2a 30 05 06 03 2b 65 70 03 21
00 ed 06 a8 ae 61 a8 29 ba 5f a5 45 25 c9 d0 7f 48 dd 44 a3 02 f4 3e
0f 23 d8 cc 20 b7 30 85 14 1e 30 05 06 03 2b 65 70 03 41 00 52 12 41
d8 b3 a7 70 99 6b cf c9 b9 ea d4 e7 e0 a1 c0 db 35 3a 3b df 29 10 b3
92 75 ae 48 b7 56 01 59 81 85 0d 27 db 67 34 e3 7f 67 21 22 67 dd 05
ee ff 27 b9 e7 a8 13 fa 57 4b 72 a0 0b 43 0b
```

```
context_3 (CBOR byte string) (294 bytes)
59 01 23 a1 18 22 82 2e 48 c2 4a b2 fd 76 43 c7 9f 58 20 5b 7d f9 b4
f5 8f 24 0c e0 41 8e 48 19 1b 5f ff 3a 22 b5 ca 57 f6 69 b1 67 77 99
65 92 e9 28 bc 58 f1 30 81 ee 30 81 a1 a0 03 02 01 02 02 04 62 31 9e
a0 30 05 06 03 2b 65 70 30 1d 31 1b 30 19 06 03 55 04 03 0c 12 45 44
48 4f 43 20 52 6f 6f 74 20 45 64 32 35 35 31 39 30 1e 17 0d 32 32 30
33 31 36 30 38 32 34 30 30 5a 17 0d 32 39 31 32 33 31 32 33 30 30 30
30 5a 30 22 31 20 30 1e 06 03 55 04 03 0c 17 45 44 48 4f 43 20 49 6e
69 74 69 61 74 6f 72 20 45 64 32 35 35 31 39 30 2a 30 05 06 03 2b 65
70 03 21 00 ed 06 a8 ae 61 a8 29 ba 5f a5 45 25 c9 d0 7f 48 dd 44 a3
02 f4 3e 0f 23 d8 cc 20 b7 30 85 14 1e 30 05 06 03 2b 65 70 03 41 00
52 12 41 d8 b3 a7 70 99 6b cf c9 b9 ea d4 e7 e0 a1 c0 db 35 3a 3b df
29 10 b3 92 75 ae 48 b7 56 01 59 81 85 0d 27 db 67 34 e3 7f 67 21 22
67 dd 05 ee ff 27 b9 e7 a8 13 fa 57 4b 72 a0 0b 43 0b
```

MAC_3 is computed through EDHOC_Expand() using the EDHOC hash algorithm (see Section 4.1.2 of [RFC9528]):

```
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'a11822822e48c24ab2fd7643c79f58205b7df9b4f58f240c
   e0418e48191b5fff3a22b5ca57f669b16777996592e928bc
   58f13081ee3081a1a003020102020462319ea0300506032b
   6570301d311b301906035504030c124544484f4320526f6f
   7420456432353531393 01e170d323230333 1363038323430
   305a170d323931323331323330303030 5a30223120301e06
   035504030c174544484f4320496e69746961746f72204564
   3235353139302a300506032b6570032100ed06a8ae61a829
   ba5fa54525c9d07f48dd44a302f43e0f23d8cc20b7308514
   1e300506032b6570034100521241d8b3a770996bcfc9b9ea
   d4e7e0a1c0db353a3bdf2910b39275ae48b75601598 1850d
   27db6734e37f67212267dd05eeff27b9e7a813fa574b72a0
   0b430b',
 32
)
```

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

```
info for MAC_3 (CBOR Sequence) (297 bytes)
06 59 01 23 a1 18 22 82 2e 48 c2 4a b2 fd 76 43 c7 9f 58 20 5b 7d f9
b4 f5 8f 24 0c e0 41 8e 48 19 1b 5f ff 3a 22 b5 ca 57 f6 69 b1 67 77
99 65 92 e9 28 bc 58 f1 30 81 ee 30 81 a1 a0 03 02 01 02 02 04 62 31
9e a0 30 05 06 03 2b 65 70 30 1d 31 1b 30 19 06 03 55 04 03 0c 12 45
44 48 4f 43 20 52 6f 6f 74 20 45 64 32 35 35 31 39 30 1e 17 0d 32 32
30 33 31 36 30 38 32 34 30 30 5a 17 0d 32 39 31 32 33 31 32 33 30 30
30 30 5a 30 22 31 20 30 1e 06 03 55 04 03 0c 17 45 44 48 4f 43 20 49
6e 69 74 69 61 74 6f 72 20 45 64 32 35 35 31 39 30 2a 30 05 06 03 2b
65 70 03 21 00 ed 06 a8 ae 61 a8 29 ba 5f a5 45 25 c9 d0 7f 48 dd 44
a3 02 f4 3e 0f 23 d8 cc 20 b7 30 85 14 1e 30 05 06 03 2b 65 70 03 41
00 52 12 41 d8 b3 a7 70 99 6b cf c9 b9 ea d4 e7 e0 a1 c0 db 35 3a 3b
df 29 10 b3 92 75 ae 48 b7 56 01 59 81 85 0d 27 db 67 34 e3 7f 67 21
22 67 dd 05 ee ff 27 b9 e7 a8 13 fa 57 4b 72 a0 0b 43 0b 18 20
```

```
MAC_3 (Raw Value) (32 bytes)
39 b1 27 c1 30 12 9a fa 30 61 8c 75 13 29 e6 37 cc 37 34 27 0d 4b 01
25 84 45 a8 ee 02 da a3 bd
```

```
MAC_3 (CBOR Data Item) (34 bytes)
58 20 39 b1 27 c1 30 12 9a fa 30 61 8c 75 13 29 e6 37 cc 37 34 27 0d 4b
01 25 84 45 a8 ee 02 da a3 bd
```

Since METHOD = 0, Signature_or_MAC_3 is the 'signature' of the COSE_Sign1 object.

The Initiator constructs the message to be signed:

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

[
 "Signature1",
 h'a11822822e48c24ab2fd7643c79f',
 h'58205b7df9b4f58f240ce0418e48191b5fff3a22b5ca57f6
   69b16777996592e928bc58f13081ee3081a1a00302010202
   0462319ea0300506032b6570301d311b301906035504030c
   124544484f4320526f6f74204564323535313930 1e170d32
   323033313630383234303035a170d32393132333132333030
   30305a30223120301e06035504030c174544484f4320496e
   69746961746f72204564323535313930 2a300506032b6570
   032100ed06a8ae61a829ba5fa54525c9d07f48dd44a302f4
   3e0f23d8cc20b73085141e300506032b6570034100521241
   d8b3a770996bcfc9b9ead4e7e0a1c0db353a3bdf2910b392
   75ae48b756015981850d27db6734e37f67212267dd05eeff
   27b9e7a813fa574b72a00b430b',
 h'39b127c130129afa30618c751329e637cc3734270d4b0125
   8445a8ee02daa3bd'
]
```

```
Message to be signed 3 (CBOR Data Item) (341 bytes)
84 6a 53 69 67 6e 61 74 75 72 65 31 4e a1 18 22 82 2e 48 c2 4a b2 fd
76 43 c7 9f 59 01 15 58 20 5b 7d f9 b4 f5 8f 24 0c e0 41 8e 48 19 1b
5f ff 3a 22 b5 ca 57 f6 69 b1 67 77 99 65 92 e9 28 bc 58 f1 30 81 ee
30 81 a1 a0 03 02 01 02 02 04 62 31 9e a0 30 05 06 03 2b 65 70 30 1d
31 1b 30 19 06 03 55 04 03 0c 12 45 44 48 4f 43 20 52 6f 6f 74 20 45
64 32 35 35 31 39 30 1e 17 0d 32 32 30 33 31 36 30 38 32 34 30 30 5a
17 0d 32 39 31 32 33 31 32 33 30 30 30 30 5a 30 22 31 20 30 1e 06 03
55 04 03 0c 17 45 44 48 4f 43 20 49 6e 69 74 69 61 74 6f 72 20 45 64
32 35 35 31 39 30 2a 30 05 06 03 2b 65 70 03 21 00 ed 06 a8 ae 61 a8
29 ba 5f a5 45 25 c9 d0 7f 48 dd 44 a3 02 f4 3e 0f 23 d8 cc 20 b7 30
85 14 1e 30 05 06 03 2b 65 70 03 41 00 52 12 41 d8 b3 a7 70 99 6b cf
c9 b9 ea d4 e7 e0 a1 c0 db 35 3a 3b df 29 10 b3 92 75 ae 48 b7 56 01
59 81 85 0d 27 db 67 34 e3 7f 67 21 22 67 dd 05 ee ff 27 b9 e7 a8 13
fa 57 4b 72 a0 0b 43 0b 58 20 39 b1 27 c1 30 12 9a fa 30 61 8c 75 13
29 e6 37 cc 37 34 27 0d 4b 01 25 84 45 a8 ee 02 da a3 bd
```

