Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: March 14, 2020

G. Selander
J. Mattsson
F. Palombini
Ericsson AB

September 11, 2019

Ephemeral Diffie-Hellman Over COSE (EDHOC)
draft-selander-ace-cose-ecdhe-14

Abstract

This document specifies Ephemeral Diffie-Hellman Over COSE (EDHOC), a very compact, and lightweight authenticated Diffie-Hellman key exchange with ephemeral keys. EDHOC provides mutual authentication, perfect forward secrecy, and identity protection. EDHOC is intended for usage in constrained scenarios and a main use case is to establish an OSCORE security context. By reusing COSE for cryptography, CBOR for encoding, and CoAP for transport, the additional code footprint can be kept very low.

Table of Contents

1. Introduction

Security at the application layer provides an attractive option for protecting Internet of Things (IoT) deployments, for example where transport layer security is not sufficient [I-D.hartke-core-e2e-security-reqs] or where the protection needs to work over a variety of underlying protocols. IoT devices may be constrained in various ways, including memory, storage, processing capacity, and energy [RFC7228]. A method for protecting individual messages at the application layer suitable for constrained devices, is provided by CBOR Object Signing and Encryption (COSE) [RFC8152]), which builds on the Concise Binary Object Representation (CBOR) [I-D.ietf-cbor-7049bis]. Object Security for Constrained RESTful Environments (OSCORE) [RFC8613] is a method for application-layer protection of the Constrained Application Protocol (CoAP), using COSE.

In order for a communication session to provide forward secrecy, the communicating parties can run an Elliptic Curve Diffie-Hellman (ECDH) key exchange protocol with ephemeral keys, from which shared key material can be derived. This document specifies Ephemeral Diffie-Hellman Over COSE (EDHOC), a lightweight key exchange protocol providing perfect forward secrecy and identity protection. Authentication is based on credentials established out of band, e.g. from a trusted third party, such as an Authorization Server as specified by [I-D.ietf-ace-oauth-authz]. EDHOC supports authentication using pre-shared keys (PSK), raw public keys (RPK), and public key certificates. After successful completion of the EDHOC protocol, application keys and other application specific data can be derived using the EDHOC-Exporter interface. A main use case for EDHOC is to establish an OSCORE security context. EDHOC uses COSE for cryptography, CBOR for encoding, and CoAP for transport. By reusing existing libraries, the additional code footprint can be kept very low. Note that this document focuses on authentication and key establishment: for integration with authorization of resource access, refer to [I-D.ietf-ace-oscore-profile].

EDHOC is designed to work in highly constrained scenarios making it especially suitable for network technologies such as Cellular IoT, 6TiSCH [I-D.ietf-6tisch-dtsecurity-zerotouch-join], and LoRaWAN [LoRa1][LoRa2]. These network technologies are characterized by their low throughput, low power consumption, and small frame sizes. Compared to the DTLS 1.3 handshake [I-D.ietf-tls-dtls13] with ECDH and connection ID, the number of bytes in EDHOC is less than 1/4 when PSK authentication is used and less than 1/3 when RPK authentication is used, see [I-D.ietf-lwig-security-protocol-comparison]. Typical message sizes for EDHOC with pre-shared keys, raw public keys, and X.509 certificates are shown in Figure 1.

	PSK	RPK	x5t	x5chain
message_1	40	38	38	38
message_2	45	114	126	116 + Certificate chain
message_3	11	80	91	81 + Certificate chain
Total	96	232	255	235 + Certificate chains

Figure 1: Typical message sizes in bytes

The ECDH exchange and the key derivation follow [SIGMA], NIST SP-800-56A [SP-800-56A], and HKDF [RFC5869]. CBOR [I-D.ietf-cbor-7049bis] and COSE [RFC8152] are used to implement these standards. The use of COSE provides crypto agility and enables use of future algorithms and headers designed for constrained IoT.

This document is organized as follows: Section 2 describes how EDHOC builds on SIGMA-I, Section 3 specifies general properties of EDHOC, including message flow, formatting of the ephemeral public keys, and key derivation, Section 4 specifies EDHOC with asymmetric key authentication, Section 5 specifies EDHOC with symmetric key authentication, Section 6 specifies the EDHOC error message, and Section 7 describes how EDHOC can be transferred in CoAP and used to establish an OSCORE security context.

1.1. Rationale for EDHOC

Many constrained IoT systems today do not use any security at all, and when they do, they often do not follow best practices. One reason is that many current security protocols are not designed with constrained IoT in mind. Constrained IoT systems often deal with personal information, valuable business data, and actuators interacting with the physical world. Not only do such systems need security and privacy, they often need end-to-end protection with

source authentication and perfect forward secrecy. EDHOC and OSCORE [RFC8613] enables security following current best practices to devices and systems where current security protocols are impractical.

EDHOC is optimized for small message sizes and can therefore be sent over a small number of radio frames. The message size of a key exchange protocol may have a large impact on the performance of an IoT deployment, especially in noisy environments. For example, in a network bootstrapping setting a large number of devices turned on in a short period of time may result in large latencies caused by parallel key exchanges. Requirements on network formation time in constrained environments can be translated into key exchange overhead. In networks technologies with transmission back-off time, each additional frame significantly increases the latency even if no other devices are transmitting.

Power consumption for wireless devices is highly dependent on message transmission, listening, and reception. For devices that only send a few bytes occasionally, the battery lifetime may be significantly reduced by a heavy key exchange protocol. Moreover, a key exchange may need to be executed more than once, e.g. due to a device losing power or rebooting for other reasons.

EDHOC is adapted to primitives and protocols designed for the Internet of Things: EDHOC is built on CBOR and COSE which enables small message overhead and efficient parsing in constrained devices. EDHOC is not bound to a particular transport layer, but it is recommended to transport the EDHOC message in CoAP payloads. EDHOC is not bound to a particular communication security protocol but works off-the-shelf with OSCORE [RFC8613] providing the necessary input parameters with required properties. Maximum code complexity (ROM/Flash) is often a constraint in many devices and by reusing already existing libraries, the additional code footprint for EDHOC + OSCORE can be kept very low.

1.2. Terminology and Requirements Language

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

The word "encryption" without qualification always refers to authenticated encryption, in practice implemented with an Authenticated Encryption with Additional Data (AEAD) algorithm, see [RFC5116].

Readers are expected to be familiar with the terms and concepts described in CBOR [I-D.ietf-cbor-7049bis], COSE [RFC8152], and CDDL [RFC8610]. The Concise Data Definition Language (CDDL) is used to express CBOR data structures [I-D.ietf-cbor-7049bis]. Examples of CBOR and CDDL are provided in Appendix A.1.

2. Background

SIGMA (SIGn-and-MAc) is a family of theoretical protocols with a large number of variants [SIGMA]. Like IKEv2 and (D)TLS 1.3 [RFC8446], EDHOC is built on a variant of the SIGMA protocol which provide identity protection of the initiator (SIGMA-I), and like (D)TLS 1.3, EDHOC implements the SIGMA-I variant as Sign-then-MAC. The SIGMA-I protocol using an authenticated encryption algorithm is shown in Figure 2.

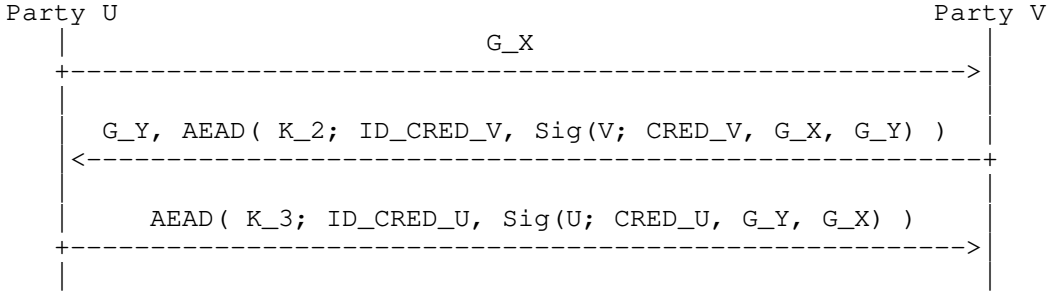

Figure 2: Authenticated encryption variant of the SIGMA-I protocol.

The parties exchanging messages are called "U" and "V". They exchange identities and ephemeral public keys, compute the shared secret, and derive symmetric application keys.

- G_X and G_Y are the ECDH ephemeral public keys of U and V, respectively.
- CRED_U and CRED_V are the credentials containing the public authentication keys of U and V, respectively.
- ID_CRED_U and ID_CRED_V are data enabling the recipient party to retrieve the credential of U and V, respectively.
- Sig(U; .) and S(V; .) denote signatures made with the private authentication key of U and V, respectively.
- AEAD(K; .) denotes authenticated encryption with additional data using the key K derived from the shared secret. The authenticated

encryption MUST NOT be replaced by plain encryption, see Section 8.

In order to create a "full-fledged" protocol some additional protocol elements are needed. EDHOC adds:

- Explicit connection identifiers C_U, C_V chosen by U and V, respectively, enabling the recipient to find the protocol state.
- Transcript hashes TH_2, TH_3, TH_4 used for key derivation and as additional authenticated data.
- Computationally independent keys derived from the ECDH shared secret and used for encryption of different messages.
- Verification of a common preferred cipher suite (AEAD algorithm, ECDH algorithm, ECDH curve, signature algorithm):
  - U lists supported cipher suites in order of preference
  - V verifies that the selected cipher suite is the first supported cipher suite
- Method types and error handling.
- Transport of opaque application defined data.

EDHOC is designed to encrypt and integrity protect as much information as possible, and all symmetric keys are derived using as much previous information as possible. EDHOC is furthermore designed to be as compact and lightweight as possible, in terms of message sizes, processing, and the ability to reuse already existing CBOR, COSE, and CoAP libraries.

To simplify for implementors, the use of CBOR in EDHOC is summarized in Appendix A and test vectors including CBOR diagnostic notation are given in Appendix C.

3. EDHOC Overview

EDHOC consists of three flights (message_1, message_2, message_3) that maps directly to the three messages in SIGMA-I, plus an EDHOC error message. EDHOC messages are CBOR Sequences [I-D.ietf-cbor-sequence], where the first data item of message_1 is an int (TYPE) specifying the method (asymmetric, symmetric) and the correlation properties of the transport used.

While EDHOC uses the COSE_Key, COSE_Sign1, and COSE_Encrypt0 structures, only a subset of the parameters is included in the EDHOC messages. After creating EDHOC message_3, Party U can derive symmetric application keys, and application protected data can therefore be sent in parallel with EDHOC message_3. The application may protect data using the algorithms (AEAD, HMAC, etc.) in the selected cipher suite and the connection identifiers (C_U, C_V). EDHOC may be used with the media type application/edhoc defined in Section 9.

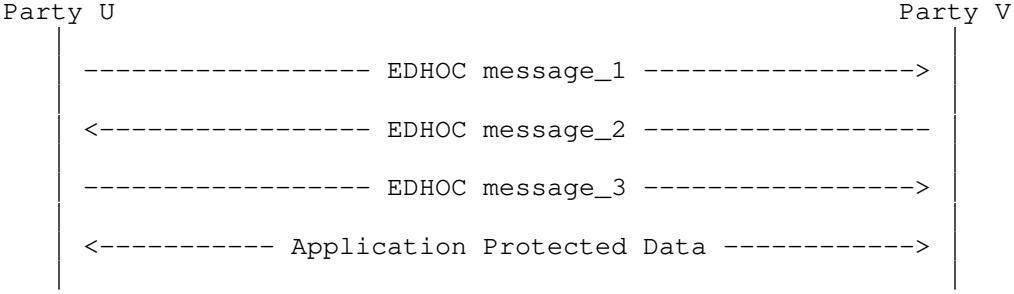

Figure 3: EDHOC message flow

The EDHOC message exchange may be authenticated using pre-shared keys (PSK), raw public keys (RPK), or public key certificates. EDHOC assumes the existence of mechanisms (certification authority, manual distribution, etc.) for binding identities with authentication keys (public or pre-shared). When a public key infrastructure is used, the identity is included in the certificate and bound to the authentication key by trust in the certification authority. When the credential is manually distributed (PSK, RPK, self-signed certificate), the identity and authentication key is distributed out-of-band and bound together by trust in the distribution method. EDHOC with symmetric key authentication is very similar to EDHOC with asymmetric key authentication, the difference being that information is only MACed, not signed, and that session keys are derived from the ECDH shared secret and the PSK.

EDHOC allows opaque application data (UAD and PAD) to be sent in the EDHOC messages. Unprotected Application Data (UAD_1, UAD_2) may be sent in message_1 and message_2 and can be e.g. be used to transfer access tokens that are protected outside of EDHOC. Protected application data (PAD_3) may be used to transfer any application data in message_3.

Cryptographically, EDHOC does not put requirements on the lower layers. EDHOC is not bound to a particular transport layer, and can be used in environments without IP. It is recommended to transport

the EDHOC message in CoAP payloads, see Section 7. An implementation may support only Party U or only Party V.

3.1. Cipher Suites

EDHOC cipher suites consist of an ordered set of COSE algorithms: an AEAD algorithm, an HMAC algorithm, an ECDH curve, a signature algorithm, and signature algorithm parameters. The signature algorithm is not used when EDHOC is authenticated with symmetric keys. Each cipher suite is either identified with a pre-defined int label or with an array of labels and values from the COSE Algorithms and Elliptic Curves registries.

```
suite = int / [ 4*4 algs: int / tstr, ? para: any ]
```

This document specifies two pre-defined cipher suites.

0. [10, 5, 4, -8, 6]
   (AES-CCM-16-64-128, HMAC 256/256, X25519, EdDSA, Ed25519)

1. [10, 5, 1, -7, 1]
   (AES-CCM-16-64-128, HMAC 256/256, P-256, ES256, P-256)

3.2. Ephemeral Public Keys

The ECDH ephemeral public keys are formatted as a COSE_Key of type EC2 or OKP according to Sections 13.1 and 13.2 of [RFC8152], but only the x-coordinate is included in the EDHOC messages. For Elliptic Curve Keys of type EC2, compact representation as per [RFC6090] MAY be used also in the COSE_Key. If the COSE implementation requires an y-coordinate, any of the possible values of the y-coordinate can be used, see Appendix C of [RFC6090]. COSE [RFC8152] always use compact output for Elliptic Curve Keys of type EC2.