The Initiator signs using the private authentication key SK_I:

```
Signature_or_MAC_3 (Raw Value) (64 bytes)
96 e1 cd 5f ce ad fa c1 b5 af 81 94 43 f7 09 24 f5 71 99 55 95 7f d0
26 55 be b4 77 5e 1a 73 18 6a 0d 1d 3e a6 83 f0 8f 8d 03 dc ec b9 cf
15 4e 1c 6f 55 5a 1e 12 ca 11 8c e4 2b db a6 87 89 07
```

```
Signature_or_MAC_3 (CBOR Data Item) (66 bytes)
58 40 96 e1 cd 5f ce ad fa c1 b5 af 81 94 43 f7 09 24 f5 71 99 55 95
7f d0 26 55 be b4 77 5e 1a 73 18 6a 0d 1d 3e a6 83 f0 8f 8d 03 dc ec
b9 cf 15 4e 1c 6f 55 5a 1e 12 ca 11 8c e4 2b db a6 87 89 07
```

The Initiator constructs PLAINTEXT_3:

```
PLAINTEXT_3 =
(
 ID_CRED_I / bstr / -24..23,
 Signature_or_MAC_3,
 ? EAD_3
)
```

```
PLAINTEXT_3 (CBOR Sequence) (80 bytes)
a1 18 22 82 2e 48 c2 4a b2 fd 76 43 c7 9f 58 40 96 e1 cd 5f ce ad fa
c1 b5 af 81 94 43 f7 09 24 f5 71 99 55 95 7f d0 26 55 be b4 77 5e 1a
73 18 6a 0d 1d 3e a6 83 f0 8f 8d 03 dc ec b9 cf 15 4e 1c 6f 55 5a 1e
12 ca 11 8c e4 2b db a6 87 89 07
```

The Initiator constructs the associated data for message_3:

```
A_3 =
[
 "Encrypt0",
 h'',
 h'5b7df9b4f58f240ce0418e48191b5fff3a22b5ca57f669b1
   6777996592e928bc'
]
```

```
A_3 (CBOR Data Item) (45 bytes)
83 68 45 6e 63 72 79 70 74 30 40 58 20 5b 7d f9 b4 f5 8f 24 0c e0 41
8e 48 19 1b 5f ff 3a 22 b5 ca 57 f6 69 b1 67 77 99 65 92 e9 28 bc
```

The Initiator constructs the input needed to derive the key K_3 (see Section 4.1.2 of [RFC9528]) using the EDHOC hash algorithm:

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

where key_length is the key length in bytes for the EDHOC Authenticated Encryption with Associated Data (AEAD) algorithm, and info for K_3 is:

```
info =
(
 3,
 h'5b7df9b4f58f240ce0418e48191b5fff3a22b5ca57f669b1
   6777996592e928bc',
 16
)
```

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

```
info for K_3 (CBOR Sequence) (36 bytes)
03 58 20 5b 7d f9 b4 f5 8f 24 0c e0 41 8e 48 19 1b 5f ff 3a 22 b5 ca
57 f6 69 b1 67 77 99 65 92 e9 28 bc 10
```

```
K_3 (Raw Value) (16 bytes)
da 19 5e 5f 64 8a c6 3b 0e 8f b0 c4 55 20 51 39
```

The Initiator constructs the input needed to derive the nonce IV_3 (see Section 4.1.2 of [RFC9528]) using the EDHOC hash algorithm:

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

where iv_length is the nonce length in bytes for the EDHOC AEAD algorithm, and info for IV_3 is:

```
info =
(
 4,
 h'5b7df9b4f58f240ce0418e48191b5fff3a22b5ca57f669b1
   6777996592e928bc',
 13
)
```

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

```
info for IV_3 (CBOR Sequence) (36 bytes)
04 58 20 5b 7d f9 b4 f5 8f 24 0c e0 41 8e 48 19 1b 5f ff 3a 22 b5 ca
57 f6 69 b1 67 77 99 65 92 e9 28 bc 0d
```

```
IV_3 (Raw Value) (13 bytes)
38 d8 c6 4c 56 25 5a ff a4 49 f4 be d7
```

The Initiator calculates CIPHERTEXT_3 as 'ciphertext' of COSE_Encrypt0 applied using the EDHOC AEAD algorithm with plaintext PLAINTEXT_3, additional data A_3, key K_3, and nonce IV_3.

```
CIPHERTEXT_3 (Raw Value) (88 bytes)
25 c3 45 88 4a aa eb 22 c5 27 f9 b1 d2 b6 78 72 07 e0 16 3c 69 b6 2a
0d 43 92 81 50 42 72 03 c3 16 74 e4 51 4e a6 e3 83 b5 66 eb 29 76 3e
fe b0 af a5 18 77 6a e1 c6 5f 85 6d 84 bf 32 af 3a 78 36 97 04 66 dc
b7 1f 76 74 5d 39 d3 02 5e 77 03 e0 c0 32 eb ad 51 94 7c
```

message_3 is the CBOR bstr encoding of CIPHERTEXT_3:

```
message_3 (CBOR Sequence) (90 bytes)
58 58 25 c3 45 88 4a aa eb 22 c5 27 f9 b1 d2 b6 78 72 07 e0 16 3c 69
b6 2a 0d 43 92 81 50 42 72 03 c3 16 74 e4 51 4e a6 e3 83 b5 66 eb 29
76 3e fe b0 af a5 18 77 6a e1 c6 5f 85 6d 84 bf 32 af 3a 78 36 97 04
66 dc b7 1f 76 74 5d 39 d3 02 5e 77 03 e0 c0 32 eb ad 51 94 7c
```

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

TH_4 = H(TH_3, PLAINTEXT_3, CRED_I)

```
Input to calculate TH_4 (CBOR Sequence) (357 bytes)
58 20 5b 7d f9 b4 f5 8f 24 0c e0 41 8e 48 19 1b 5f ff 3a 22 b5 ca 57
f6 69 b1 67 77 99 65 92 e9 28 bc a1 18 22 82 2e 48 c2 4a b2 fd 76 43
c7 9f 58 40 96 e1 cd 5f ce ad fa c1 b5 af 81 94 43 f7 09 24 f5 71 99
55 95 7f d0 26 55 be b4 77 5e 1a 73 18 6a 0d 1d 3e a6 83 f0 8f 8d 03
dc ec b9 cf 15 4e 1c 6f 55 5a 1e 12 ca 11 8c e4 2b db a6 87 89 07 58
f1 30 81 ee 30 81 a1 a0 03 02 01 02 02 04 62 31 9e a0 30 05 06 03 2b
65 70 30 1d 31 1b 30 19 06 03 55 04 03 0c 12 45 44 48 4f 43 20 52 6f
6f 74 20 45 64 32 35 35 31 39 30 1e 17 0d 32 32 30 33 31 36 30 38 32
34 30 30 5a 17 0d 32 39 31 32 33 31 32 33 30 30 30 30 5a 30 22 31 20
30 1e 06 03 55 04 03 0c 17 45 44 48 4f 43 20 49 6e 69 74 69 61 74 6f
72 20 45 64 32 35 35 31 39 30 2a 30 05 06 03 2b 65 70 03 21 00 ed 06
a8 ae 61 a8 29 ba 5f a5 45 25 c9 d0 7f 48 dd 44 a3 02 f4 3e 0f 23 d8
cc 20 b7 30 85 14 1e 30 05 06 03 2b 65 70 03 41 00 52 12 41 d8 b3 a7
70 99 6b cf c9 b9 ea d4 e7 e0 a1 c0 db 35 3a 3b df 29 10 b3 92 75 ae
48 b7 56 01 59 81 85 0d 27 db 67 34 e3 7f 67 21 22 67 dd 05 ee ff 27
b9 e7 a8 13 fa 57 4b 72 a0 0b 43 0b
```

```
TH_4 (Raw Value) (32 bytes)
0e b8 68 f2 63 cf 35 55 dc cd 39 6d d8 de c2 9d 37 50 d5 99 be 42 d5
a4 1a 5a 37 c8 96 f2 94 ac
```

```
TH_4 (CBOR Data Item) (34 bytes)
58 20 0e b8 68 f2 63 cf 35 55 dc cd 39 6d d8 de c2 9d 37 50 d5 99 be
42 d5 a4 1a 5a 37 c8 96 f2 94 ac
```

2.4. message_4

No external authorization data:

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

The Responder constructs PLAINTEXT_4:

```
PLAINTEXT_4 =
(
 ? EAD_4
)
```

```
PLAINTEXT_4 (CBOR Sequence) (0 bytes)
```

The Responder constructs the associated data for message_4:

```
A_4 =
[
 "Encrypt0",
 h'',
 h'0eb868f263cf3555dccd396dd8dec29d3750d599be42d5a4
   1a5a37c896f294ac'
]
```

```
A_4 (CBOR Data Item) (45 bytes)
83 68 45 6e 63 72 79 70 74 30 40 58 20 0e b8 68 f2 63 cf 35 55 dc cd
39 6d d8 de c2 9d 37 50 d5 99 be 42 d5 a4 1a 5a 37 c8 96 f2 94 ac
```

The Responder constructs the input needed to derive the EDHOC message_4 key (see Section 4.1.2 of [RFC9528]) using the EDHOC hash algorithm:

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

where key_length is the key length in bytes for the EDHOC AEAD algorithm, and info for K_4 is:

```
info =
(
 8,
 h'0eb868f263cf3555dccd396dd8dec29d3750d599be42d5a4
   1a5a37c896f294ac',
 16
)
```

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

```
info for K_4 (CBOR Sequence) (36 bytes)
08 58 20 0e b8 68 f2 63 cf 35 55 dc cd 39 6d d8 de c2 9d 37 50 d5 99
be 42 d5 a4 1a 5a 37 c8 96 f2 94 ac 10
```

```
K_4 (Raw Value) (16 bytes)
df 8c b5 86 1e 1f df ed d3 b2 30 15 a3 9d 1e 2e
```

The Responder constructs the input needed to derive the EDHOC message_4 nonce (see Section 4.1.2 of [RFC9528]) using the EDHOC hash algorithm:

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

where length is the nonce length in bytes for the EDHOC AEAD algorithm, and info for IV_4 is:

```
info =
(
 9,
 h'0eb868f263cf3555dccd396dd8dec29d3750d599be42d5a4
   1a5a37c896f294ac',
 13
)
```

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

```
info for IV_4 (CBOR Sequence) (36 bytes)
09 58 20 0e b8 68 f2 63 cf 35 55 dc cd 39 6d d8 de c2 9d 37 50 d5 99
be 42 d5 a4 1a 5a 37 c8 96 f2 94 ac 0d
```

```
IV_4 (Raw Value) (13 bytes)
12 8e c6 58 d9 70 d7 38 0f 74 fc 6c 27
```

The Responder calculates CIPHERTEXT_4 as 'ciphertext' of COSE_Encrypt0 applied using the EDHOC AEAD algorithm with plaintext PLAINTEXT_4, additional data A_4, key K_4, and nonce IV_4.

```
CIPHERTEXT_4 (8 bytes)
4f 0e de e3 66 e5 c8 83
```

message_4 is the CBOR bstr encoding of CIPHERTEXT_4:

```
message_4 (CBOR Sequence) (9 bytes)
48 4f 0e de e3 66 e5 c8 83
```

2.5. PRK_out and PRK_exporter

PRK_out is specified in Section 4.1.3 of [RFC9528].

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

where hash_length is the length in bytes of the output of the EDHOC hash algorithm, and info for PRK_out is:

```
info =
(
 7,
 h'0eb868f263cf3555dccd396dd8dec29d3750d599be42d5a4
   1a5a37c896f294ac',
 32
)
```

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

```
info for PRK_out (CBOR Sequence) (37 bytes)
07 58 20 0e b8 68 f2 63 cf 35 55 dc cd 39 6d d8 de c2 9d 37 50 d5 99
be 42 d5 a4 1a 5a 37 c8 96 f2 94 ac 18 20
```

```
PRK_out (Raw Value) (32 bytes)
b7 44 cb 7d 8a 87 cc 04 47 c3 35 0e 16 5b 25 0d ab 12 ec 45 33 25 ab
b9 22 b3 03 07 e5 c3 68 f0
```

The Object Security for Constrained RESTful Environments (OSCORE) Master Secret and OSCORE Master Salt are derived with the EDHOC_Exporter as specified in Section 4.2.1 of [RFC9528].

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

where PRK_exporter is derived from PRK_out:

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

where hash_length is the length in bytes of the output of the EDHOC hash algorithm, and info for the PRK_exporter is:

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

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

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

```
PRK_exporter (Raw Value) (32 bytes)
2a ae c8 fc 4a b3 bc 32 95 de f6 b5 51 05 1a 2f a5 61 42 4d b3 01 fa
84 f6 42 f5 57 8a 6d f5 1a
```

2.6. OSCORE Parameters

The derivation of OSCORE parameters is specified in Appendix A.1 of [RFC9528].

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 [RFC9528].

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

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

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

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

The OSCORE Master Secret is computed through EDHOC_Expand() using the application hash algorithm (see Appendix A.1 of [RFC9528]):

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

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

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

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

```
OSCORE Master Secret (Raw Value) (16 bytes)
1e 1c 6b ea c3 a8 a1 ca c4 35 de 7e 2f 9a e7 ff
```

The OSCORE Master Salt is computed through EDHOC_Expand() using the application hash algorithm (see Section 4.2 of [RFC9528]):

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

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

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

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

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

```
OSCORE Master Salt (Raw Value) (8 bytes)
ce 7a b8 44 c0 10 6d 73
```

2.7. Key Update

Key update is defined in Appendix H of [RFC9528].

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

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

where info for key update is:

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

```
PRK_out after KeyUpdate (Raw Value) (32 bytes)
da 6e ac d9 a9 85 f4 fb a9 ae c2 a9 29 90 22 97 6b 25 b1 4e 89 fa 15
97 94 f2 8d 82 fa f2 da ad
```

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

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

where info and hash_length are unchanged as in Section 2.5.

```
PRK_exporter after KeyUpdate (Raw Value) (32 bytes)
00 14 d2 52 5e e0 d8 e2 13 ea 59 08 02 8e 9a 1c e9 a0 1c 30 54 6f 09
30 c0 44 d3 8d b5 36 2c 05
```

The OSCORE Master Secret is derived with the updated PRK_exporter:

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

where info and key_length are unchanged as in Section 2.6.

```
OSCORE Master Secret after KeyUpdate (Raw Value) (16 bytes)
ee 0f f5 42 c4 7e b0 e0 9c 69 30 76 49 bd bb e5
```

The OSCORE Master Salt is derived with the updated PRK_exporter:

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

where info and salt_length are unchanged as in Section 2.6.

```
OSCORE Master Salt after KeyUpdate (Raw Value) (8 bytes)
80 ce de 2a 1e 5a ab 48
```

2.8. Certificates

2.8.1. Responder Certificate

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

2.8.2. Initiator Certificate

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

2.8.3. Common Root Certificate

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

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

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

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

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

3.1. message_1 (First Time)

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

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

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

```
SUITES_I (CBOR Data Item) (1 byte)
06
```

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

```
Initiator's ephemeral private key
X (Raw Value) (32 bytes)
5c 41 72 ac a8 b8 2b 5a 62 e6 6f 72 22 16 f5 a1 0f 72 aa 69 f4 2c 1d
1c d3 cc d7 bf d2 9c a4 e9
```

```
Initiator's ephemeral public key, 'x'-coordinate
G_X (Raw Value) (32 bytes)
74 1a 13 d7 ba 04 8f bb 61 5e 94 38 6a a3 b6 1b ea 5b 3d 8f 65 f3 26
20 b7 49 be e8 d2 78 ef a9
```

```
Initiator's ephemeral public key, 'x'-coordinate
G_X (CBOR Data Item) (34 bytes)
58 20 74 1a 13 d7 ba 04 8f bb 61 5e 94 38 6a a3 b6 1b ea 5b 3d 8f 65
f3 26 20 b7 49 be e8 d2 78 ef a9
```

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

```
Connection identifier chosen by Initiator
C_I (Raw Value) (1 byte)
0e
```

```
Connection identifier chosen by Initiator
C_I (CBOR Data Item) (1 byte)
0e
```

No external authorization data:

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

The Initiator constructs message_1:

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

```
message_1 (CBOR Sequence) (37 bytes)
03 06 58 20 74 1a 13 d7 ba 04 8f bb 61 5e 94 38 6a a3 b6 1b ea 5b 3d
8f 65 f3 26 20 b7 49 be e8 d2 78 ef a9 0e
```

3.2. error

The Responder does not support cipher suite 6 and sends an error with ERR_CODE 2 containing SUITES_R as ERR_INFO. The Responder proposes cipher suite 2, a single cipher suite thus encoded as an int.

```
SUITES_R
02
```

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

3.3. message_1 (Second Time)

Same steps are performed as for message_1 the first time (Section 3.1) but with SUITES_I updated.