3.3. Key Derivation

Key and IV derivation SHALL be performed with HKDF [RFC5869] following the specification in Section 11 of [RFC8152] using the HMAC algorithm in the selected cipher suite. The pseudorandom key (PRK) is derived using HKDF-Extract [RFC5869]

```
PRK = HKDF-Extract( salt, IKM )
```

with the following input:

- The salt SHALL be the PSK when EDHOC is authenticated with symmetric keys, and the empty byte string when EDHOC is authenticated with asymmetric keys. The PSK is used as 'salt' to

simplify implementation. Note that [RFC5869] specifies that if the salt is not provided, it is set to a string of zeros (see Section 2.2 of [RFC5869]). For implementation purposes, not providing the salt is the same as setting the salt to the empty byte string.

- The input keying material (IKM) SHALL be the ECDH shared secret G_XY as defined in Section 12.4.1 of [RFC8152]. When using the curve25519, the ECDH shared secret is the output of the X25519 function [RFC7748].

Example: Assuming use of HMAC 256/256 the extract phase of HKDF produces a PRK as follows:

```
PRK = HMAC-SHA-256( salt, G_XY )
```

where salt = 0x (the empty byte string) in the asymmetric case and salt = PSK in the symmetric case.

The keys and IVs used in EDHOC are derived from PRK using HKDF-Expand [RFC5869]

```
OKM = HKDF-Expand( PRK, info, L )
```

where L is the length of output keying material (OKM) in bytes and info is the CBOR encoding of a COSE_KDF_Context

```
info = [
  AlgorithmID,
  [ null, null, null ],
  [ null, null, null ],
  [ keyDataLength, h'', other ]
]
```

where

- AlgorithmID is an int or tstr, see below
- keyDataLength is a uint set to the length of output keying material in bits, see below
- other is a bstr set to one of the transcript hashes TH_2, TH_3, or TH_4 as defined in Sections 4.3.1, 4.4.1, and 3.3.1.

For message_2 and message_3, the keys K_2 and K_3 SHALL be derived using transcript hashes TH_2 and TH_3 respectively. The key SHALL be derived using AlgorithmID set to the integer value of the AEAD in the

selected cipher suite, and keyDataLength equal to the key length of the AEAD.

If the AEAD algorithm uses an IV, then IV_2 and IV_3 for message_2 and message_3 SHALL be derived using the transcript hashes TH_2 and TH_3 respectively. The IV SHALL be derived using AlgorithmID = "IV-GENERATION" as specified in Section 12.1.2. of [RFC8152], and keyDataLength equal to the IV length of the AEAD.

Assuming the output OKM length L is smaller than the hash function output size, the expand phase of HKDF consists of a single HMAC invocation

```
OKM = first L bytes of HMAC( PRK, info || 0x01 )
```

where || means byte string concatenation.

Example: Assuming use of the algorithm AES-CCM-16-64-128 and HMAC 256/256, K_i and IV_i are therefore the first 16 and 13 bytes, respectively, of

```
HMAC-SHA-256( PRK, info || 0x01 )
```

calculated with (AlgorithmID, keyDataLength) = (10, 128) and (AlgorithmID, keyDataLength) = ("IV-GENERATION", 104), respectively.

3.3.1. EDHOC-Exporter Interface

Application keys and other application specific data can be derived using the EDHOC-Exporter interface defined as:

```
EDHOC-Exporter( label, length ) = HKDF-Expand( PRK, info, length )
```

The output of the EDHOC-Exporter function SHALL be derived using AlgorithmID = label, keyDataLength = 8 * length, and other = TH_4 where label is a tstr defined by the application and length is a uint defined by the application. The label SHALL be different for each different exporter value. The transcript hash TH_4 is a CBOR encoded bstr and the input to the hash function is a CBOR Sequence.

```
TH_4 = H( TH_3, CIPHERTEXT_3 )
```

where H() is the hash function in the HMAC algorithm. Example use of the EDHOC-Exporter is given in Sections 3.3.2 and 7.1.1.

3.3.2. EDHOC PSK Chaining

An application using EDHOC may want to derive new PSKs to use for authentication in future EDHOC exchanges. In this case, the new PSK and the ID_PSK 'kid_value' parameter SHOULD be derived as follows where length is the key length (in bytes) of the AEAD Algorithm.

```
PSK    = EDHOC-Exporter( "EDHOC Chaining PSK", length )
ID_PSK = EDHOC-Exporter( "EDHOC Chaining ID_PSK", 4 )
```

4. EDHOC Authenticated with Asymmetric Keys

4.1. Overview

EDHOC supports authentication with raw public keys (RPK) and public key certificates with the requirements that:

- Only Party V SHALL have access to the private authentication key of Party V,
- Only Party U SHALL have access to the private authentication key of Party U,
- Party U is able to retrieve Party V's public authentication key using ID_CRED_V,
- Party V is able to retrieve Party U's public authentication key using ID_CRED_U,

where the identifiers ID_CRED_U and ID_CRED_V are COSE header_maps, i.e. a CBOR map containing COSE Common Header Parameters, see [RFC8152]). ID_CRED_U and ID_CRED_V need to contain parameters that can identify a public authentication key, see Appendix A.2. In the following we give some examples of possible COSE header parameters.

Raw public keys are most optimally stored as COSE_Key objects and identified with a 'kid' parameter (see [RFC8152]):

- ID_CRED_x = { 4 : kid_value }, where kid_value : bstr, for x = U or V.

Public key certificates can be identified in different ways. Several header parameters for identifying X.509 certificates are defined in [I-D.ietf-cose-x509] (the exact labels are TBD):

- by a hash value with the 'x5t' parameter;
  - ID_CRED_x = { TBD1 : COSE_CertHash }, for x = U or V,

- by a URL with the 'x5u' parameter;
  - ID_CRED_x = { TBD2 : uri }, for x = U or V,
- or by a bag of certificates with the 'x5bag' parameter;
  - ID_CRED_x = { TBD3 : COSE_X509 }, for x = U or V.
- by a certificate chain with the 'x5chain' parameter;
  - ID_CRED_x = { TBD4 : COSE_X509 }, for x = U or V,

In the latter two examples, ID_CRED_U and ID_CRED_V contain the actual credential used for authentication. The purpose of ID_CRED_U and ID_CRED_V is to facilitate retrieval of a public authentication key and when they do not contain the actual credential, they may be very short. It is RECOMMENDED that they uniquely identify the public authentication key as the recipient may otherwise have to try several keys. ID_CRED_U and ID_CRED_V are transported in the ciphertext, see Section 4.3.2 and Section 4.4.2.

The actual credentials CRED_U and CRED_V (e.g. a COSE_Key or a single X.509 certificate) are signed by party U and V, respectively to prevent duplicate-signature key selection (DSKS) attacks, see Section 4.4.1 and Section 4.3.1. Party U and Party V MAY use different types of credentials, e.g. one uses RPK and the other uses certificate. When included in the signature payload, COSE_Keys of type OKP SHALL only include the parameters 1 (kty), -1 (crv), and -2 (x-coordinate). COSE_Keys of type EC2 SHALL only include the parameters 1 (kty), -1 (crv), -2 (x-coordinate), and -3 (y-coordinate). The parameters SHALL be encoded in decreasing order.

The connection identifiers C_U and C_V do not have any cryptographic purpose in EDHOC. They contain information facilitating retrieval of the protocol state and may therefore be very short. The connection identifier MAY be used with an application protocol (e.g. OSCORE) for which EDHOC establishes keys, in which case the connection identifiers SHALL adhere to the requirements for that protocol. Each party choses a connection identifier it desires the other party to use in outgoing messages.

The first data item of message_1 is an int TYPE = 4 * method + corr specifying the method and the correlation properties of the transport used. corr = 0 is used when there is no external correlation mechanism. corr = 1 is used when there is an external correlation mechanism (e.g. the Token in CoAP) that enables Party U to correlate message_1 and message_2. corr = 2 is used when there is an external correlation mechanism that enables Party V to correlate message_2 and

message_3. corr = 3 is used when there is an external correlation mechanism that enables the parties to correlate all the messages. The use of the correlation parameter is exemplified in Section 7.1.

1 byte connection and credential identifiers are realistic in many scenarios as most constrained devices only have a few keys and connections. In cases where a node only has one connection or key, the identifiers may even be the empty byte string.

EDHOC with asymmetric key authentication is illustrated in Figure 4.

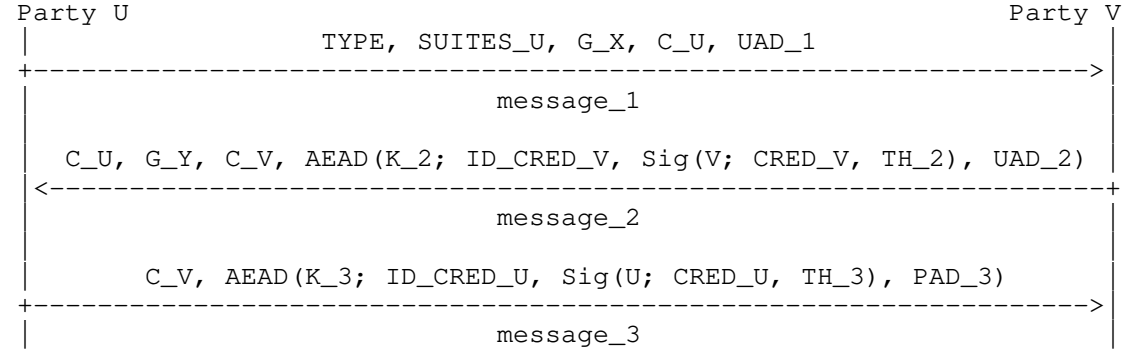

Figure 4: Overview of EDHOC with asymmetric key authentication.

4.2. EDHOC Message 1

4.2.1. Formatting of Message 1

message_1 SHALL be a CBOR Sequence (see Appendix A.1) as defined below

```
message_1 = (
  TYPE : int,
  SUITES_U : suite / [ index : uint, 2* suite ],
  G_X : bstr,
  C_U : bstr,
  ? UAD_1 : bstr,
)
```

where:

- TYPE = 4 * method + corr, where the method = 0 and the correlation parameter corr is chosen based on the transport and determines which connection identifiers that are omitted (see Section 4.1).

- SUITES_U - cipher suites which Party U supports in order of decreasing preference. One cipher suite is selected. If a single cipher suite is conveyed then that cipher suite is selected. If multiple cipher suites are conveyed then zero-based index (i.e. 0 for the first suite, 1 for the second suite, etc.) identifies the selected cipher suite out of the array elements listing the cipher suites (see Section 6).

- G_X - the x-coordinate of the ephemeral public key of Party U

- C_U - variable length connection identifier

- UAD_1 - bstr containing unprotected opaque application data

4.2.2. Party U Processing of Message 1

Party U SHALL compose message_1 as follows:

- The supported cipher suites and the order of preference MUST NOT be changed based on previous error messages. However, the list SUITES_U sent to Party V MAY be truncated such that cipher suites which are the least preferred are omitted. The amount of truncation MAY be changed between sessions, e.g. based on previous error messages (see next bullet), but all cipher suites which are more preferred than the least preferred cipher suite in the list MUST be included in the list.

- Determine the cipher suite to use with Party V in message_1. If Party U previously received from Party V an error message to message_1 with diagnostic payload identifying a cipher suite that U supports, then U SHALL use that cipher suite. Otherwise the first cipher suite in SUITES_U MUST be used.

- Generate an ephemeral ECDH key pair as specified in Section 5 of [SP-800-56A] using the curve in the selected cipher suite. Let G_X be the x-coordinate of the ephemeral public key.

- Choose a connection identifier C_U and store it for the length of the protocol.

- Encode message_1 as a sequence of CBOR encoded data items as specified in Section 4.2.1

4.2.3. Party V Processing of Message 1

Party V SHALL process message_1 as follows:

- Decode message_1 (see Appendix A.1).

- Verify that the selected cipher suite is supported and that no prior cipher suites in SUITES_U are supported.

- Validate that there is a solution to the curve definition for the given x-coordinate G_X.

- Pass UAD_1 to the application.

If any verification step fails, Party V MUST send an EDHOC error message back, formatted as defined in Section 6, and the protocol MUST be discontinued. If V does not support the selected cipher suite, then SUITES_V MUST include one or more supported cipher suites. If V does not support the selected cipher suite, but supports another cipher suite in SUITES_U, then SUITES_V MUST include the first supported cipher suite in SUITES_U.

4.3. EDHOC Message 2

4.3.1. Formatting of Message 2

message_2 and data_2 SHALL be CBOR Sequences (see Appendix A.1) as defined below

```
message_2 = (
  data_2,
  CIPHERTEXT_2 : bstr,
)

data_2 = (
  ? C_U : bstr,
  G_Y : bstr,
  C_V : bstr,
)
```

where:

- G_Y - the x-coordinate of the ephemeral public key of Party V

- C_V - variable length connection identifier

4.3.2. Party V Processing of Message 2

Party V SHALL compose message_2 as follows:

- If TYPE mod 4 equals 1 or 3, C_U is omitted, otherwise C_U is not omitted.

- Generate an ephemeral ECDH key pair as specified in Section 5 of [SP-800-56A] using the curve in the selected cipher suite. Let G_Y be the x-coordinate of the ephemeral public key.

- Choose a connection identifier C_V and store it for the length of the protocol.

- Compute the transcript hash TH_2 = H(message_1, data_2) where H() is the hash function in the HMAC algorithm. The transcript hash TH_2 is a CBOR encoded bstr and the input to the hash function is a CBOR Sequence.

- Compute COSE_Sign1 as defined in Section 4.4 of [RFC8152], using the signature algorithm in the selected cipher suite, the private authentication key of Party V, and the parameters below. Note that only 'signature' of the COSE_Sign1 object is used to create message_2, see next bullet. The unprotected header (not included in the EDHOC message) MAY contain parameters (e.g. 'alg').