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

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

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

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

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

```
Initiator's ephemeral private key
X (Raw Value) (32 bytes)
36 8e c1 f6 9a eb 65 9b a3 7d 5a 8d 45 b2 1b dc 02 99 dc ea a8 ef 23
5f 3c a4 2c e3 53 0f 95 25
```

```
Initiator's ephemeral public key, 'x'-coordinate
G_X (Raw Value) (32 bytes)
8a f6 f4 30 eb e1 8d 34 18 40 17 a9 a1 1b f5 11 c8 df f8 f8 34 73 0b
96 c1 b7 c8 db ca 2f c3 b6
```

```
Initiator's ephemeral public key, one 'y'-coordinate
(Raw Value) (32 bytes)
51 e8 af 6c 6e db 78 16 01 ad 1d 9c 5f a8 bf 7a a1 57 16 c7 c0 6a 5d
03 85 03 c6 14 ff 80 c9 b3
```

```
Initiator's ephemeral public key, 'x'-coordinate
G_X (CBOR Data Item) (34 bytes)
58 20 8a f6 f4 30 eb e1 8d 34 18 40 17 a9 a1 1b f5 11 c8 df f8 f8 34
73 0b 96 c1 b7 c8 db ca 2f c3 b6
```

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

```
Connection identifier chosen by Initiator
C_I (Raw Value) (1 byte)
37
```

```
Connection identifier chosen by Initiator
C_I (CBOR Data Item) (1 byte)
37
```

No external authorization data:

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

The Initiator constructs message_1:

```
message_1 =
(
 3,
 [6, 2],
 h'8af6f430ebe18d34184017a9a11bf511c8dff8f834730b96
   c1b7c8dbca2fc3b6',
 -24
)
```

```
message_1 (CBOR Sequence) (39 bytes)
03 82 06 02 58 20 8a f6 f4 30 eb e1 8d 34 18 40 17 a9 a1 1b f5 11 c8
df f8 f8 34 73 0b 96 c1 b7 c8 db ca 2f c3 b6 37
```

3.4. message_2

The Responder supports the selected cipher suite 2 and not the Initiator's more preferred cipher suite(s) 6, so SUITES_I is acceptable.

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

```
Responder's ephemeral private key
Y (Raw Value) (32 bytes)
e2 f4 12 67 77 20 5e 85 3b 43 7d 6e ac a1 e1 f7 53 cd cc 3e 2c 69 fa
88 4b 0a 1a 64 09 77 e4 18
```

```
Responder's ephemeral public key, 'x'-coordinate
G_Y (Raw Value) (32 bytes)
41 97 01 d7 f0 0a 26 c2 dc 58 7a 36 dd 75 25 49 f3 37 63 c8 93 42 2c
8e a0 f9 55 a1 3a 4f f5 d5
```

```
Responder's ephemeral public key, one 'y'-coordinate
(Raw Value) (32 bytes)
5e 4f 0d d8 a3 da 0b aa 16 b9 d3 ad 56 a0 c1 86 0a 94 0a f8 59 14 91
5e 25 01 9b 40 24 17 e9 9d
```

```
Responder's ephemeral public key, 'x'-coordinate
G_Y (CBOR Data Item) (34 bytes)
58 20 41 97 01 d7 f0 0a 26 c2 dc 58 7a 36 dd 75 25 49 f3 37 63 c8 93
42 2c 8e a0 f9 55 a1 3a 4f f5 d5
```

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

```
Connection identifier chosen by Responder
C_R (raw value) (1 byte)
27
```

```
Connection identifier chosen by Responder
C_R (CBOR Data Item) (1 byte)
27
```

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

TH_2 = H(G_Y, H(message_1))

```
H(message_1) (Raw Value) (32 bytes)
ca 02 ca bd a5 a8 90 27 49 b4 2f 71 10 50 bb 4d bd 52 15 3e 87 52 75
94 b3 9f 50 cd f0 19 88 8c
```

```
H(message_1) (CBOR Data Item) (34 bytes)
58 20 ca 02 ca bd a5 a8 90 27 49 b4 2f 71 10 50 bb 4d bd 52 15 3e 87
52 75 94 b3 9f 50 cd f0 19 88 8c
```

The input to calculate TH_2 is the CBOR sequence:

G_Y, H(message_1)

```
Input to calculate TH_2 (CBOR Sequence) (68 bytes)
58 20 41 97 01 d7 f0 0a 26 c2 dc 58 7a 36 dd 75 25 49 f3 37 63 c8 93
42 2c 8e a0 f9 55 a1 3a 4f f5 d5 58 20 ca 02 ca bd a5 a8 90 27 49 b4
2f 71 10 50 bb 4d bd 52 15 3e 87 52 75 94 b3 9f 50 cd f0 19 88 8c
```

```
TH_2 (Raw Value) (32 bytes)
35 6e fd 53 77 14 25 e0 08 f3 fe 3a 86 c8 3f f4 c6 b1 6e 57 02 8f f3
9d 52 36 c1 82 b2 02 08 4b
```

```
TH_2 (CBOR Data Item) (34 bytes)
58 20 35 6e fd 53 77 14 25 e0 08 f3 fe 3a 86 c8 3f f4 c6 b1 6e 57 02
8f f3 9d 52 36 c1 82 b2 02 08 4b
```

PRK_2e is specified in Section 4.1.1.1 of [RFC9528].

First, the ECDH shared secret G_XY is computed from G_X and Y or G_Y and X:

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

Then, PRK_2e is calculated using EDHOC_Extract(), which is determined by the EDHOC hash algorithm:

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

where salt is TH_2:

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

```
PRK_2e (Raw Value) (32 bytes)
5a a0 d6 9f 3e 3d 1e 0c 47 9f 0b 8a 48 66 90 c9 80 26 30 c3 46 6b 1d
c9 23 71 c9 82 56 31 70 b5
```

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

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

```
Responder's private authentication key
SK_R (Raw Value) (32 bytes)
72 cc 47 61 db d4 c7 8f 75 89 31 aa 58 9d 34 8d 1e f8 74 a7 e3 03 ed
e2 f1 40 dc f3 e6 aa 4a ac
```

```
Responder's public authentication key, 'x'-coordinate
(Raw Value) (32 bytes)
bb c3 49 60 52 6e a4 d3 2e 94 0c ad 2a 23 41 48 dd c2 17 91 a1 2a fb
cb ac 93 62 20 46 dd 44 f0
```

```
Responder's public authentication key, 'y'-coordinate
(Raw Value) (32 bytes)
45 19 e2 57 23 6b 2a 0c e2 02 3f 09 31 f1 f3 86 ca 7a fd a6 4f cd e0
10 8c 22 4c 51 ea bf 60 72
```

Since the Responder authenticates with static DH (METHOD = 3), PRK_3e2m is derived from SALT_3e2m and G_RX.

The input needed to calculate SALT_3e2m is defined in Section 4.1.2 of [RFC9528], using EDHOC_Expand() with the EDHOC hash algorithm:

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

where hash_length is the length in bytes of the output of the EDHOC hash algorithm, and info for SALT_3e2m is:

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

```
info for SALT_3e2m (CBOR Sequence) (37 bytes)
01 58 20 35 6e fd 53 77 14 25 e0 08 f3 fe 3a 86 c8 3f f4 c6 b1 6e 57
02 8f f3 9d 52 36 c1 82 b2 02 08 4b 18 20
```

```
SALT_3e2m (Raw Value) (32 bytes)
af 4e 10 3a 47 cb 3c f3 25 70 d5 c2 5a d2 77 32 bd 8d 81 78 e9 a6 9d
06 1c 31 a2 7f 8e 3c a9 26
```

PRK_3e2m is specified in Section 4.1.1.2 of [RFC9528].

PRK_3e2m is derived from G_RX using EDHOC_Extract() with the EDHOC hash algorithm:

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

where G_RX is the ECDH shared secret calculated from G_X and R, or G_R and X.

```
G_RX (Raw Value) (ECDH shared secret) (32 bytes)
f2 b6 ee a0 22 20 b9 5e ee 5a 0b c7 01 f0 74 e0 0a 84 3e a0 24 22 f6
08 25 fb 26 9b 3e 16 14 23
```

```
PRK_3e2m (Raw Value) (32 bytes)
0c a3 d3 39 82 96 b3 c0 39 00 98 76 20 c1 1f 6f ce 70 78 1c 1d 12 19
72 0f 9e c0 8c 12 2d 84 34
```

The Responder constructs the remaining input needed to calculate MAC_2:

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

context_2 = << C_R, 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)
a2 02 6b 65 78 61 6d 70 6c 65 2e 65 64 75 08 a1 01 a5 01 02 02 41 32
20 01 21 58 20 bb c3 49 60 52 6e a4 d3 2e 94 0c ad 2a 23 41 48 dd c2
17 91 a1 2a fb cb ac 93 62 20 46 dd 44 f0 22 58 20 45 19 e2 57 23 6b
2a 0c e2 02 3f 09 31 f1 f3 86 ca 7a fd a6 4f cd e0 10 8c 22 4c 51 ea
bf 60 72
```

No external authorization data:

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

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

```
context_2 (CBOR Sequence) (134 bytes)
27 a1 04 41 32 58 20 35 6e fd 53 77 14 25 e0 08 f3 fe 3a 86 c8 3f f4
c6 b1 6e 57 02 8f f3 9d 52 36 c1 82 b2 02 08 4b a2 02 6b 65 78 61 6d
70 6c 65 2e 65 64 75 08 a1 01 a5 01 02 02 41 32 20 01 21 58 20 bb c3
49 60 52 6e a4 d3 2e 94 0c ad 2a 23 41 48 dd c2 17 91 a1 2a fb cb ac
93 62 20 46 dd 44 f0 22 58 20 45 19 e2 57 23 6b 2a 0c e2 02 3f 09 31
f1 f3 86 ca 7a fd a6 4f cd e0 10 8c 22 4c 51 ea bf 60 72
```

```
context_2 (CBOR byte string) (136 bytes)
58 86 27 a1 04 41 32 58 20 35 6e fd 53 77 14 25 e0 08 f3 fe 3a 86 c8
3f f4 c6 b1 6e 57 02 8f f3 9d 52 36 c1 82 b2 02 08 4b a2 02 6b 65 78
61 6d 70 6c 65 2e 65 64 75 08 a1 01 a5 01 02 02 41 32 20 01 21 58 20
bb c3 49 60 52 6e a4 d3 2e 94 0c ad 2a 23 41 48 dd c2 17 91 a1 2a fb
cb ac 93 62 20 46 dd 44 f0 22 58 20 45 19 e2 57 23 6b 2a 0c e2 02 3f
09 31 f1 f3 86 ca 7a fd a6 4f cd e0 10 8c 22 4c 51 ea bf 60 72
```

MAC_2 is computed through EDHOC_Expand() using the EDHOC hash algorithm (see Section 4.1.2 of [RFC9528]):

```
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'27a10441325820356efd53771425e008f3fe3a86c83ff4c6
   b16e57028ff39d5236c182b202084ba2026b6578616d706c
   652e65647508a101a501020241322001215820bbc3496052
   6ea4d32e940cad2a234148ddc21791a12afbcbac93622046
   dd44f02258204519e257236b2a0ce2023f0931f1f386ca7a
   fda64fcde0108c224c51eabf6072',
 8
)
```

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

```
info for MAC_2 (CBOR Sequence) (138 bytes)
02 58 86 27 a1 04 41 32 58 20 35 6e fd 53 77 14 25 e0 08 f3 fe 3a 86
c8 3f f4 c6 b1 6e 57 02 8f f3 9d 52 36 c1 82 b2 02 08 4b a2 02 6b 65
78 61 6d 70 6c 65 2e 65 64 75 08 a1 01 a5 01 02 02 41 32 20 01 21 58
20 bb c3 49 60 52 6e a4 d3 2e 94 0c ad 2a 23 41 48 dd c2 17 91 a1 2a
fb cb ac 93 62 20 46 dd 44 f0 22 58 20 45 19 e2 57 23 6b 2a 0c e2 02
3f 09 31 f1 f3 86 ca 7a fd a6 4f cd e0 10 8c 22 4c 51 ea bf 60 72 08
```

```
MAC_2 (Raw Value) (8 bytes)
09 43 30 5c 89 9f 5c 54
```

```
MAC_2 (CBOR Data Item) (9 bytes)
48 09 43 30 5c 89 9f 5c 54
```

Since METHOD = 3, Signature_or_MAC_2 is MAC_2:

```
Signature_or_MAC_2 (Raw Value) (8 bytes)
09 43 30 5c 89 9f 5c 54
```

```
Signature_or_MAC_2 (CBOR Data Item) (9 bytes)
48 09 43 30 5c 89 9f 5c 54
```

The Responder constructs PLAINTEXT_2:

```
PLAINTEXT_2 =
(
 C_R,
 ID_CRED_R / bstr / -24..23,
 Signature_or_MAC_2,
 ? EAD_2
)
```

Since ID_CRED_R contains a single 'kid' parameter, only the byte string value is included in the plaintext, represented as described in Section 3.3.2 of [RFC9528]. The CBOR map { 4 : h'32' } is thus replaced, not by the CBOR byte string 0x4132, but by the CBOR int 0x32, since that is a one-byte encoding of a CBOR integer (-19).

```
PLAINTEXT_2 (CBOR Sequence) (11 bytes)
27 32 48 09 43 30 5c 89 9f 5c 54
```

The input needed to calculate KEYSTREAM_2 is defined in Section 4.1.2 of [RFC9528], using EDHOC_Expand() with the EDHOC hash algorithm:

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

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

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

where the last value is the length in bytes of PLAINTEXT_2.

```
info for KEYSTREAM_2 (CBOR Sequence) (36 bytes)
00 58 20 35 6e fd 53 77 14 25 e0 08 f3 fe 3a 86 c8 3f f4 c6 b1 6e 57
02 8f f3 9d 52 36 c1 82 b2 02 08 4b 0b
```

```
KEYSTREAM_2 (Raw Value) (11 bytes)
bf 50 e9 e7 ba d0 bb 68 17 33 99
```

The Responder calculates CIPHERTEXT_2 as XOR between PLAINTEXT_2 and KEYSTREAM_2:

```
CIPHERTEXT_2 (Raw Value) (11 bytes)
98 62 a1 ee f9 e0 e7 e1 88 6f cd
```

The Responder constructs message_2:

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

where G_Y_CIPHERTEXT_2 is the bstr encoding of the concatenation of the raw values of G_Y and CIPHERTEXT_2.

```
message_2 (CBOR Sequence) (45 bytes)
58 2b 41 97 01 d7 f0 0a 26 c2 dc 58 7a 36 dd 75 25 49 f3 37 63 c8 93
42 2c 8e a0 f9 55 a1 3a 4f f5 d5 98 62 a1 ee f9 e0 e7 e1 88 6f cd
```

3.5. message_3

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

TH_3 = H(TH_2, PLAINTEXT_2, CRED_R)

```
Input to calculate TH_3 (CBOR Sequence) (140 bytes)
58 20 35 6e fd 53 77 14 25 e0 08 f3 fe 3a 86 c8 3f f4 c6 b1 6e 57 02
8f f3 9d 52 36 c1 82 b2 02 08 4b 27 32 48 09 43 30 5c 89 9f 5c 54 a2
02 6b 65 78 61 6d 70 6c 65 2e 65 64 75 08 a1 01 a5 01 02 02 41 32 20
01 21 58 20 bb c3 49 60 52 6e a4 d3 2e 94 0c ad 2a 23 41 48 dd c2 17
91 a1 2a fb cb ac 93 62 20 46 dd 44 f0 22 58 20 45 19 e2 57 23 6b 2a
0c e2 02 3f 09 31 f1 f3 86 ca 7a fd a6 4f cd e0 10 8c 22 4c 51 ea bf
60 72
```

```
TH_3 (Raw Value) (32 bytes)
ad af 67 a7 8a 4b cc 91 e0 18 f8 88 27 62 a7 22 00 0b 25 07 03 9d f0
bc 1b bf 0c 16 1b b3 15 5c
```

```
TH_3 (CBOR Data Item) (34 bytes)
58 20 ad af 67 a7 8a 4b cc 91 e0 18 f8 88 27 62 a7 22 00 0b 25 07 03
9d f0 bc 1b bf 0c 16 1b b3 15 5c
```

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

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

```
Initiator's private authentication key
SK_I (Raw Value) (32 bytes)
fb 13 ad eb 65 18 ce e5 f8 84 17 66 08 41 14 2e 83 0a 81 fe 33 43 80
a9 53 40 6a 13 05 e8 70 6b
```

```
Initiator's public authentication key, 'x'-coordinate
(Raw Value) (32 bytes)
ac 75 e9 ec e3 e5 0b fc 8e d6 03 99 88 95 22 40 5c 47 bf 16 df 96 66
0a 41 29 8c b4 30 7f 7e b6
```

```
Initiator's public authentication key, 'y'-coordinate
(Raw Value) (32 bytes)
6e 5d e6 11 38 8a 4b 8a 82 11 33 4a c7 d3 7e cb 52 a3 87 d2 57 e6 db
3c 2a 93 df 21 ff 3a ff c8
```

Since I authenticates with static DH (METHOD = 3), PRK_4e3m is derived from SALT_4e3m and G_IY.

The input needed to calculate SALT_4e3m is defined in Section 4.1.2 of [RFC9528], using EDHOC_Expand() with the EDHOC hash algorithm:

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

where hash_length is the length in bytes of the output of the EDHOC hash algorithm, and info for SALT_4e3m is:

```
info =
(
 5,
 h'adaf67a78a4bcc91e018f8882762a722000b2507039df0bc
   1bbf0c161bb3155c',
 32
)
```

```
info for SALT_4e3m (CBOR Sequence) (37 bytes)
05 58 20 ad af 67 a7 8a 4b cc 91 e0 18 f8 88 27 62 a7 22 00 0b 25 07
03 9d f0 bc 1b bf 0c 16 1b b3 15 5c 18 20
```

```
SALT_4e3m (Raw Value) (32 bytes)
cf dd f9 51 5a 7e 46 e7 b4 db ff 31 cb d5 6c d0 4b a3 32 25 0d e9 ea
5d e1 ca f9 f6 d1 39 14 a7
```

PRK_4e3m is specified in Section 4.1.1.3 of [RFC9528].

Since I authenticates with static DH (METHOD = 3), PRK_4e3m is derived from G_IY using EDHOC_Extract() with the EDHOC hash algorithm:

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

where G_IY is the ECDH shared secret calculated from G_I and Y, or G_Y and I.

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