  * protected = bstr .cbor ID_CRED_V

  * payload = CRED_V

  * external_aad = TH_2

  * ID_CRED_V - identifier to facilitate retrieval of CRED_V, see Section 4.1

  * CRED_V - bstr credential containing the credential of Party V, e.g. its public authentication key or X.509 certificate see Section 4.1. The public key must be a signature key. Note that if objects that are not bstr are used, such as COSE_Key for public authentication keys, these objects must be wrapped in a CBOR bstr.

  COSE constructs the input to the Signature Algorithm as follows:

  * The key is the private authentication key of V.

  * The message M to be signed is the CBOR encoding of:

    ["Signature1", << ID_CRED_V >>, TH_2, CRED_V]

- Compute COSE_Encrypt0 as defined in Section 5.3 of [RFC8152], with the AEAD algorithm in the selected cipher suite, K_2, IV_2, and the parameters below. Note that only 'ciphertext' of the COSE_Encrypt0 object is used to create message_2, see next bullet. The protected header SHALL be empty. The unprotected header (not

included in the EDHOC message) MAY contain parameters (e.g. 'alg').

* plaintext = (ID_CRED_V / kid_value, signature, ? UAD_2)

* external_aad = TH_2

* UAD_2 = bstr containing opaque unprotected application data

where signature is taken from the COSE_Sign1 object, ID_CRED_V is a COSE header_map (i.e. a CBOR map containing COSE Common Header Parameters, see [RFC8152]), and kid_value is a bstr. If ID_CRED_V contains a single 'kid' parameter, i.e., ID_CRED_V = { 4 : kid_value }, only kid_value is conveyed in the plaintext.

COSE constructs the input to the AEAD [RFC5116] as follows:

* Key K = K_2

* Nonce N = IV_2

* Plaintext P = (ID_CRED_V / kid_value, signature, ? UAD_2)

* Associated data A = ["Encrypt0", h'', TH_2]

- Encode message_2 as a sequence of CBOR encoded data items as specified in Section 4.3.1. CIPHERTEXT_2 is the COSE_Encrypt0 ciphertext.

4.3.3. Party U Processing of Message 2

Party U SHALL process message_2 as follows:

- Decode message_2 (see Appendix A.1).

- Retrieve the protocol state using the connection identifier C_U and/or other external information such as the CoAP Token and the 5-tuple.

- Validate that there is a solution to the curve definition for the given x-coordinate G_Y.

- Decrypt and verify COSE_Encrypt0 as defined in Section 5.3 of [RFC8152], with the AEAD algorithm in the selected cipher suite, K_2, and IV_2.

- Verify COSE_Sign1 as defined in Section 4.4 of [RFC8152], using the signature algorithm in the selected cipher suite and the public authentication key of Party V.

If any verification step fails, Party U MUST send an EDHOC error message back, formatted as defined in Section 6, and the protocol MUST be discontinued.

4.4. EDHOC Message 3

4.4.1. Formatting of Message 3

message_3 and data_3 SHALL be CBOR Sequences (see Appendix A.1) as defined below

```
message_3 = (
  data_3,
  CIPHERTEXT_3 : bstr,
)

data_3 = (
  ? C_V : bstr,
)
```

4.4.2. Party U Processing of Message 3

Party U SHALL compose message_3 as follows:

- If TYPE mod 4 equals 2 or 3, C_V is omitted, otherwise C_V is not omitted.

- Compute the transcript hash TH_3 = H(TH_2 , CIPHERTEXT_2, data_3) where H() is the hash function in the HMAC algorithm. The transcript hash TH_3 is a CBOR encoded bstr and the input to the hash function is a CBOR Sequence.

- Compute COSE_Sign1 as defined in Section 4.4 of [RFC8152], using the signature algorithm in the selected cipher suite, the private authentication key of Party U, and the parameters below. Note that only 'signature' of the COSE_Sign1 object is used to create message_3, see next bullet. The unprotected header (not included in the EDHOC message) MAY contain parameters (e.g. 'alg').

  * protected = bstr .cbor ID_CRED_U

  * payload = CRED_U

  * external_aad = TH_3

* ID_CRED_U - identifier to facilitate retrieval of CRED_U, see Section 4.1

* CRED_U - bstr credential containing the credential of Party U, e.g. its public authentication key or X.509 certificate see Section 4.1. The public key must be a signature key. Note that if objects that are not bstr are used, such as COSE_Key for public authentication keys, these objects must be wrapped in a CBOR bstr.

COSE constructs the input to the Signature Algorithm as follows:

* The key is the private authentication key of U.

* The message M to be signed is the CBOR encoding of:

  ["Signature1", << ID_CRED_U >>, TH_3, CRED_U]

o Compute COSE_Encrypt0 as defined in Section 5.3 of [RFC8152], with the AEAD algorithm in the selected cipher suite, K_3, and IV_3 and the parameters below. Note that only 'ciphertext' of the COSE_Encrypt0 object is used to create message_3, see next bullet. The protected header SHALL be empty. The unprotected header (not included in the EDHOC message) MAY contain parameters (e.g. 'alg').

  * plaintext = (ID_CRED_U / kid_value, signature, ? PAD_3)

  * external_aad = TH_3

  * PAD_3 = bstr containing opaque protected application data

  where signature is taken from the COSE_Sign1 object, ID_CRED_U is a COSE header_map (i.e. a CBOR map containing COSE Common Header Parameters, see [RFC8152]), and kid_value is a bstr. If ID_CRED_U contains a single 'kid' parameter, i.e., ID_CRED_U = { 4 : kid_value }, only kid_value is conveyed in the plaintext.

  COSE constructs the input to the AEAD [RFC5116] as follows:

  * Key K = K_3

  * Nonce N = IV_2

  * Plaintext P = (ID_CRED_U / kid_value, signature, ? PAD_3)

  * Associated data A = ["Encrypt0", h'', TH_3]

- Encode message_3 as a sequence of CBOR encoded data items as specified in Section 4.4.1. CIPHERTEXT_3 is the COSE_Encrypt0 ciphertext.

- Pass the connection identifiers (C_U, C_V) and the selected cipher suite to the application. The application can now derive application keys using the EDHOC-Exporter interface.

4.4.3. Party V Processing of Message 3

Party V SHALL process message_3 as follows:

- Decode message_3 (see Appendix A.1).

- Retrieve the protocol state using the connection identifier C_V and/or other external information such as the CoAP Token and the 5-tuple.

- Decrypt and verify COSE_Encrypt0 as defined in Section 5.3 of [RFC8152], with the AEAD algorithm in the selected cipher suite, K_3, and IV_3.

- Verify COSE_Sign1 as defined in Section 4.4 of [RFC8152], using the signature algorithm in the selected cipher suite and the public authentication key of Party U.

If any verification step fails, Party V MUST send an EDHOC error message back, formatted as defined in Section 6, and the protocol MUST be discontinued.

- Pass PAD_3, the connection identifiers (C_U, C_V), and the selected cipher suite to the application. The application can now derive application keys using the EDHOC-Exporter interface.

5. EDHOC Authenticated with Symmetric Keys

5.1. Overview

EDHOC supports authentication with pre-shared keys. Party U and V are assumed to have a pre-shared key (PSK) with a good amount of randomness and the requirement that:

- Only Party U and Party V SHALL have access to the PSK,

- Party V is able to retrieve the PSK using ID_PSK.

where the identifier ID_PSK is a COSE header_map (i.e. a CBOR map containing COSE Common Header Parameters, see [RFC8152]) containing

COSE header parameter that can identify a pre-shared key. Pre-shared keys are typically stored as COSE_Key objects and identified with a 'kid' parameter (see [RFC8152]):

o ID_PSK = { 4 : kid_value } , where kid_value : bstr

The purpose of ID_PSK is to facilitate retrieval of the PSK and in the case a 'kid' parameter is used it may be very short. It is RECOMMENDED that it uniquely identify the PSK as the recipient may otherwise have to try several keys.

EDHOC with symmetric key authentication is illustrated in Figure 5.

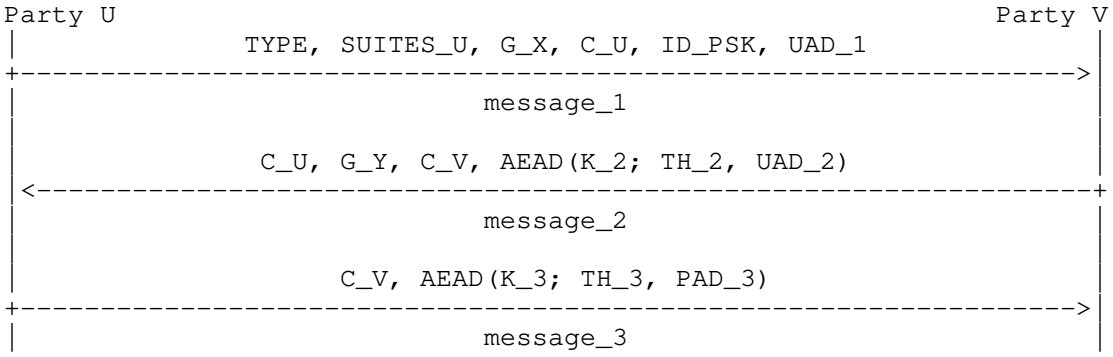

Figure 5: Overview of EDHOC with symmetric key authentication.

EDHOC with symmetric key authentication is very similar to EDHOC with asymmetric key authentication. In the following subsections the differences compared to EDHOC with asymmetric key authentication are described.

5.2. EDHOC Message 1

5.2.1. Formatting of Message 1

message_1 SHALL be a CBOR Sequence (see Appendix A.1) as defined below

```
message_1 = (
  TYPE : int,
  SUITES_U : suite / [ index : uint, 2* suite ],
  G_X : bstr,
  C_U : bstr,
  ID_PSK : header_map // kid_value : bstr,
  ? UAD_1 : bstr,
)
```

where:

- TYPE = 4 * method + corr, where the method = 1 and the connection parameter corr is chosen based on the transport and determines which connection identifiers that are omitted (see Section 4.1).

- ID_PSK - identifier to facilitate retrieval of the pre-shared key. If ID_PSK contains a single 'kid' parameter, i.e., ID_PSK = { 4 : kid_value }, with kid_value: bstr, only kid_value is conveyed.

5.3. EDHOC Message 2

5.3.1. Processing of Message 2

- COSE_Sign1 is not used.

- COSE_Encrypt0 is computed as defined in Section 5.3 of [RFC8152], with the AEAD algorithm in the selected cipher suite, K_2, IV_2, and the following parameters. The protected header SHALL be empty. The unprotected header MAY contain parameters (e.g. 'alg').

  * external_aad = TH_2

  * plaintext = ? UAD_2

  * UAD_2 = bstr containing opaque unprotected application data

5.4. EDHOC Message 3

5.4.1. Processing of Message 3

- COSE_Sign1 is not used.

- COSE_Encrypt0 is computed as defined in Section 5.3 of [RFC8152], with the AEAD algorithm in the selected cipher suite, K_3, IV_3, and the following parameters. The protected header SHALL be empty. The unprotected header MAY contain parameters (e.g. 'alg').

  * external_aad = TH_3

  * plaintext = ? PAD_3

  * PAD_3 = bstr containing opaque protected application data

6. Error Handling

6.1. EDHOC Error Message

This section defines a message format for the EDHOC error message, used during the protocol. An EDHOC error message can be sent by both parties as a reply to any non-error EDHOC message. After sending an error message, the protocol MUST be discontinued. Errors at the EDHOC layer are sent as normal successful messages in the lower layers (e.g. CoAP POST and 2.04 Changed). An advantage of using such a construction is to avoid issues created by usage of cross protocol proxies (e.g. UDP to TCP).

error SHALL be a CBOR Sequence (see Appendix A.1) as defined below

```
error = (
  ? C_x : bstr,
  ERR_MSG : tstr,
  ? SUITES_V : suite / [ 2* suite ],
)
```

where:

- C_x - if error is sent by Party V and TYPE mod 4 equals 0 or 2 then C_x is set to C_U, else if error is sent by Party U and TYPE mod 4 equals 0 or 1 then C_x is set to C_V, else C_x is omitted.

- ERR_MSG - text string containing the diagnostic payload, defined in the same way as in Section 5.5.2 of [RFC7252]. ERR_MSG MAY be a 0-length text string.

- SUITES_V - cipher suites from SUITES_U or the EDHOC cipher suites registry that V supports. Note that SUITES_V only contains the values from the EDHOC cipher suites registry and no index. SUITES_V MUST only be included in replies to message_1.

6.1.1. Example Use of EDHOC Error Message with SUITES_V

Assuming that Party U supports the five cipher suites {5, 6, 7, 8, 9} in decreasing order of preference, Figures 6 and 7 show examples of how Party U can truncate SUITES_U and how SUITES_V is used by Party V to give Party U information about the cipher suites that Party V supports. In Figure 6, Party V supports cipher suite 6 but not the selected cipher suite 5.

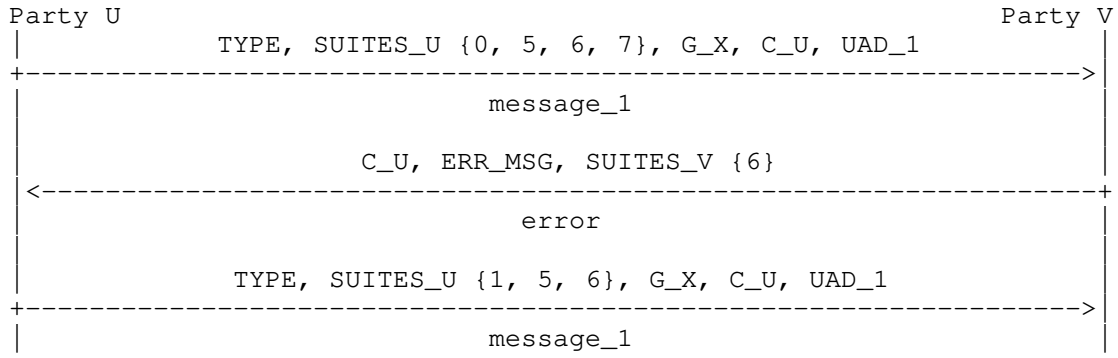

Figure 6: Example use of error message with SUITES_V.

In Figure 7, Party V supports cipher suite 7 but not cipher suites 5 and 6.