```
PRK_4e3m (Raw Value) (32 bytes)
81 cc 8a 29 8e 35 70 44 e3 c4 66 bb 5c 0a 1e 50 7e 01 d4 92 38 ae ba
13 8d f9 46 35 40 7c 0f f7
```

The Initiator constructs the remaining input needed to calculate MAC_3:

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

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

CRED_I is identified by a 'kid' with byte string value 0x2b:

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

```
ID_CRED_I (CBOR Data Item) (4 bytes)
a1 04 41 2b
```

CRED_I is an RPK encoded as a CCS:

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

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

No external authorization data:

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

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

```
context_3 (CBOR Sequence) (145 bytes)
a1 04 41 2b 58 20 ad af 67 a7 8a 4b cc 91 e0 18 f8 88 27 62 a7 22 00
0b 25 07 03 9d f0 bc 1b bf 0c 16 1b b3 15 5c a2 02 77 34 32 2d 35 30
2d 33 31 2d 46 46 2d 45 46 2d 33 37 2d 33 32 2d 33 39 08 a1 01 a5 01
02 02 41 2b 20 01 21 58 20 ac 75 e9 ec e3 e5 0b fc 8e d6 03 99 88 95
22 40 5c 47 bf 16 df 96 66 0a 41 29 8c b4 30 7f 7e b6 22 58 20 6e 5d
e6 11 38 8a 4b 8a 82 11 33 4a c7 d3 7e cb 52 a3 87 d2 57 e6 db 3c 2a
93 df 21 ff 3a ff c8
```

```
context_3 (CBOR byte string) (147 bytes)
58 91 a1 04 41 2b 58 20 ad af 67 a7 8a 4b cc 91 e0 18 f8 88 27 62 a7
22 00 0b 25 07 03 9d f0 bc 1b bf 0c 16 1b b3 15 5c a2 02 77 34 32 2d
35 30 2d 33 31 2d 46 46 2d 45 46 2d 33 37 2d 33 32 2d 33 39 08 a1 01
a5 01 02 02 41 2b 20 01 21 58 20 ac 75 e9 ec e3 e5 0b fc 8e d6 03 99
88 95 22 40 5c 47 bf 16 df 96 66 0a 41 29 8c b4 30 7f 7e b6 22 58 20
6e 5d e6 11 38 8a 4b 8a 82 11 33 4a c7 d3 7e cb 52 a3 87 d2 57 e6 db
3c 2a 93 df 21 ff 3a ff c8
```

MAC_3 is computed through EDHOC_Expand() using the EDHOC hash algorithm (see Section 4.1.2 of [RFC9528]):

```
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'a104412b5820adaf67a78a4bcc91e018f8882762a722000b
   2507039df0bc1bbf0c161bb3155ca2027734322d35302d33
   312d46462d45462d33372d33322d333908a101a501020241
   2b2001215820ac75e9ece3e50bfc8ed60399889522405c47
   bf16df96660a41298cb4307f7eb62258206e5de611388a4b
   8a8211334ac7d37ecb52a387d257e6db3c2a93df21ff3aff
   c8',
 8
)
```

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

```
info for MAC_3 (CBOR Sequence) (149 bytes)
06 58 91 a1 04 41 2b 58 20 ad af 67 a7 8a 4b cc 91 e0 18 f8 88 27 62
a7 22 00 0b 25 07 03 9d f0 bc 1b bf 0c 16 1b b3 15 5c a2 02 77 34 32
2d 35 30 2d 33 31 2d 46 46 2d 45 46 2d 33 37 2d 33 32 2d 33 39 08 a1
01 a5 01 02 02 41 2b 20 01 21 58 20 ac 75 e9 ec e3 e5 0b fc 8e d6 03
99 88 95 22 40 5c 47 bf 16 df 96 66 0a 41 29 8c b4 30 7f 7e b6 22 58
20 6e 5d e6 11 38 8a 4b 8a 82 11 33 4a c7 d3 7e cb 52 a3 87 d2 57 e6
db 3c 2a 93 df 21 ff 3a ff c8 08
```

```
MAC_3 (Raw Value) (8 bytes)
62 3c 91 df 41 e3 4c 2f
```

```
MAC_3 (CBOR Data Item) (9 bytes)
48 62 3c 91 df 41 e3 4c 2f
```

Since METHOD = 3, Signature_or_MAC_3 is MAC_3:

```
Signature_or_MAC_3 (Raw Value) (8 bytes)
62 3c 91 df 41 e3 4c 2f
```

```
Signature_or_MAC_3 (CBOR Data Item) (9 bytes)
48 62 3c 91 df 41 e3 4c 2f
```

The Initiator constructs PLAINTEXT_3:

```
PLAINTEXT_3 =
(
 ID_CRED_I / bstr / -24..23,
 Signature_or_MAC_3,
 ? EAD_3
)
```

Since ID_CRED_I contains a single 'kid' parameter, only the byte string value is included in the plaintext, represented as described in Section 3.3.2 of [RFC9528]. 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 62 3c 91 df 41 e3 4c 2f
```

The Initiator constructs the associated data for message_3:

```
A_3 =
[
 "Encrypt0",
 h'',
 h'adaf67a78a4bcc91e018f8882762a722000b2507039df0bc
   1bbf0c161bb3155c'
]
```

```
A_3 (CBOR Data Item) (45 bytes)
83 68 45 6e 63 72 79 70 74 30 40 58 20 ad af 67 a7 8a 4b cc 91 e0 18
f8 88 27 62 a7 22 00 0b 25 07 03 9d f0 bc 1b bf 0c 16 1b b3 15 5c
```

The Initiator constructs the input needed to derive the key K_3 (see Section 4.1.2 of [RFC9528]) using the EDHOC hash algorithm:

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

where key_length is the key length in bytes for the EDHOC AEAD algorithm, and info for K_3 is:

```
info =
(
 3,
 h'adaf67a78a4bcc91e018f8882762a722000b2507039df0bc
   1bbf0c161bb3155c',
 16
)
```

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

```
info for K_3 (CBOR Sequence) (36 bytes)
03 58 20 ad af 67 a7 8a 4b cc 91 e0 18 f8 88 27 62 a7 22 00 0b 25 07
03 9d f0 bc 1b bf 0c 16 1b b3 15 5c 10
```

```
K_3 (Raw Value) (16 bytes)
8e 7a 30 04 20 00 f7 90 0e 81 74 13 1f 75 f3 ed
```

The Initiator constructs the input needed to derive the nonce IV_3 (see Section 4.1.2 of [RFC9528]) using the EDHOC hash algorithm:

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

where iv_length is the nonce length in bytes for the EDHOC AEAD algorithm, and info for IV_3 is:

```
info =
(
 4,
 h'adaf67a78a4bcc91e018f8882762a722000b2507039df0bc
   1bbf0c161bb3155c',
 13
)
```

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

```
info for IV_3 (CBOR Sequence) (36 bytes)
04 58 20 ad af 67 a7 8a 4b cc 91 e0 18 f8 88 27 62 a7 22 00 0b 25 07
03 9d f0 bc 1b bf 0c 16 1b b3 15 5c 0d
```

```
IV_3 (Raw Value) (13 bytes)
6d 83 00 c1 e2 3b 56 15 3a e7 0e e4 57
```

The Initiator calculates CIPHERTEXT_3 as 'ciphertext' of COSE_Encrypt0 applied using the EDHOC AEAD algorithm with plaintext PLAINTEXT_3, additional data A_3, key K_3, and nonce IV_3.

```
CIPHERTEXT_3 (Raw Value) (18 bytes)
e5 62 09 7b c4 17 dd 59 19 48 5a c7 89 1f fd 90 a9 fc
```

message_3 is the CBOR bstr encoding of CIPHERTEXT_3:

```
message_3 (CBOR Sequence) (19 bytes)
52 e5 62 09 7b c4 17 dd 59 19 48 5a c7 89 1f fd 90 a9 fc
```