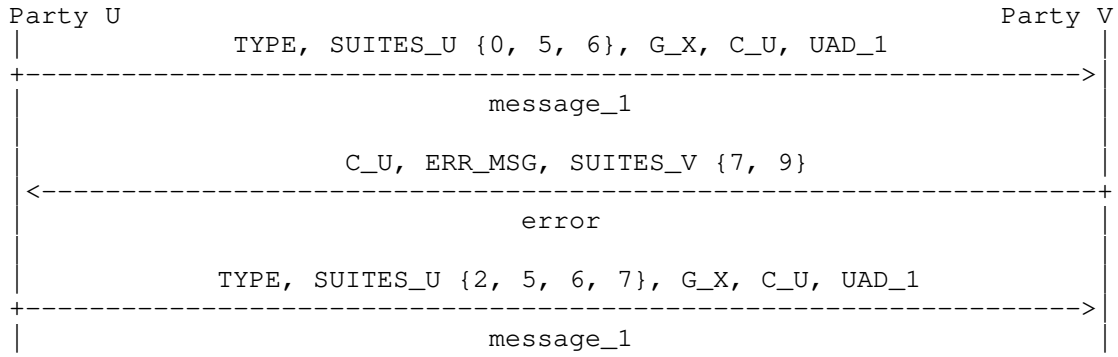

Figure 7: Example use of error message with SUITES_V.

As Party U's list of supported cipher suites and order of preference is fixed, and Party V only accepts message_1 if the selected cipher suite is the first cipher suite in SUITES_U that Party V supports, the parties can verify that the selected cipher suite is the most preferred (by Party U) cipher suite supported by both parties. If the selected cipher suite is not the first cipher suite in SUITES_U that Party V supports, Party V will discontinue the protocol.

7. Transferring EDHOC and Deriving Application Keys

7.1. Transferring EDHOC in CoAP

It is recommended to transport EDHOC as an exchange of CoAP [RFC7252] messages. CoAP is a reliable transport that can preserve packet ordering and handle message duplication. CoAP can also perform

fragmentation and protect against denial of service attacks. It is recommended to carry the EDHOC flights in Confirmable messages, especially if fragmentation is used.

By default, the CoAP client is Party U and the CoAP server is Party V, but the roles SHOULD be chosen to protect the most sensitive identity, see Section 8. By default, EDHOC is transferred in POST requests and 2.04 (Changed) responses to the Uri-Path: "/.well-known/edhoc", but an application may define its own path that can be discovered e.g. using resource directory [I-D.ietf-core-resource-directory].

By default, the message flow is as follows: EDHOC message_1 is sent in the payload of a POST request from the client to the server's resource for EDHOC. EDHOC message_2 or the EDHOC error message is sent from the server to the client in the payload of a 2.04 (Changed) response. EDHOC message_3 or the EDHOC error message is sent from the client to the server's resource in the payload of a POST request. If needed, an EDHOC error message is sent from the server to the client in the payload of a 2.04 (Changed) response.

An example of a successful EDHOC exchange using CoAP is shown in Figure 8. In this case the CoAP Token enables Party U to correlate message_1 and message_2 so the correlation parameter corr = 1.

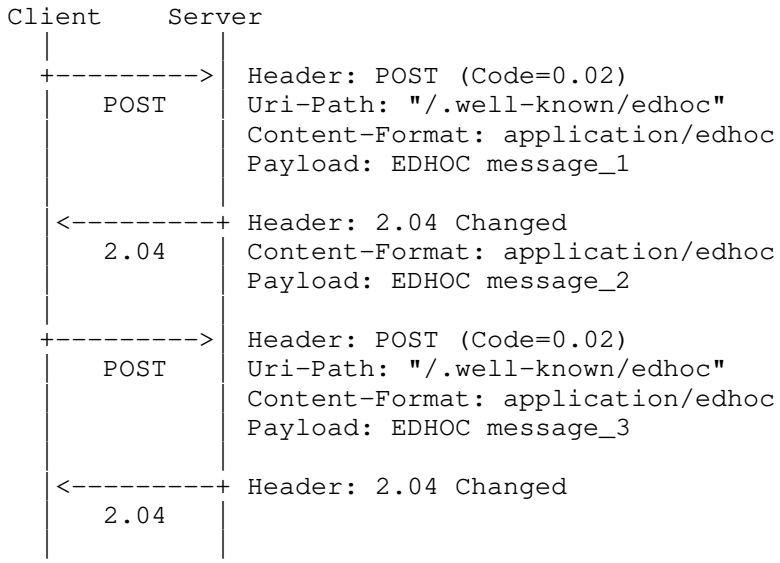

Figure 8: Transferring EDHOC in CoAP

The exchange in Figure 8 protects the client identity against active attackers and the server identity against passive attackers. An alternative exchange that protects the server identity against active attackers and the client identity against passive attackers is shown in Figure 9. In this case the CoAP Token enables Party V to correlate message_2 and message_3 so the correlation parameter corr = 2.

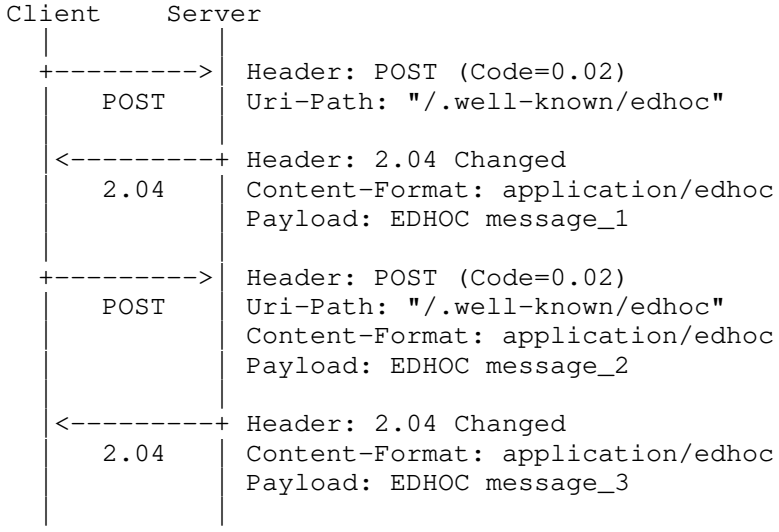

Figure 9: Transferring EDHOC in CoAP

To protect against denial-of-service attacks, the CoAP server MAY respond to the first POST request with a 4.01 (Unauthorized) containing an Echo option [I-D.ietf-core-echo-request-tag]. This forces the initiator to demonstrate its reachability at its apparent network address. If message fragmentation is needed, the EDHOC messages may be fragmented using the CoAP Block-Wise Transfer mechanism [RFC7959].

7.1.1. Deriving an OSCORE Context from EDHOC

When EDHOC is used to derive parameters for OSCORE [RFC8613], the parties must make sure that the EDHOC connection identifiers are unique, i.e. C_V MUST NOT be equal to C_U. The CoAP client and server MUST be able to retrieve the OSCORE protocol state using its chosen connection identifier and optionally other information such as the 5-tuple. In case that the CoAP client is party U and the CoAP server is party V:

- The client's OSCORE Sender ID is C_V and the server's OSCORE Sender ID is C_U, as defined in this document

- The AEAD Algorithm and the HMAC algorithms are the AEAD and HMAC algorithms in the selected cipher suite.

- The Master Secret and Master Salt are derived as follows where length is the key length (in bytes) of the AEAD Algorithm.

  ```
  Master Secret = EDHOC-Exporter( "OSCORE Master Secret", length )
  Master Salt   = EDHOC-Exporter( "OSCORE Master Salt", 8 )
  ```

7.2. Transferring EDHOC over Other Protocols

EDHOC may be transported over a different transport than CoAP. In this case the lower layers need to handle message loss, reordering, message duplication, fragmentation, and denial of service protection.

8. Security Considerations

8.1. Security Properties

EDHOC inherits its security properties from the theoretical SIGMA-I protocol [SIGMA]. Using the terminology from [SIGMA], EDHOC provides perfect forward secrecy, mutual authentication with aliveness, consistency, peer awareness, and identity protection. As described in [SIGMA], peer awareness is provided to Party V, but not to Party U. EDHOC also inherits Key Compromise Impersonation (KCI) resistance from SIGMA-I.

EDHOC with asymmetric authentication offers identity protection of Party U against active attacks and identity protection of Party V against passive attacks. The roles should be assigned to protect the most sensitive identity, typically that which is not possible to infer from routing information in the lower layers.

Compared to [SIGMA], EDHOC adds an explicit method type and expands the message authentication coverage to additional elements such as algorithms, application data, and previous messages. This protects against an attacker replaying messages or injecting messages from another session.

EDHOC also adds negotiation of connection identifiers and downgrade protected negotiation of cryptographic parameters, i.e. an attacker cannot affect the negotiated parameters. A single session of EDHOC does not include negotiation of cipher suites, but it enables Party V to verify that the selected cipher suite is the most preferred cipher suite by U which is supported by both U and V.

As required by [RFC7258], IETF protocols need to mitigate pervasive monitoring when possible. One way to mitigate pervasive monitoring is to use a key exchange that provides perfect forward secrecy. EDHOC therefore only supports methods with perfect forward secrecy. To limit the effect of breaches, it is important to limit the use of symmetrical group keys for bootstrapping. EDHOC therefore strives to make the additional cost of using raw public keys and self-signed certificates as small as possible. Raw public keys and self-signed certificates are not a replacement for a public key infrastructure, but SHOULD be used instead of symmetrical group keys for bootstrapping.

Compromise of the long-term keys (PSK or private authentication keys) does not compromise the security of completed EDHOC exchanges. Compromising the private authentication keys of one party lets the attacker impersonate that compromised party in EDHOC exchanges with other parties, but does not let the attacker impersonate other parties in EDHOC exchanges with the compromised party. Compromising the PSK lets the attacker impersonate Party U in EDHOC exchanges with Party V and impersonate Party V in EDHOC exchanges with Party U. Compromise of the HDKF input parameters (ECDH shared secret and/or PSK) leads to compromise of all session keys derived from that compromised shared secret. Compromise of one session key does not compromise other session keys.

8.2. Cryptographic Considerations

The security of the SIGMA protocol requires the MAC to be bound to the identity of the signer. Hence the message authenticating functionality of the authenticated encryption in EDHOC is critical: authenticated encryption MUST NOT be replaced by plain encryption only, even if authentication is provided at another level or through a different mechanism. EDHOC implements SIGMA-I using the same Sign-then-MAC approach as TLS 1.3.

To reduce message overhead EDHOC does not use explicit nonces and instead rely on the ephemeral public keys to provide randomness to each session. A good amount of randomness is important for the key generation, to provide liveness, and to protect against interleaving attacks. For this reason, the ephemeral keys MUST NOT be reused, and both parties SHALL generate fresh random ephemeral key pairs.

The choice of key length used in the different algorithms needs to be harmonized, so that a sufficient security level is maintained for certificates, EDHOC, and the protection of application data. Party U and V should enforce a minimum security level.

The data rates in many IoT deployments are very limited. Given that the application keys are protected as well as the long-term authentication keys they can often be used for years or even decades before the cryptographic limits are reached. If the application keys established through EDHOC need to be renewed, the communicating parties can derive application keys with other labels or run EDHOC again.

8.3. Cipher Suites

Cipher suite number 0 (AES-CCM-64-64-128, ECDH-SS + HKDF-256, X25519, Ed25519) is mandatory to implement. For many constrained IoT devices it is problematic to support more than one cipher suites, so some deployments with P-256 may not support the mandatory cipher suite. This is not a problem for local deployments.

The HMAC algorithm HMAC 256/64 (HMAC w/ SHA-256 truncated to 64 bits) SHALL NOT be supported for use in EDHOC.

8.4. Unprotected Data

Party U and V must make sure that unprotected data and metadata do not reveal any sensitive information. This also applies for encrypted data sent to an unauthenticated party. In particular, it applies to UAD_1, ID_CRED_V, UAD_2, and ERR_MSG in the asymmetric case, and ID_PSK, UAD_1, and ERR_MSG in the symmetric case. Using the same ID_PSK or UAD_1 in several EDHOC sessions allows passive eavesdroppers to correlate the different sessions. The communicating parties may therefore anonymize ID_PSK. Another consideration is that the list of supported cipher suites may be used to identify the application.

Party U and V must also make sure that unauthenticated data does not trigger any harmful actions. In particular, this applies to UAD_1 and ERR_MSG in the asymmetric case, and ID_PSK, UAD_1, and ERR_MSG in the symmetric case.

8.5. Denial-of-Service

EDHOC itself does not provide countermeasures against Denial-of-Service attacks. By sending a number of new or replayed message_1 an attacker may cause Party V to allocate state, perform cryptographic operations, and amplify messages. To mitigate such attacks, an implementation SHOULD rely on lower layer mechanisms such as the Echo option in CoAP [I-D.ietf-core-echo-request-tag] that forces the initiator to demonstrate reachability at its apparent network address.

8.6. Implementation Considerations

The availability of a secure pseudorandom number generator and truly random seeds are essential for the security of EDHOC. If no true random number generator is available, a truly random seed must be provided from an external source. As each pseudoranom number must only be used once, an implementation need to get a new truly random seed after reboot, or continously store state in nonvolatile memory, see ([RFC8613], Appendix B.1.1) for issues and solution approaches for writing to nonvolatile memory. If ECDSA is supported, "deterministic ECDSA" as specified in [RFC6979] is RECOMMENDED.

The referenced processing instructions in [SP-800-56A] must be complied with, including deleting the intermediate computed values along with any ephemeral ECDH secrets after the key derivation is completed. The ECDH shared secret, keys (K_2, K_3), and IVs (IV_2, IV_3) MUST be secret. Implementations should provide countermeasures to side-channel attacks such as timing attacks.

Party U and V are responsible for verifying the integrity of certificates. The selection of trusted CAs should be done very carefully and certificate revocation should be supported. The private authentication keys and the PSK (even though it is used as salt) MUST be kept secret.

Party U and V are allowed to select the connection identifiers C_U and C_V, respectively, for the other party to use in the ongoing EDHOC protocol as well as in a subsequent application protocol (e.g. OSCORE [RFC8613]). The choice of connection identifier is not security critical in EDHOC but intended to simplify the retrieval of the right security context in combination with using short identifiers. If the wrong connection identifier of the other party is used in a protocol message it will result in the receiving party not being able to retrieve a security context (which will terminate the protocol) or retrieve the wrong security context (which also terminates the protocol as the message cannot be verified).