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

TH_4 = H(TH_3, PLAINTEXT_3, CRED_I)

```
Input to calculate TH_4 (CBOR Sequence) (151 bytes)
58 20 ad af 67 a7 8a 4b cc 91 e0 18 f8 88 27 62 a7 22 00 0b 25 07 03
9d f0 bc 1b bf 0c 16 1b b3 15 5c 2b 48 62 3c 91 df 41 e3 4c 2f a2 02
77 34 32 2d 35 30 2d 33 31 2d 46 46 2d 45 46 2d 33 37 2d 33 32 2d 33
39 08 a1 01 a5 01 02 02 41 2b 20 01 21 58 20 ac 75 e9 ec e3 e5 0b fc
8e d6 03 99 88 95 22 40 5c 47 bf 16 df 96 66 0a 41 29 8c b4 30 7f 7e
b6 22 58 20 6e 5d e6 11 38 8a 4b 8a 82 11 33 4a c7 d3 7e cb 52 a3 87
d2 57 e6 db 3c 2a 93 df 21 ff 3a ff c8
```

```
TH_4 (Raw Value) (32 bytes)
c9 02 b1 e3 a4 32 6c 93 c5 55 1f 5f 3a a6 c5 ec c0 24 68 06 76 56 12
e5 2b 5d 99 e6 05 9d 6b 6e
```

```
TH_4 (CBOR Data Item) (34 bytes)
58 20 c9 02 b1 e3 a4 32 6c 93 c5 55 1f 5f 3a a6 c5 ec c0 24 68 06 76
56 12 e5 2b 5d 99 e6 05 9d 6b 6e
```

3.6. message_4

No external authorization data:

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

The Responder constructs PLAINTEXT_4:

```
PLAINTEXT_4 =
(
 ? EAD_4
)
```

```
PLAINTEXT_4 (CBOR Sequence) (0 bytes)
```

The Responder constructs the associated data for message_4:

```
A_4 =
[
 "Encrypt0",
 h'',
 h'c902b1e3a4326c93c5551f5f3aa6c5ecc0246806765612e5
   2b5d99e6059d6b6e'
]
```

```
A_4 (CBOR Data Item) (45 bytes)
83 68 45 6e 63 72 79 70 74 30 40 58 20 c9 02 b1 e3 a4 32 6c 93 c5 55
1f 5f 3a a6 c5 ec c0 24 68 06 76 56 12 e5 2b 5d 99 e6 05 9d 6b 6e
```

The Responder constructs the input needed to derive the EDHOC message_4 key (see Section 4.1.2 of [RFC9528]) using the EDHOC hash algorithm:

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

where key_length is the key length in bytes for the EDHOC AEAD algorithm, and info for K_4 is:

```
info =
(
 8,
 h'c902b1e3a4326c93c5551f5f3aa6c5ecc0246806765612e5
   2b5d99e6059d6b6e',
 16
)
```

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

```
info for K_4 (CBOR Sequence) (36 bytes)
08 58 20 c9 02 b1 e3 a4 32 6c 93 c5 55 1f 5f 3a a6 c5 ec c0 24 68 06
76 56 12 e5 2b 5d 99 e6 05 9d 6b 6e 10
```

```
K_4 (Raw Value) (16 bytes)
d3 c7 78 72 b6 ee b5 08 91 1b db d3 08 b2 e6 a0
```

The Responder constructs the input needed to derive the EDHOC message_4 nonce (see Section 4.1.2 of [RFC9528]) using the EDHOC hash algorithm:

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

where iv_length is the nonce length in bytes for the EDHOC AEAD algorithm, and info for IV_4 is:

```
info =
(
 9,
 h'c902b1e3a4326c93c5551f5f3aa6c5ecc0246806765612e5
   2b5d99e6059d6b6e',
 13
)
```

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

```
info for IV_4 (CBOR Sequence) (36 bytes)
09 58 20 c9 02 b1 e3 a4 32 6c 93 c5 55 1f 5f 3a a6 c5 ec c0 24 68 06
76 56 12 e5 2b 5d 99 e6 05 9d 6b 6e 0d
```

```
IV_4 (Raw Value) (13 bytes)
04 ff 0f 44 45 6e 96 e2 17 85 3c 36 01
```

The Responder calculates CIPHERTEXT_4 as 'ciphertext' of COSE_Encrypt0 applied using the EDHOC AEAD algorithm with plaintext PLAINTEXT_4, additional data A_4, key K_4, and nonce IV_4.

```
CIPHERTEXT_4 (8 bytes)
28 c9 66 b7 ca 30 4f 83
```

message_4 is the CBOR bstr encoding of CIPHERTEXT_4:

```
message_4 (CBOR Sequence) (9 bytes)
48 28 c9 66 b7 ca 30 4f 83
```

3.7. PRK_out and PRK_exporter

PRK_out is specified in Section 4.1.3 of [RFC9528].

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

where hash_length is the length in bytes of the output of the EDHOC hash algorithm, and info for PRK_out is:

```
info =
(
 7,
 h'c902b1e3a4326c93c5551f5f3aa6c5ecc0246806765612e5
   2b5d99e6059d6b6e',
 32
)
```

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

```
info for PRK_out (CBOR Sequence) (37 bytes)
07 58 20 c9 02 b1 e3 a4 32 6c 93 c5 55 1f 5f 3a a6 c5 ec c0 24 68 06
76 56 12 e5 2b 5d 99 e6 05 9d 6b 6e 18 20
```

```
PRK_out (Raw Value) (32 bytes)
2c 71 af c1 a9 33 8a 94 0b b3 52 9c a7 34 b8 86 f3 0d 1a ba 0b 4d c5
1b ee ae ab df ea 9e cb f8
```

The OSCORE Master Secret and OSCORE Master Salt are derived with the EDHOC_Exporter as specified in Section 4.2.1 of [RFC9528].

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

where PRK_exporter is derived from PRK_out:

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

where hash_length is the length in bytes of the output of the EDHOC hash algorithm, and info for the PRK_exporter is:

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

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

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

```
PRK_exporter (Raw Value) (32 bytes)
e1 4d 06 69 9c ee 24 8c 5a 04 bf 92 27 bb cd 4c e3 94 de 7d cb 56 db
43 55 54 74 17 1e 64 46 db
```

3.8. OSCORE Parameters

The derivation of OSCORE parameters is specified in Appendix A.1 of [RFC9528].

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 [RFC9528].

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

```
Client's OSCORE Sender ID (Raw Value) (1 byte)
27
```

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

```
Server's OSCORE Sender ID (Raw Value) (1 byte)
37
```

The OSCORE Master Secret is computed through EDHOC_Expand() using the application hash algorithm (see Appendix A.1 of [RFC9528]):

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

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

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

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

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

```
OSCORE Master Secret (Raw Value) (16 bytes)
f9 86 8f 6a 3a ca 78 a0 5d 14 85 b3 50 30 b1 62
```

The OSCORE Master Salt is computed through EDHOC_Expand() using the application hash algorithm (see Section 4.2 of [RFC9528]):

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

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

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

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

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

```
OSCORE Master Salt (Raw Value) (8 bytes)
ad a2 4c 7d bf c8 5e eb
```

3.9. Key Update

The key update is defined in Appendix H of [RFC9528].

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

where hash_length is the length in bytes of the output of the EDHOC hash function, and the 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 the key update is:

```
info =3.7
(
 11,
 h'a01158fdb820890cd6be169602b8bcea',
 32
)
```

```
PRK_out after KeyUpdate (Raw Value) (32 bytes)
f9 79 53 77 43 fe 0b d6 b9 b1 41 dd bd 79 65 6c 52 e6 dc 7c 50 ad 80
77 54 d7 4d 07 e8 7d 0d 16
```

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

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

where info and hash_length are unchanged as in Section 3.7.

```
PRK_exporter after KeyUpdate (Raw Value) (32 bytes)
00 fc f7 db 9b 2e ad 73 82 4e 7e 83 03 63 c8 05 c2 96 f9 02 83 0f ac
23 d8 6c 35 9c 75 2f 0f 17
```

The OSCORE Master Secret is derived with the updated PRK_exporter:

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

where info and key_length are unchanged as in Section 2.6.

```
OSCORE Master Secret after KeyUpdate (Raw Value) (16 bytes)
49 f7 2f ac 02 b4 65 8b da 21 e2 da c6 6f c3 74
```

The OSCORE Master Salt is derived with the updated PRK_exporter:

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

where info and salt_length are unchanged as in Section 2.6.

```
OSCORE Master Salt after KeyUpdate (Raw Value) (8 bytes)
dd 8b 24 f2 aa 9b 01 1a
```

4. Invalid Traces

This section contains examples of invalid messages, which a compliant implementation will not compose and must or may reject according to [RFC9528], [RFC8949], [RFC9053], and [SP-800-56A]. This is just a small set of examples of different reasons a message might be invalid. The same types of invalidities applies to other fields and messages as well. Implementations should make sure to check for similar types of invalidities in all EDHOC fields and messages.