Party V MUST finish the verification step of message_3 before passing PAD_3 to the application.

If two nodes unintentionally initiate two simultaneous EDHOC message exchanges with each other even if they only want to complete a single EDHOC message exchange, they MAY terminate the exchange with the lexicographically smallest G_X. If the two G_X values are equal, the received message_1 MUST be discarded to mitigate reflection attacks. Note that in the case of two simultaneous EDHOC exchanges where the nodes only complete one and where the nodes have different preferred

cipher suites, an attacker can affect which of the two nodes' preferred cipher suites will be used by blocking the other exchange.

8.7. Other Documents Referencing EDHOC

EDHOC has been analyzed in several other documents. A formal verification of EDHOC was done in [SSR18], an analysis of EDHOC for certificate enrollment was done in [Kron18], the use of EDHOC in LoRaWAN is analyzed in [LoRa1] and [LoRa2], the use of EDHOC in IoT bootstrapping is analyzed in [Perez18], and the use of EDHOC in 6TiSCH is described in [I-D.ietf-6tisch-dtsecurity-zerotouch-join].

9. IANA Considerations

9.1. EDHOC Cipher Suites Registry

IANA has created a new registry titled "EDHOC Cipher Suites" under the new heading "EDHOC". The registration procedure is "Expert Review". The columns of the registry are Value, Array, Description, and Reference, where Value is an integer and the other columns are text strings. The initial contents of the registry are:

Value: 1
Array: [10, 5, 1, -7, 1]
Desc: AES-CCM-16-64-128, HMAC 256/256, P-256, ES256, P-256
Reference: [[this document]]

Value: 0
Array: [10, 5, 4, -8, 6]
Desc: AES-CCM-16-64-128, HMAC 256/256, X25519, EdDSA, Ed25519
Reference: [[this document]]

Value: -5
Array:
Desc: Reserved for Private Use
Reference: [[this document]]

Value: -6
Array:
Desc: Reserved for Private Use
Reference: [[this document]]

9.2. EDHOC Method Type Registry

IANA has created a new registry titled "EDHOC Method Type" under the new heading "EDHOC". The registration procedure is "Expert Review". The columns of the registry are Value, Description, and Reference,

where Value is an integer and the other columns are text strings. The initial contents of the registry are:

Value	Specification	Reference
0	EDHOC Authenticated with Asymmetric Keys	[[this document]]
1	EDHOC Authenticated with Symmetric Keys	[[this document]]

9.3. The Well-Known URI Registry

IANA has added the well-known URI 'edhoc' to the Well-Known URIs registry.

- URI suffix: edhoc
- Change controller: IETF
- Specification document(s): [[this document]]
- Related information: None

9.4. Media Types Registry

IANA has added the media type 'application/edhoc' to the Media Types registry.

- Type name: application
- Subtype name: edhoc
- Required parameters: N/A
- Optional parameters: N/A
- Encoding considerations: binary
- Security considerations: See Section 7 of this document.
- Interoperability considerations: N/A
- Published specification: [[this document]] (this document)
- Applications that use this media type: To be identified
- Fragment identifier considerations: N/A

- Additional information:
  - Magic number(s): N/A
  - File extension(s): N/A
  - Macintosh file type code(s): N/A
- Person & email address to contact for further information: See "Authors' Addresses" section.
- Intended usage: COMMON
- Restrictions on usage: N/A
- Author: See "Authors' Addresses" section.
- Change Controller: IESG

9.5. CoAP Content-Formats Registry

IANA has added the media type 'application/edhoc' to the CoAP Content-Formats registry.

- Media Type: application/edhoc
- Encoding:
- ID: TBD42
- Reference: [[this document]]

9.6. Expert Review Instructions

The IANA Registries established in this document is defined as "Expert Review". This section gives some general guidelines for what the experts should be looking for, but they are being designated as experts for a reason so they should be given substantial latitude.

Expert reviewers should take into consideration the following points:

- Clarity and correctness of registrations. Experts are expected to check the clarity of purpose and use of the requested entries. Expert needs to make sure the values of algorithms are taken from the right registry, when that's required. Expert should consider requesting an opinion on the correctness of registered parameters from relevant IETF working groups. Encodings that do not meet

these objective of clarity and completeness should not be registered.

- Experts should take into account the expected usage of fields when approving point assignment. The length of the encoded value should be weighed against how many code points of that length are left, the size of device it will be used on, and the number of code points left that encode to that size.

- Specifications are recommended. When specifications are not provided, the description provided needs to have sufficient information to verify the points above.

10. References

10.1. Normative References

[I-D.ietf-cbor-7049bis]
Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", draft-ietf-cbor-7049bis-07 (work in progress), August 2019.

[I-D.ietf-cbor-sequence]
Bormann, C., "Concise Binary Object Representation (CBOR) Sequences", draft-ietf-cbor-sequence-01 (work in progress), August 2019.

[I-D.ietf-core-echo-request-tag]
Amsuess, C., Mattsson, J., and G. Selander, "CoAP: Echo, Request-Tag, and Token Processing", draft-ietf-core-echo-request-tag-05 (work in progress), May 2019.

[I-D.ietf-cose-x509]
Schaad, J., "CBOR Object Signing and Encryption (COSE): Headers for carrying and referencing X.509 certificates", draft-ietf-cose-x509-03 (work in progress), August 2019.

[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>.

[RFC5116] McGrew, D., "An Interface and Algorithms for Authenticated Encryption", RFC 5116, DOI 10.17487/RFC5116, January 2008, <https://www.rfc-editor.org/info/rfc5116>.

[RFC5869] Krawczyk, H. and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)", RFC 5869, DOI 10.17487/RFC5869, May 2010, <https://www.rfc-editor.org/info/rfc5869>.

[RFC6090] McGrew, D., Igoe, K., and M. Salter, "Fundamental Elliptic Curve Cryptography Algorithms", RFC 6090, DOI 10.17487/RFC6090, February 2011, <https://www.rfc-editor.org/info/rfc6090>.

[RFC6979] Pornin, T., "Deterministic Usage of the Digital Signature Algorithm (DSA) and Elliptic Curve Digital Signature Algorithm (ECDSA)", RFC 6979, DOI 10.17487/RFC6979, August 2013, <https://www.rfc-editor.org/info/rfc6979>.

[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>.

[RFC7959] Bormann, C. and Z. Shelby, Ed., "Block-Wise Transfers in the Constrained Application Protocol (CoAP)", RFC 7959, DOI 10.17487/RFC7959, August 2016, <https://www.rfc-editor.org/info/rfc7959>.

[RFC8152] Schaad, J., "CBOR Object Signing and Encryption (COSE)", RFC 8152, DOI 10.17487/RFC8152, July 2017, <https://www.rfc-editor.org/info/rfc8152>.

[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>.

[RFC8610] Birkholz, H., Vigano, C., and C. Bormann, "Concise Data Definition Language (CDDL): A Notational Convention to Express Concise Binary Object Representation (CBOR) and JSON Data Structures", RFC 8610, DOI 10.17487/RFC8610, June 2019, <https://www.rfc-editor.org/info/rfc8610>.

[RFC8613] Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security for Constrained RESTful Environments (OSCORE)", RFC 8613, DOI 10.17487/RFC8613, July 2019, <https://www.rfc-editor.org/info/rfc8613>.

[SIGMA] Krawczyk, H., "SIGMA - The 'SIGn-and-MAc' Approach to Authenticated Diffie-Hellman and Its Use in the IKE-Protocols (Long version)", June 2003, <http://webee.technion.ac.il/~hugo/sigma-pdf.pdf>.

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

10.2. Informative References

[CborMe] Bormann, C., "CBOR Playground", May 2018, <http://cbor.me/>.

[I-D.hartke-core-e2e-security-reqs] Selander, G., Palombini, F., and K. Hartke, "Requirements for CoAP End-To-End Security", draft-hartke-core-e2e-security-reqs-03 (work in progress), July 2017.

[I-D.ietf-6tisch-dtsecurity-zerotouch-join] Richardson, M., "6tisch Zero-Touch Secure Join protocol", draft-ietf-6tisch-dtsecurity-zerotouch-join-04 (work in progress), July 2019.

[I-D.ietf-ace-oauth-authz] Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "Authentication and Authorization for Constrained Environments (ACE) using the OAuth 2.0 Framework (ACE-OAuth)", draft-ietf-ace-oauth-authz-24 (work in progress), March 2019.

[I-D.ietf-ace-oscore-profile] Palombini, F., Seitz, L., Selander, G., and M. Gunnarsson, "OSCORE profile of the Authentication and Authorization for Constrained Environments Framework", draft-ietf-ace-oscore-profile-08 (work in progress), July 2019.

[I-D.ietf-core-resource-directory] Shelby, Z., Koster, M., Bormann, C., Stok, P., and C. Amsuess, "CoRE Resource Directory", draft-ietf-core-resource-directory-23 (work in progress), July 2019.

[I-D.ietf-lwig-security-protocol-comparison]
Mattsson, J. and F. Palombini, "Comparison of CoAP Security Protocols", draft-ietf-lwig-security-protocol-comparison-03 (work in progress), March 2019.

[I-D.ietf-tls-dtls13]
Rescorla, E., Tschofenig, H., and N. Modadugu, "The Datagram Transport Layer Security (DTLS) Protocol Version 1.3", draft-ietf-tls-dtls13-32 (work in progress), July 2019.

[Kron18] Krontiris, A., "Evaluation of Certificate Enrollment over Application Layer Security", May 2018, <https://www.nada.kth.se/~ann/exjobb/alexandros_krontiris.pdf>.

[LoRa1] Sanchez-Iborra, R., Sanchez-Gomez, J., Perez, S., Fernandez, P., Santa, J., Hernandez-Ramos, J., and A. Skarmeta, "Enhancing LoRaWAN Security through a Lightweight and Authenticated Key Management Approach", June 2018, <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6021899/pdf/sensors-18-01833.pdf>.

[LoRa2] Sanchez-Iborra, R., Sanchez-Gomez, J., Perez, S., Fernandez, P., Santa, J., Hernandez-Ramos, J., and A. Skarmeta, "Internet Access for LoRaWAN Devices Considering Security Issues", June 2018, <https://ants.inf.um.es/~josesanta/doc/GIoTS1.pdf>.

[OPTLS] Krawczyk, H. and H. Wee, "The OPTLS Protocol and TLS 1.3", October 2015, <https://eprint.iacr.org/2015/978.pdf>.

[Perez18] Perez, S., Garcia-Carrillo, D., Marin-Lopez, R., Hernandez-Ramos, J., Marin-Perez, R., and A. Skarmeta, "Architecture of security association establishment based on bootstrapping technologies for enabling critical IoT infrastructures", October 2018, <http://www.anastacia-h2020.eu/publications/Architecture_of_security_association_establishment_based_on_bootstrapping_technologies_for_enabling_critical_IoT_infrastructures.pdf>.

[RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, DOI 10.17487/RFC7228, May 2014, <https://www.rfc-editor.org/info/rfc7228>.

[RFC7258] Farrell, S. and H. Tschofenig, "Pervasive Monitoring Is an Attack", BCP 188, RFC 7258, DOI 10.17487/RFC7258, May 2014, <https://www.rfc-editor.org/info/rfc7258>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

[SSR18] Bruni, A., Sahl Joergensen, T., Groenbech Petersen, T., and C. Schuermann, "Formal Verification of Ephemeral Diffie-Hellman Over COSE (EDHOC)", November 2018, <https://www.springerprofessional.de/en/formal-verification-of-ephemeral-diffie-hellman-over-cose-edhoc/16284348>.

Appendix A. Use of CBOR, CDDL and COSE in EDHOC

This Appendix is intended to simplify for implementors not familiar with CBOR [I-D.ietf-cbor-7049bis], CDDL [RFC8610], COSE [RFC8152], and HKDF [RFC5869].

A.1. CBOR and CDDL

The Concise Binary Object Representation (CBOR) [I-D.ietf-cbor-7049bis] is a data format designed for small code size and small message size. CBOR builds on the JSON data model but extends it by e.g. encoding binary data directly without base64 conversion. In addition to the binary CBOR encoding, CBOR also has a diagnostic notation that is readable and editable by humans. The Concise Data Definition Language (CDDL) [RFC8610] provides a way to express structures for protocol messages and APIs that use CBOR. [RFC8610] also extends the diagnostic notation.

CBOR data items are encoded to or decoded from byte strings using a type-length-value encoding scheme, where the three highest order bits of the initial byte contain information about the major type. CBOR supports several different types of data items, in addition to integers (int, uint), simple values (e.g. null), byte strings (bstr), and text strings (tstr), CBOR also supports arrays [] of data items, maps {} of pairs of data items, and sequences [I-D.ietf-cbor-sequence] of data items. Some examples are given below. For a complete specification and more examples, see [I-D.ietf-cbor-7049bis] and [RFC8610]. We recommend implementors to get used to CBOR by using the CBOR playground [CborMe].

Diagnostic	Encoded	Type
1	0x01	unsigned integer
24	0x1818	unsigned integer
-24	0x37	negative integer
-25	0x3818	negative integer
null	0xf6	simple value
h'12cd'	0x4212cd	byte string
'12cd'	0x4431326364	byte string
"12cd"	0x6431326364	text string
{ 4 : h'cd' }	0xa10441cd	map
<< 1, 2, null >>	0x430102f6	byte string
[1, 2, null]	0x830102f6	array
(1, 2, null)	0x0102f6	sequence
1, 2, null	0x0102f6	sequence

EDHOC messages are CBOR Sequences [I-D.ietf-cbor-sequence]. The message format specification uses the construct '.cbor' enabling conversion between different CDDL types matching different CBOR items with different encodings. Some examples are given below.

A type (e.g. an uint) may be wrapped in a byte string (bstr):

CDDL Type	Diagnostic	Encoded
uint	24	0x1818
bstr .cbor uint	<< 24 >>	0x421818

A.2. COSE

CBOR Object Signing and Encryption (COSE) [RFC8152] describes how to create and process signatures, message authentication codes, and encryption using CBOR. COSE builds on JOSE, but is adapted to allow more efficient processing in constrained devices. EDHOC makes use of COSE_Key, COSE_Encrypt0, COSE_Sign1, and COSE_KDF_Context objects.

Appendix B. EDHOC Authenticated withDiffie-Hellman Keys

The SIGMA protocol is mainly optimized for PKI and certificates. The OPTLS protocol [OPTLS] shows how authentication can be provided by a MAC computed from an ephemeral-static ECDH shared secret. Instead of signature authentication keys, U and V would have Diffie-Hellman authentication keys G_U and G_V, respectively. This type of authentication keys could easily be used with RPK and would provide significant reductions in message sizes as the 64 bytes signature would be replaced by an 8 bytes MAC.

EDHOC authenticated with asymmetric Diffie-Hellman keys should have similar security properties as EDHOC authenticated with asymmetric signature keys with a few differences:

- Repudiation: In EDHOC authenticated with asymmetric signature keys, Party U could theoretically prove that Party V performed a run of the protocol by presenting the private ephemeral key, and vice versa. Note that storing the private ephemeral keys violates the protocol requirements. With asymmetric Diffie-Hellman key authentication, both parties can always deny having participated in the protocol, this is similar to EDHOC with symmetric key authentication.

- Key compromise impersonation (KCI): In EDHOC authenticated with asymmetric signature keys, EDHOC provides KCI protection against an attacker having access to the long term key or the ephemeral secret key. In EDHOC authenticated with symmetric keys, EDHOC provides KCI protection against an attacker having access to the ephemeral secret key, but not against an attacker having access to the long-term PSK. With asymmetric Diffie-Hellman key authentication, KCI protection would be provided against an attacker having access to the long-term Diffie-Hellman key, but not to an attacker having access to the ephemeral secret key. Note that the term KCI has typically been used for compromise of long-term keys, and that an attacker with access to the ephemeral secret key can only attack that specific protocol run.

TODO: Initial suggestion for key derivation, message formats, and processing

Appendix C. Test Vectors

This appendix provides detailed test vectors to ease implementation and ensure interoperability. In addition to hexadecimal, all CBOR data items and sequences are given in CBOR diagnostic notation. The test vectors use 1 byte key identifiers, 1 byte connection IDs, and the default mapping to CoAP where Party U is CoAP client (this means that corr = 1).

C.1. Test Vectors for EDHOC Authenticated with Asymmetric Keys (RPK)

Asymmetric EDHOC is used:

method (Asymmetric Authentication)
0

CoAP is used as transport:

corr (Party U is CoAP client)
1

No unprotected opaque application data is sent in the message exchanges.

The pre-defined Cipher Suite 0 is in place both on Party U and Party V, see Section 3.1.

C.1.1. Input for Party U

The following are the parameters that are set in Party U before the first message exchange.

Party U's private authentication key (32 bytes)
53 21 fc 01 c2 98 20 06 3a 72 50 8f c6 39 25 1d c8 30 e2 f7 68 3e b8 e3 8a f1 64 a5 b9 af 9b e3

Party U's public authentication key (32 bytes)
42 4c 75 6a b7 7c c6 fd ec f0 b3 ec fc ff b7 53 10 c0 15 bf 5c ba 2e c0 a2 36 e6 65 0c 8a b9 c7

kid value to identify U's public authentication key (1 bytes)
a2

This test vector uses COSE_Key objects to store the raw public keys. Moreover, EC2 keys with curve Ed25519 are used. That is in agreement with the Cipher Suite 0.

```
CRED_U =
<< {
  1:  1,
 -1:  6,
 -2:  h'424c756ab77cc6fdecf0b3ecfcffb75310c015bf5cba2ec0a236e6650c8ab9c7'
} >>
```

CRED_U (COSE_Key) (CBOR-encoded) (42 bytes)
58 28 a3 01 01 20 06 21 58 20 42 4c 75 6a b7 7c c6 fd ec f0 b3 ec fc ff b7 53 10 c0 15 bf 5c ba 2e c0 a2 36 e6 65 0c 8a b9 c7

Because COSE_Keys are used, and because kid = h'a2':

```
ID_CRED_U =
{
  4:  h'a2'
}
```

Note that since the map for ID_CRED_U contains a single 'kid' parameter, ID_CRED_U is used when transported in the protected header of the COSE Object, but only the kid_value is used when added to the plaintext (see Section 4.4.2):

ID_CRED_U (in protected header) (CBOR-encoded) (4 bytes)
a1 04 41 a2

kid_value (in plaintext) (CBOR-encoded) (2 bytes)
41 a2

C.1.2. Input for Party V

The following are the parameters that are set in Party V before the first message exchange.

Party V's private authentication key (32 bytes)
74 56 b3 a3 e5 8d 8d 26 dd 36 bc 75 d5 5b 88 63 a8 5d 34 72 f4 a0 1f 02 24 62 1b 1c b8 16 6d a9

Party V's public authentication key (32 bytes)
1b 66 1e e5 d5 ef 16 72 a2 d8 77 cd 5b c2 0f 46 30 dc 78 a1 14 de 65 9c 7e 50 4d 0f 52 9a 6b d3

kid value to identify U's public authentication key (1 bytes)
a3

This test vector uses COSE_Key objects to store the raw public keys. Moreover, EC2 keys with curve Ed25519 are used. That is in agreement with the Cipher Suite 0.

```
CRED_V =
<< {
  1:  1,
 -1:  6,
 -2:  h'1b661ee5d5ef1672a2d877cd5bc20f4630dc78a114de659c7e504d0f529a6bd3'
} >>
```

CRED_V (COSE_Key) (CBOR-encoded) (42 bytes)
58 28 a3 01 01 20 06 21 58 20 1b 66 1e e5 d5 ef 16 72 a2 d8 77 cd 5b c2 0f 46 30 dc 78 a1 14 de 65 9c 7e 50 4d 0f 52 9a 6b d3

Because COSE_Keys are used, and because kid = h'a3':

```
ID_CRED_V =
{
  4:  h'a3'
}
```

Note that since the map for ID_CRED_U contains a single 'kid' parameter, ID_CRED_U is used when transported in the protected header of the COSE Object, but only the kid_value is used when added to the plaintext (see Section 4.4.2):

ID_CRED_V (in protected header) (CBOR-encoded) (4 bytes)
a1 04 41 a3

kid_value (in plaintext) (CBOR-encoded) (2 bytes)
41 a3

C.1.3. Message 1

From the input parameters (in Appendix C.1.1):

TYPE (4 * method + corr)
1

suite
0

SUITES_U : suite
0

G_X (X-coordinate of the ephemeral public key of Party U) (32 bytes)
b1 a3 e8 94 60 e8 8d 3a 8d 54 21 1d c9 5f 0b 90 3f f2 05 eb 71 91 2d 6d b8 f4 af 98 0d 2d b8 3a

C_U (Connection identifier chosen by U) (1 bytes)
c3

No UAD_1 is provided, so UAD_1 is absent from message_1.

Message_1 is constructed, as the CBOR Sequence of the CBOR data items above.

```
message_1 =
(
  1,
  0,
  h'b1a3e89460e88d3a8d54211dc95f0b903ff205eb71912d6db8f4af980d2db83a',
  h'c3'
)
```

message_1 (CBOR Sequence) (38 bytes)
01 00 58 20 b1 a3 e8 94 60 e8 8d 3a 8d 54 21 1d c9 5f 0b 90 3f f2 05 eb 71 91 2d 6d b8 f4 af 98 0d 2d b8 3a 41 c3

C.1.4. Message 2

Since TYPE mod 4 equals 1, C_U is omitted from data_2.

G_Y (X-coordinate of the ephemeral public key of Party V) (32 bytes)
8d b5 77 f9 b9 c2 74 47 98 98 7d b5 57 bf 31 ca 48 ac d2 05 a9 db 8c 32 0e
5d 49 f3 02 a9 64 74

C_V (Connection identifier chosen by V) (1 bytes)
c4

Data_2 is constructed, as the CBOR Sequence of the CBOR data items above.

```
data_2 =
(
  h'8db577f9b9c2744798987db557bf31ca48acd205a9db8c320e5d49f302a96474',
  h'c4'
)
```

data_2 (CBOR Sequence) (36 bytes)
58 20 8d b5 77 f9 b9 c2 74 47 98 98 7d b5 57 bf 31 ca 48 ac d2 05 a9 db 8c
32 0e 5d 49 f3 02 a9 64 74 41 c4

From data_2 and message_1 (from Appendix C.1.3), compute the input to the transcript hash TH_2 = H(message_1, data_2), as a CBOR Sequence of these 2 data items.

(message_1, data_2) (CBOR Sequence)
(74 bytes)
01 00 58 20 b1 a3 e8 94 60 e8 8d 3a 8d 54 21 1d c9 5f 0b 90 3f f2 05 eb 71
91 2d 6d b8 f4 af 98 0d 2d b8 3a 41 c3 58 20 8d b5 77 f9 b9 c2 74 47 98 98
7d b5 57 bf 31 ca 48 ac d2 05 a9 db 8c 32 0e 5d 49 f3 02 a9 64 74 41 c4

And from there, compute the transcript hash TH_2 = SHA-256(message_1, data_2)

TH_2 value (32 bytes)
55 50 b3 dc 59 84 b0 20 9a e7 4e a2 6a 18 91 89 57 50 8e 30 33 2b 11 da 68
1d c2 af dd 87 03 55

When encoded as a CBOR bstr, that gives:

TH_2 (CBOR-encoded) (34 bytes)
58 20 55 50 b3 dc 59 84 b0 20 9a e7 4e a2 6a 18 91 89 57 50 8e 30 33 2b 11
da 68 1d c2 af dd 87 03 55

C.1.4.1. Signature Computation

COSE_Sign1 is computed with the following parameters. From Appendix C.1.2:

- protected = bstr .cbor ID_CRED_V
- payload = CRED_V

And from Appendix C.1.4:

- external_aad = TH_2

The Sig_structure M_V to be signed is: ["Signature1", << ID_CRED_V >>, TH_2, CRED_V] , as defined in Section 4.3.2:

```
M_V =
[
  "Signature1",
  << { 4: h'a3' } >>,
  h'5550b3dc5984b0209ae74ea26a18918957508e30332b11da681dc2afdd870355',
  << {
    1:  1,
   -1:  6,
   -2:  h'1b661ee5d5ef1672a2d877cd5bc20f4630dc78a114de659c7e504d0f529a6b
          d3'
  } >>
]
```

Which encodes to the following byte string ToBeSigned:

```
M_V (message to be signed with Ed25519) (CBOR-encoded) (93 bytes)
84 6a 53 69 67 6e 61 74 75 72 65 31 44 a1 04 41 a3 58 20 55 50 b3 dc 59 84
b0 20 9a e7 4e a2 6a 18 91 89 57 50 8e 30 33 2b 11 da 68 1d c2 af dd 87 03
55 58 28 a3 01 01 20 06 21 58 20 1b 66 1e e5 d5 ef 16 72 a2 d8 77 cd 5b c2
0f 46 30 dc 78 a1 14 de 65 9c 7e 50 4d 0f 52 9a 6b d3
```

The message is signed using the private authentication key of V, and produces the following signature:

```
V's signature (64 bytes)
52 3d 99 6d fd 9e 2f 77 c7 68 71 8a 30 c3 48 77 8c 5e b8 64 dd 53 7e 55 5e
4a 00 05 e2 09 53 07 13 ca 14 62 0d e8 18 7e 81 99 6e e8 04 d1 53 b8 a1 f6
08 49 6f dc d9 3d 30 fc 1c 8b 45 be cc 06
```

C.1.4.2. Key and Nonce Computation

The key and nonce for calculating the ciphertext are calculated as follows, as specified in Section 3.3.

HKDF SHA-256 is the HKDF used (as defined by cipher suite 0).

PRK = HMAC-SHA-256(salt, G_XY)

Since this is the asymmetric case, salt is the empty byte string.

G_XY is the shared secret, and since the curve25519 is used, the ECDH shared secret is the output of the X25519 function.

```
G_XY (32 bytes)
c6 1e 09 09 a1 9d 64 24 01 63 ec 26 2e 9c c4 f8 8c e7 7b e1 23 c5 ab 53 8d
26 b0 69 22 a5 20 67
```

From there, PRK is computed:

```
PRK (32 bytes)
ba 9c 2c a1 c5 62 14 a6 e0 f6 13 ed a8 91 86 8a 4c a3 e3 fa bc c7 79 8f dc
01 60 80 07 59 16 71
```

Key K_2 is the output of HKDF-Expand(PRK, info, L).

info is defined as follows:

```
info for K_2
[
  10,
  [ null, null, null ],
  [ null, null, null ],
  [ 128, h'', h'5550b3dc5984b0209ae74ea26a18918957508e30332b11da681dc2afdd
                870355' ]
]
```

Which as a CBOR encoded data item is:

```
info (K_2) (CBOR-encoded) (48 bytes)
84 0a 83 f6 f6 f6 83 f6 f6 f6 83 18 80 40 58 20 55 50 b3 dc 59 84 b0 20 9a
e7 4e a2 6a 18 91 89 57 50 8e 30 33 2b 11 da 68 1d c2 af dd 87 03 55
```

L is the length of K_2, so 16 bytes.

From these parameters, K_2 is computed:

K_2 (16 bytes)
da d7 44 af 07 c4 da 27 d1 f0 a3 8a 0c 4b 87 38

Nonce IV_2 is the output of HKDF-Expand(PRK, info, L).

info is defined as follows:

```
info for IV_2
[
  "IV-GENERATION",
  [ null, null, null ],
  [ null, null, null ],
  [ 104, h'', h'5550b3dc5984b0209ae74ea26a18918957508e30332b11da681dc2afdd
                870355' ]
]
```

Which as a CBOR encoded data item is:

```
info (IV_2) (CBOR-encoded) (61 bytes)
84 6d 49 56 2d 47 45 4e 45 52 41 54 49 4f 4e 83 f6 f6 f6 83 f6 f6 f6 83 18
68 40 58 20 55 50 b3 dc 59 84 b0 20 9a e7 4e a2 6a 18 91 89 57 50 8e 30 33
2b 11 da 68 1d c2 af dd 87 03 55
```

L is the length of IV_2, so 13 bytes.