4.1. Encoding Errors

4.1.1. Surplus Array Encoding of a Message

Invalid encoding of message_1 as array. The correct encoding is a CBOR sequence according to Section 5.2.1 of [RFC9528].

```
Invalid message_1 (38 bytes)
84 03 02 58 20 74 1a 13 d7 ba 04 8f bb 61 5e 94 38 6a a3 b6 1b ea 5b
3d 8f 65 f3 26 20 b7 49 be e8 d2 78 ef a9 0e
```

4.1.2. Surplus bstr Encoding of the Connection Identifier

Invalid encoding 41 0e of C_I = 0x0e. The correct encoding is 0e according to Section 3.3.2 of [RFC9528].

```
Invalid message_1 (38 bytes)
03 02 58 20 74 1a 13 d7 ba 04 8f bb 61 5e 94 38 6a a3 b6 1b ea 5b 3d
8f 65 f3 26 20 b7 49 be e8 d2 78 ef a9 41 0e
```

4.1.3. Surplus Array Encoding of the Ciphersuite

Invalid array encoding 81 02 of SUITES_I = 2. The correct encoding is 02 according to Section 5.2.2 of [RFC9528].

```
Invalid message_1 (38 bytes)
03 81 02 58 20 74 1a 13 d7 ba 04 8f bb 61 5e 94 38 6a a3 b6 1b ea 5b
3d 8f 65 f3 26 20 b7 49 be e8 d2 78 ef a9 0e
```

4.1.4. Text String Encoding of the Ephemeral Key

Invalid type of the third element (G_X). The correct encoding is a byte string according to Section 5.2.1 of [RFC9528].

```
Invalid message_1 (37 bytes)
03 02 78 20 20 61 69 72 20 73 70 65 65 64 20 6F 66 20 61 20 75 6E 6C
61 64 65 6E 20 73 77 61 6C 6C 6F 77 20 0e
```

4.1.5. Wrong Number of CBOR Sequence Elements

Invalid number of elements in the CBOR sequence. Correct number of elements is 1 according to Section 5.3.1 of [RFC9528].

```
Invalid message_2 (46 bytes)
58 20 41 97 01 d7 f0 0a 26 c2 dc 58 7a 36 dd 75 25 49 f3 37 63 c8 93
42 2c 8e a0 f9 55 a1 3a 4f f5 d5 4B 98 62 a1 1d e4 2a 95 d7 85 38 6a
```

4.1.6. Surplus Map Encoding of the ID_CRED Field

Invalid encoding a1 04 42 32 10 of ID_CRED_R in PLAINTEXT_2. The correct encoding is 42 32 10 according to Section 3.5.3.2 of [RFC9528].

```
Invalid PLAINTEXT_2 (15 bytes)
27 a1 04 42 32 10 48 fa 5e fa 2e bf 92 0b f3
```

4.1.7. Surplus bstr Encoding of the ID_CRED Field

Invalid encoding 41 32 of ID_CRED_R in PLAINTEXT_2. The correct encoding is 32 according to Section 3.5.3.2 of [RFC9528].

```
Invalid PLAINTEXT_2 (12 bytes)
27 41 32 48 fa 5e fa 2e bf 92 0b f3
```

4.2. Crypto-Related Errors

4.2.1. Error in the Length of the Ephemeral Key

Invalid length of the third element (G_X). The selected cipher suite is cipher suite 24 with curve P-384 according to Sections 5.2.2 and 10.2 of [RFC9528]. The correct length of the x-coordinate is 48 bytes according to Section 3.7 of [RFC9528] and Section 7.1.1 of [RFC9053].

```
Invalid message_1 (40 bytes)
03 82 02 18 18 58 20 74 1a 13 d7 ba 04 8f bb 61 5e 94 38 6a a3 b6 1b
ea 5b 3d 8f 65 f3 26 20 b7 49 be e8 d2 78 ef a9 0e
```

4.2.2. Error in Elliptic Curve Representation

Invalid x-coordinate in G_X as $x \geq p$. Requirement that $x < p$ according to Section 9.2 of [RFC9528] and Section 5.6.2.3 of [SP-800-56A].

```
Invalid message_1 (37 bytes)
03 02 58 20 ff ff ff ff 00 00 00 01 00 00 00 00 00 00 00 00 00 00
00 ff ff ff ff ff ff ff ff ff ff ff ff 0e
```

4.2.3. Error in the Elliptic Curve Point

Invalid x-coordinate in (G_X) not corresponding to a point on the P-256 curve. Requirement that $y^2 \equiv x^3 + a \cdot x + b \pmod{p}$ according to Section 9.2 of [RFC9528] and Section 5.6.2.3 of [SP-800-56A].

```
Invalid message_1 (37 bytes)
03 02 58 20 a0 4e 73 60 1d f5 44 a7 0b a7 ea 1e 57 03 0f 7d 4b 4e b7
f6 73 92 4e 58 d5 4c a7 7a 5e 7d 4d 4a 0e
```

4.2.4. Curve Point of the Low Order

Curve25519 point of low order which fails the check for all-zero output according to Section 9.2 of [RFC9528].

```
Invalid message_1 (37 bytes)
03 00 58 20 ed ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff 7f 0e
```

4.2.5. Error in the Length of the MAC

Invalid length of third element (Signature_or_MAC_2). The length of Signature_or_MAC_2 is given by the cipher suite, and the MAC length is at least 8 bytes according to Section 9.3 of [RFC9528].

```
Invalid PLAINTEXT_2 (7 bytes)
27 32 44 fa 5e fa 2e
```

4.2.6. Error in the Elliptic Curve Encoding

Invalid encoding of third element (G_X). The correct encoding is with leading zeros according to Section 3.7 of [RFC9528] and Section 7.1.1 of [RFC9053].

```
Invalid message_1 (36 bytes)
03 02 58 1f d9 69 77 25 d2 3a 68 8b 12 d1 c7 e0 10 8a 08 c9 f7 1a 85
a0 9c 20 81 49 76 ab 21 12 22 48 fc 0e
```

4.3. Non-deterministic CBOR

4.3.1. Unnecessary Long Encoding

Invalid 16-bit encoding 19 00 03 of METHOD = 3. Correct is the deterministic encoding 03 according to Section 3.1 of [RFC9528] and Section 4.2.1 of [RFC8949], which states that the arguments for integers, lengths in major types 2 through 5, and tags are required to be as short as possible.

```
Invalid message_1 (39 bytes)
19 00 03 02 58 20 74 1a 13 d7 ba 04 8f bb 61 5e 94 38 6a a3 b6 1b ea
5b 3d 8f 65 f3 26 20 b7 49 be e8 d2 78 ef a9 0e
```

4.3.2. Indefinite-Length Array Encoding

Invalid indefinite-length array encoding 9F 06 02 FF of SUITES_I = [6, 2]. The correct encoding is 82 06 02 according to Section 5.2.2 of [RFC9528].

```
Invalid message_1 (40 bytes)
03 9F 06 02 FF 58 20 74 1a 13 d7 ba 04 8f bb 61 5e 94 38 6a a3 b6 1b
ea 5b 3d 8f 65 f3 26 20 b7 49 be e8 d2 78 ef a9 0e
```

5. Security Considerations

This document contains examples of EDHOC [RFC9528]. The security considerations described in [RFC9528] apply. The keys printed in these examples cannot be considered secret and **MUST NOT** be used.

6. IANA Considerations

This document has no IANA actions.

7. References

7.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC9528] Selander, G., Preuß Mattsson, J., and F. Palombini, "Ephemeral Diffie-Hellman Over COSE (EDHOC)", RFC 9528, DOI 10.17487/RFC9528, March 2024, <https://www.rfc-editor.org/rfc/rfc9528>.

7.2. Informative References

[CborMe] Bormann, C., "CBOR playground", <https://cbor.me/>.

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <https://www.rfc-editor.org/info/rfc7252>.

[RFC7748] Langley, A., Hamburg, M., and S. Turner, "Elliptic Curves for Security", RFC 7748, DOI 10.17487/RFC7748, January 2016, <https://www.rfc-editor.org/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>.

[RFC9053] Schaad, J., "CBOR Object Signing and Encryption (COSE): Initial Algorithms", RFC 9053, DOI 10.17487/RFC9053, August 2022, <https://www.rfc-editor.org/info/rfc9053>.

[SP-800-186] Chen, L., Moody, D., Randall, K., Regenscheid, A., and A. Robinson, "Recommendations for Discrete Logarithm-based Cryptography: Elliptic Curve Domain Parameters", NIST Special Publication 800-186, DOI 10.6028/NIST.SP.800-186, February 2023, <https://doi.org/10.6028/NIST.SP.800-186>.

[SP-800-56A] Barker, E., Chen, L., Roginsky, A., Vassilev, A., and R. Davis, "Recommendation for Pair-Wise Key-Establishment Schemes Using Discrete Logarithm Cryptography", NIST Special Publication 800-56A Revision 3, DOI 10.6028/NIST.SP.800-56Ar3, April 2018, <https://doi.org/10.6028/NIST.SP.800-56Ar3>.

Acknowledgments

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

Authors' Addresses

Göran Selander
Ericsson
Sweden
Email: goran.selander@ericsson.com

John Preuß Mattsson
Ericsson
Sweden
Email: john.mattsson@ericsson.com

Marek Serafin
ASSA ABLOY
Poland
Email: marek.serafin@assaabloy.com

Marco Tiloca
RISE
Sweden
Email: marco.tiloca@ri.se

Mališa Vučinić
Inria
France
Email: malisa.vucinic@inria.fr