From these parameters, IV_2 is computed:

IV_2 (13 bytes)
fb a1 65 d9 08 da a7 8e 4f 84 41 42 d0

C.1.4.3. Ciphertext Computation

COSE_Encrypt0 is computed with the following parameters. Note that UAD_2 is omitted.

- empty protected header
- external_aad = TH_2
- plaintext = CBOR Sequence of the items kid_value, signature, in this order.

with kid_value taken from Appendix C.1.2, and signature as calculated in Appendix C.1.4.1.

The plaintext is the following:

```
P_2 (68 bytes)
41 a3 58 40 52 3d 99 6d fd 9e 2f 77 c7 68 71 8a 30 c3 48 77 8c 5e b8 64 dd
53 7e 55 5e 4a 00 05 e2 09 53 07 13 ca 14 62 0d e8 18 7e 81 99 6e e8 04 d1
53 b8 a1 f6 08 49 6f dc d9 3d 30 fc 1c 8b 45 be cc 06
```

From the parameters above, the Enc_structure A_2 is computed.

```
A_2 =
[
  "Encrypt0",
  h'',
  h'5550b3dc5984b0209ae74ea26a18918957508e30332b11da681dc2afdd870355'
]
```

Which encodes to the following byte string to be used as Additional Authenticated Data:

```
A_2 (CBOR-encoded) (45 bytes)
83 68 45 6e 63 72 79 70 74 30 40 58 20 55 50 b3 dc 59 84 b0 20 9a e7 4e a2
6a 18 91 89 57 50 8e 30 33 2b 11 da 68 1d c2 af dd 87 03 55
```

The key and nonce used are defined in Appendix C.1.4.2:

- key = K_2

- nonce = IV_2

Using the parameters above, the ciphertext CIPHERTEXT_2 can be computed:

```
CIPHERTEXT_2 (76 bytes)
1e 6b fe 0e 77 99 ce f0 66 a3 4f 08 ef aa 90 00 6d b4 4c 90 1c f7 9b 23 85
3a b9 7f d8 db c8 53 39 d5 ed 80 87 78 3c f7 a4 a7 e0 ea 38 c2 21 78 9f a3
71 be 64 e9 3c 43 a7 db 47 d1 e3 fb 14 78 8e 96 7f dd 78 d8 80 78 e4 9b 78
bf
```

C.1.4.4. message_2

From the parameter computed in Appendix C.1.4 and Appendix C.1.4.3, message_2 is computed, as the CBOR Sequence of the following items: (G_Y, C_V, CIPHERTEXT_2).

```
message_2 =
(
  h'8db577f9b9c2744798987db557bf31ca48acd205a9db8c320e5d49f302a96474',
  h'c4',
  h'1e6bfe0e7799cef066a34f08efaa90006db44c901cf79b23853ab97fd8dbc85339d5ed
  8087783cf7a4a7e0ea38c221789fa371be64e93c43a7db47d1e3fb14788e967fdd78d880
  78e49b78bf'
)
```

Which encodes to the following byte string:

```
message_2 (CBOR Sequence) (114 bytes)
58 20 8d b5 77 f9 b9 c2 74 47 98 98 7d b5 57 bf 31 ca 48 ac d2 05 a9 db 8c
32 0e 5d 49 f3 02 a9 64 74 41 c4 58 4c 1e 6b fe 0e 77 99 ce f0 66 a3 4f 08
ef aa 90 00 6d b4 4c 90 1c f7 9b 23 85 3a b9 7f d8 db c8 53 39 d5 ed 80 87
78 3c f7 a4 a7 e0 ea 38 c2 21 78 9f a3 71 be 64 e9 3c 43 a7 db 47 d1 e3 fb
14 78 8e 96 7f dd 78 d8 80 78 e4 9b 78 bf
```

C.1.5. Message 3

Since TYPE mod 4 equals 1, C_V is not omitted from data_3.

```
C_V (1 bytes)
c4
```

Data_3 is constructed, as the CBOR Sequence of the CBOR data item above.

```
data_3 =
(
  h'c4'
)

data_3 (CBOR Sequence) (2 bytes)
41 c4
```

From data_3, CIPHERTEXT_2 (Appendix C.1.4.3), and TH_2 (Appendix C.1.4), compute the input to the transcript hash TH_2 = H(TH_2 , CIPHERTEXT_2, data_3), as a CBOR Sequence of these 3 data items.

```
( TH_2, CIPHERTEXT_2, data_3 )
(CBOR Sequence) (114 bytes)
58 20 55 50 b3 dc 59 84 b0 20 9a e7 4e a2 6a 18 91 89 57 50 8e 30 33 2b 11
da 68 1d c2 af dd 87 03 55 58 4c 1e 6b fe 0e 77 99 ce f0 66 a3 4f 08 ef aa
90 00 6d b4 4c 90 1c f7 9b 23 85 3a b9 7f d8 db c8 53 39 d5 ed 80 87 78 3c
f7 a4 a7 e0 ea 38 c2 21 78 9f a3 71 be 64 e9 3c 43 a7 db 47 d1 e3 fb 14 78
8e 96 7f dd 78 d8 80 78 e4 9b 78 bf 41 c4
```

And from there, compute the transcript hash TH_3 = SHA-256(TH_2 , CIPHERTEXT_2, data_3)

```
TH_3 value (32 bytes)
21 cc b6 78 b7 91 14 96 09 55 88 5b 90 a2 b8 2e 3b 2c a2 7e 8e 37 4a 79 07
f3 e7 85 43 67 fc 22
```

When encoded as a CBOR bstr, that gives:

```
TH_3 (CBOR-encoded) (34 bytes)
58 20 21 cc b6 78 b7 91 14 96 09 55 88 5b 90 a2 b8 2e 3b 2c a2 7e 8e 37 4a
79 07 f3 e7 85 43 67 fc 22
```

C.1.5.1. Signature Computation

COSE_Sign1 is computed with the following parameters. From Appendix C.1.2:

- protected = bstr .cbor ID_CRED_U
- payload = CRED_U

And from Appendix C.1.4:

- external_aad = TH_3

The Sig_structure M_V to be signed is: ["Signature1", << ID_CRED_U >>, TH_3, CRED_U] , as defined in Section 4.4.2:

```
M_U =
[
  "Signature1",
  << { 4: h'a2' } >>,
  h'734bef323d867a12956127c2e62ade42c0f119e5487750c0c31fd093376dceed',
  << {
    1:  1,
   -1:  6,
   -2:  h'424c756ab77cc6fdecf0b3ecfcffb75310c015bf5cba2ec0a236e6650c8ab9
   c7'
  } >>
]
```

Which encodes to the following byte string ToBeSigned:

M_U (message to be signed with Ed25519) (CBOR-encoded) (93 bytes)
84 6a 53 69 67 6e 61 74 75 72 65 31 44 a1 04 41 a2 58 20 73 4b ef 32 3d 86
7a 12 95 61 27 c2 e6 2a de 42 c0 f1 19 e5 48 77 50 c0 c3 1f d0 93 37 6d ce
ed 58 28 a3 01 01 20 06 21 58 20 42 4c 75 6a b7 7c c6 fd ec f0 b3 ec fc ff
b7 53 10 c0 15 bf 5c ba 2e c0 a2 36 e6 65 0c 8a b9 c7

The message is signed using the private authentication key of U, and produces the following signature:

U's signature (64 bytes)
5c 7d 7d 64 c9 61 c5 f5 2d cf 33 91 25 92 a1 af f0 2c 33 62 b0 e7 55 0e 4b
c5 66 b7 0c 20 61 f3 c5 f6 49 e5 ed 32 3d 30 a2 6c 61 2f bb 5c bd 25 f3 1c
27 22 8c ea ec 64 29 31 95 41 fe 07 8e 0e

C.1.5.2. Key and Nonce Computation

The key and nonce for calculating the ciphertext are calculated as follows, as specified in Section 3.3.

HKDF SHA-256 is the HKDF used (as defined by cipher suite 0).

PRK = HMAC-SHA-256(salt, G_XY)

Since this is the asymmetric case, salt is the empty byte string.

G_XY is the shared secret, and since the curve25519 is used, the ECDH shared secret is the output of the X25519 function.

G_XY (32 bytes)
c6 1e 09 09 a1 9d 64 24 01 63 ec 26 2e 9c c4 f8 8c e7 7b e1 23 c5 ab 53 8d
26 b0 69 22 a5 20 67

From there, PRK is computed:

PRK (32 bytes)
ba 9c 2c a1 c5 62 14 a6 e0 f6 13 ed a8 91 86 8a 4c a3 e3 fa bc c7 79 8f dc
01 60 80 07 59 16 71

Key K_3 is the output of HKDF-Expand(PRK, info, L).

info is defined as follows:

```
info for K_3
[
  10,
  [ null, null, null ],
  [ null, null, null ],
  [ 128, h'', h'21ccb678b79114960955885b90a2b82e3b2ca27e8e374a7907f3e78543
  67fc22' ]
]
```

Which as a CBOR encoded data item is:

```
info (K_3) (CBOR-encoded) (48 bytes)
84 0a 83 f6 f6 f6 83 f6 f6 f6 83 18 80 40 58 20 21 cc b6 78 b7 91 14 96 09
55 88 5b 90 a2 b8 2e 3b 2c a2 7e 8e 37 4a 79 07 f3 e7 85 43 67 fc 22
```

L is the length of K_3, so 16 bytes.

From these parameters, K_3 is computed:

```
K_3 (16 bytes)
e1 ac d4 76 f5 96 a4 60 72 44 a8 da 8c ff 49 df
```

Nonce IV_3 is the output of HKDF-Expand(PRK, info, L).

info is defined as follows:

```
info for IV_3
[
  "IV-GENERATION",
  [ null, null, null ],
  [ null, null, null ],
  [ 104, h'', h'21ccb678b79114960955885b90a2b82e3b2ca27e8e374a7907f3e78543
  67fc22' ]
]
```

Which as a CBOR encoded data item is:

```
info (IV_3) (CBOR-encoded) (61 bytes)
84 6d 49 56 2d 47 45 4e 45 52 41 54 49 4f 4e 83 f6 f6 f6 83 f6 f6 f6 83 18
68 40 58 20 21 cc b6 78 b7 91 14 96 09 55 88 5b 90 a2 b8 2e 3b 2c a2 7e 8e
37 4a 79 07 f3 e7 85 43 67 fc 22
```

L is the length of IV_3, so 13 bytes.

From these parameters, IV_3 is computed:

```
IV_3 (13 bytes)
de 53 02 13 ab a2 6a 47 1a 51 f3 d6 fb
```

C.1.5.3. Ciphertext Computation

COSE_Encrypt0 is computed with the following parameters. Note that PAD_3 is omitted.

- empty protected header
- external_aad = TH_3
- plaintext = CBOR Sequence of the items kid_value, signature, in this order.

with kid_value taken from Appendix C.1.1, and signature as calculated in Appendix C.1.5.1.

The plaintext is the following:

```
P_3 (68 bytes)
41 a2 58 40 5c 7d 7d 64 c9 61 c5 f5 2d cf 33 91 25 92 a1 af f0 2c 33 62 b0
e7 55 0e 4b c5 66 b7 0c 20 61 f3 c5 f6 49 e5 ed 32 3d 30 a2 6c 61 2f bb 5c
bd 25 f3 1c 27 22 8c ea ec 64 29 31 95 41 fe 07 8e 0e
```

From the parameters above, the Enc_structure A_3 is computed.

```
A_3 =
[
  "Encrypt0",
  h'',
  h'21ccb678b79114960955885b90a2b82e3b2ca27e8e374a7907f3e7854367fc22'
]
```

Which encodes to the following byte string to be used as Additional Authenticated Data:

```
A_2 (CBOR-encoded) (45 bytes)
83 68 45 6e 63 72 79 70 74 30 40 58 20 21 cc b6 78 b7 91 14 96 09 55 88 5b
90 a2 b8 2e 3b 2c a2 7e 8e 37 4a 79 07 f3 e7 85 43 67 fc 22
```

The key and nonce used are defined in Appendix C.1.4.2:

- key = K_3
- nonce = IV_3

Using the parameters above, the ciphertext CIPHERTEXT_3 can be computed:

CIPHERTEXT_3 (76 bytes)
```
de 4a 83 3d 48 b6 64 74 14 2c c9 bd ce 87 d9 3a f8 35 57 9c 2d bf 1b 9e 2f
b4 dc 66 60 0d ba c6 bb 3c c0 5c 29 0e f3 5d 51 5b 4d 7d 64 83 f5 09 61 43
b5 56 44 cf af d1 ff aa 7f 2b a3 86 36 57 83 1d d2 e5 bd 04 04 38 60 14 0d
c8
```

C.1.5.4. message_3

From the parameter computed in Appendix C.1.5 and Appendix C.1.5.3, message_3 is computed, as the CBOR Sequence of the following items: (C_V, CIPHERTEXT_3).

```
message_3 =
(
  h'c4',
  h'de4a833d48b66474142cc9bdce87d93af835579c2dbf1b9e2fb4dc66600dbac6bb3cc0
  5c290ef35d515b4d7d6483f5096143b55644cfafd1ffaa7f2ba3863657831dd2e5bd0404
  3860140dc8'
)
```

Which encodes to the following byte string:

message_3 (CBOR Sequence) (80 bytes)
```
41 c4 58 4c de 4a 83 3d 48 b6 64 74 14 2c c9 bd ce 87 d9 3a f8 35 57 9c 2d bf 1b
 9e 2f b4 dc 66 60 0d ba c6 bb 3c c0 5c 29 0e f3 5d 51 5b 4d 7d 64 83 f5 09 61 4
3 b5 56 44 cf af d1 ff aa 7f 2b a3 86 36 57 83 1d d2 e5 bd 04 04 38 60 14 0d c8
```

C.1.5.5. OSCORE Security Context Derivation

From the previous message exchange, the Common Security Context for OSCORE [RFC8613] can be derived, as specified in Section 3.3.1.

First af all, TH_4 is computed: TH_4 = H(TH_3, CIPHERTEXT_3), where the input to the hash function is the CBOR Sequence of TH_3 and CIPHERTEXT_3

(TH_3, CIPHERTEXT_3)
(CBOR Sequence) (112 bytes)
```
58 20 21 cc b6 78 b7 91 14 96 09 55 88 5b 90 a2 b8 2e 3b 2c a2 7e 8e 37 4a
79 07 f3 e7 85 43 67 fc 22 58 4c de 4a 83 3d 48 b6 64 74 14 2c c9 bd ce 87
d9 3a f8 35 57 9c 2d bf 1b 9e 2f b4 dc 66 60 0d ba c6 bb 3c c0 5c 29 0e f3
5d 51 5b 4d 7d 64 83 f5 09 61 43 b5 56 44 cf af d1 ff aa 7f 2b a3 86 36 57
83 1d d2 e5 bd 04 04 38 60 14 0d c8
```

And from there, compute the transcript hash TH_4 = SHA-256(TH_3, CIPHERTEXT_3)

TH_4 value (32 bytes)
```
51 ed 39 32 bc ba e8 90 1c 1d 4d eb 94 bd 67 3a b4 d3 8c 34 81 96 09 ee 0d
5c 9d a6 e9 80 7f e5
```

When encoded as a CBOR bstr, that gives:

```
TH_4 (CBOR-encoded) (34 bytes)
58 20 51 ed 39 32 bc ba e8 90 1c 1d 4d eb 94 bd 67 3a b4 d3 8c 34 81 96 09
ee 0d 5c 9d a6 e9 80 7f e5
```

To derive the Master Secret and Master Salt the same HKDF-Expand (PRK, info, L) is used, with different info and L.

For Master Secret:

L for Master Secret = 16

```
Info for Master Secret =
[
  "OSCORE Master Secret",
  [ null, null, null ],
  [ null, null, null ],
  [ 128, h'', h'51ed3932bcbae8901c1d4deb94bd673ab4d38c34819609ee0d5c9da6e9
  807fe5' ]
]
```

When encoded as a CBOR bstr, that gives:

```
info (OSCORE Master Secret) (CBOR-encoded) (68 bytes)
84 74 4f 53 43 4f 52 45 20 4d 61 73 74 65 72 20 53 65 63 72 65 74 83 f6 f6
f6 83 f6 f6 f6 83 18 80 40 58 20 51 ed 39 32 bc ba e8 90 1c 1d 4d eb 94 bd
67 3a b4 d3 8c 34 81 96 09 ee 0d 5c 9d a6 e9 80 7f e5
```

Finally, the Master Secret value computed is:

```
OSCORE Master Secret (16 bytes)
09 02 9d b0 0c 3e 01 27 42 c3 a8 69 04 07 4c 0e
```

For Master Salt:

L for Master Secret = 8

```
Info for Master Salt =
[
  "OSCORE Master Salt",
  [ null, null, null ],
  [ null, null, null ],
  [ 64, h'', h'51ed3932bcbae8901c1d4deb94bd673ab4d38c34819609ee0d5c9da6e98
  07fe5' ]
]
```

When encoded as a CBOR bstr, that gives:

info (OSCORE Master Salt) (CBOR-encoded) (66 bytes)
84 72 4f 53 43 4f 52 45 20 4d 61 73 74 65 72 20 53 61 6c 74 83 f6 f6 f6 83
f6 f6 f6 83 18 40 40 58 20 51 ed 39 32 bc ba e8 90 1c 1d 4d eb 94 bd 67 3a
b4 d3 8c 34 81 96 09 ee 0d 5c 9d a6 e9 80 7f e5

Finally, the Master Secret value computed is:

OSCORE Master Salt (8 bytes)
81 02 97 22 a2 30 4a 06

The Client's Sender ID takes the value of C_V:

Client's OSCORE Sender ID (1 bytes)
c4

The Server's Sender ID takes the value of C_U:

Server's OSCORE Sender ID (1 bytes)
c3

The algorithms are those negociated in the cipher suite:

AEAD Algorithm
10

HMAC Algorithm
5

C.2. Test Vectors for EDHOC Authenticated with Symmetric Keys (PSK)

Symmetric EDHOC is used:

method (Symmetric Authentication)
1

CoAP is used as transport:

corr (Party U is CoAP client)
1

No unprotected opaque application data is sent in the message exchanges.

The pre-defined Cipher Suite 0 is in place both on Party U and Party V, see Section 3.1.

C.2.1. Input for Party U

The following are the parameters that are set in Party U before the first message exchange.

Party U's ephemeral private key (32 bytes)
f4 0c ea f8 6e 57 76 92 33 32 b8 d8 fd 3b ef 84 9c ad b1 9c 69 96 bc 27 2a f1 f6 48 d9 56 6a 4c

Party U's ephemeral public key (value of X_U) (32 bytes)
ab 2f ca 32 89 83 22 c2 08 fb 2d ab 50 48 bd 43 c3 55 c6 43 0f 58 88 97 cb 57 49 61 cf a9 80 6f

Connection identifier chosen by U (value of C_U) (1 bytes)
c1

Pre-shared Key (PSK) (16 bytes)
a1 1f 8f 12 d0 87 6f 73 6d 2d 8f d2 6e 14 c2 de

kid value to identify PSK (1 bytes)
a1

So ID_PSK is defined as the following:

```
ID_PSK =
{
  4:  h'a1'
}
```

This test vector uses COSE_Key objects to store the pre-shared key.

Note that since the map for ID_PSK contains a single 'kid' parameter, ID_PSK is used when transported in the protected header of the COSE Object, but only the kid_value is used when added to the plaintext (see Section 5.1):

ID_PSK (in protected header) (CBOR-encoded) (4 bytes)
a1 04 41 a1

kid_value (in plaintext) (CBOR-encoded) (2 bytes)
41 a1

C.2.2. Input for Party V

The following are the parameters that are set in Party U before the first message exchange.

Party V's ephemeral private key (32 bytes)
d9 81 80 87 de 72 44 ab c1 b5 fc f2 8e 55 e4 2c 7f f9 c6 78 c0 60 51 81 f3
7a c5 d7 41 4a 7b 95

Party V's ephemeral public key (value of X_V) (32 bytes)
fc 3b 33 93 67 a5 22 5d 53 a9 2d 38 03 23 af d0 35 d7 81 7b 6d 1b e4 7d 94
6f 6b 09 a9 cb dc 06

Connection identifier chosen by V (value of C_V) (1 bytes)
c2

Pre-shared Key (PSK) (16 bytes)
a1 1f 8f 12 d0 87 6f 73 6d 2d 8f d2 6e 14 c2 de

kid value to identify PSK (1 bytes)
a1

So ID_PSK is defined as the following:

```
ID_PSK =
{
  4:  h'a1'
}
```

This test vector uses COSE_Key objects to store the pre-shared key.

Note that since the map for ID_PSK contains a single 'kid' parameter, ID_PSK is used when transported in the protected header of the COSE Object, but only the kid_value is used when added to the plaintext (see Section 5.1):

ID_PSK (in protected header) (CBOR-encoded) (4 bytes)
a1 04 41 a1

kid_value (in plaintext) (CBOR-encoded) (2 bytes)
41 a1

C.2.3. Message 1

From the input parameters (in Appendix C.2.1):

TYPE (4 * method + corr)
5

suite
0

SUITES_U : suite
0

G_X (X-coordinate of the ephemeral public key of Party U) (32 bytes)
ab 2f ca 32 89 83 22 c2 08 fb 2d ab 50 48 bd 43 c3 55 c6 43 0f 58 88 97 cb
57 49 61 cf a9 80 6f

C_U (Connection identifier chosen by U) (CBOR encoded) (2 bytes)
41 c1

kid_value of ID_PSK (CBOR encoded) (2 bytes)
41 a1

No UAD_1 is provided, so UAD_1 is absent from message_1.

Message_1 is constructed, as the CBOR Sequence of the CBOR data items above.

```
message_1 =
(
  5,
  0,
  h'ab2fca32898322c208fb2dab5048bd43c355c6430f588897cb574961cfa9806f',
  h'c1',
  h'a1'
)
```

message_1 (CBOR Sequence) (40 bytes)
05 00 58 20 ab 2f ca 32 89 83 22 c2 08 fb 2d ab 50 48 bd 43 c3 55 c6 43 0f
58 88 97 cb 57 49 61 cf a9 80 6f 41 c1 41 a1

C.2.4. Message 2

Since TYPE mod 4 equals 1, C_U is omitted from data_2.

G_Y (X-coordinate of the ephemeral public key of Party V) (32 bytes)
fc 3b 33 93 67 a5 22 5d 53 a9 2d 38 03 23 af d0 35 d7 81 7b 6d 1b e4 7d 94
6f 6b 09 a9 cb dc 06

C_V (Connection identifier chosen by V) (1 bytes)
c2

Data_2 is constructed, as the CBOR Sequence of the CBOR data items above.

```
  data_2 =
  (
    h'fc3b339367a5225d53a92d380323afd035d7817b6d1be47d946f6b09a9cbdc06',
    h'c2'
  )

data_2 (CBOR Sequence) (36 bytes)
58 20 fc 3b 33 93 67 a5 22 5d 53 a9 2d 38 03 23 af d0 35 d7 81 7b 6d 1b e4
7d 94 6f 6b 09 a9 cb dc 06 41 c2
```

From data_2 and message_1 (from Appendix C.2.3), compute the input to the transcript hash TH_2 = H(message_1, data_2), as a CBOR Sequence of these 2 data items.

```
( message_1, data_2 ) (CBOR Sequence)
(76 bytes)
05 00 58 20 ab 2f ca 32 89 83 22 c2 08 fb 2d ab 50 48 bd 43 c3 55 c6 43 0f
58 88 97 cb 57 49 61 cf a9 80 6f 41 c1 41 a1 58 20 fc 3b 33 93 67 a5 22 5d
53 a9 2d 38 03 23 af d0 35 d7 81 7b 6d 1b e4 7d 94 6f 6b 09 a9 cb dc 06 41
c2
```

And from there, compute the transcript hash TH_2 = SHA-256(message_1, data_2)

```
TH_2 value (32 bytes)
16 4f 44 d8 56 dd 15 22 2f a4 63 f2 02 d9 c6 0b e3 c6 9b 40 f7 35 8d 34 1c
db 7b 07 de e1 70 ca
```

When encoded as a CBOR bstr, that gives:

```
TH_2 (CBOR-encoded) (34 bytes)
58 20 16 4f 44 d8 56 dd 15 22 2f a4 63 f2 02 d9 c6 0b e3 c6 9b 40 f7 35 8d
34 1c db 7b 07 de e1 70 ca
```

C.2.4.1. Key and Nonce Computation

The key and nonce for calculating the ciphertext are calculated as follows, as specified in Section 3.3.

HKDF SHA-256 is the HKDF used (as defined by cipher suite 0).

PRK = HMAC-SHA-256(salt, G_XY)

Since this is the symmetric case, salt is the PSK:

```
salt (16 bytes)
a1 1f 8f 12 d0 87 6f 73 6d 2d 8f d2 6e 14 c2 de
```

G_XY is the shared secret, and since the curve25519 is used, the ECDH shared secret is the output of the X25519 function.

```
G_XY (32 bytes)
d5 75 05 50 6d 8f 30 a8 60 a0 63 d0 1b 5b 7a d7 6a 09 4f 70 61 3b 4a e6 6c
5a 90 e5 c2 1f 23 11
```

From there, PRK is computed:

```
PRK (32 bytes)
aa b2 f1 3c cb 1a 4f f7 96 a9 7a 32 a4 d2 fb 62 47 ef 0b 6b 06 da 04 d3 d1
06 39 4b 28 76 e2 8c
```

Key K_2 is the output of HKDF-Expand(PRK, info, L).

info is defined as follows:

```
info for K_2
[
  10,
  [ null, null, null ],
  [ null, null, null ],
  [ 128, h'', h'164f44d856dd15222fa463f202d9c60be3c69b40f7358d341cdb7b07de
  e170ca' ]
]
```

Which as a CBOR encoded data item is:

```
info (K_2) (CBOR-encoded) (48 bytes)
84 0a 83 f6 f6 f6 83 f6 f6 f6 83 18 80 40 58 20 16 4f 44 d8 56 dd 15 22 2f
a4 63 f2 02 d9 c6 0b e3 c6 9b 40 f7 35 8d 34 1c db 7b 07 de e1 70 ca
```

L is the length of K_2, so 16 bytes.

From these parameters, K_2 is computed:

```
K_2 (16 bytes)
ac 42 6e 5e 7d 7a d6 ae 3b 19 aa bd e0 f6 25 57
```

Nonce IV_2 is the output of HKDF-Expand(PRK, info, L).

info is defined as follows:

```
info for IV_2
[
  "IV-GENERATION",
  [ null, null, null ],
  [ null, null, null ],
  [ 104, h'', h'164f44d856dd15222fa463f202d9c60be3c69b40f7358d341cdb7b07de
  e170ca' ]
]
```

Which as a CBOR encoded data item is:

```
info (IV_2) (CBOR-encoded) (61 bytes)
84 6d 49 56 2d 47 45 4e 45 52 41 54 49 4f 4e 83 f6 f6 f6 83 f6 f6 f6 83 18
68 40 58 20 16 4f 44 d8 56 dd 15 22 2f a4 63 f2 02 d9 c6 0b e3 c6 9b 40 f7
35 8d 34 1c db 7b 07 de e1 70 ca
```

L is the length of IV_2, so 13 bytes.

From these parameters, IV_2 is computed:

```
IV_2 (13 bytes)
ff 11 2e 1c 26 8a a2 a7 7c c3 ee 6c 4d
```

C.2.4.2. Ciphertext Computation

COSE_Encrypt0 is computed with the following parameters. Note that UAD_2 is omitted.

- empty protected header
- external_aad = TH_2
- empty plaintext, since UAD_2 is omitted

From the parameters above, the Enc_structure A_2 is computed.

```
A_2 =
[
  "Encrypt0",
  h'',
  h'164f44d856dd15222fa463f202d9c60be3c69b40f7358d341cdb7b07dee170ca'
]
```

Which encodes to the following byte string to be used as Additional Authenticated Data:

A_2 (CBOR-encoded) (45 bytes)
83 68 45 6e 63 72 79 70 74 30 40 58 20 16 4f 44 d8 56 dd 15 22 2f a4 63 f2
02 d9 c6 0b e3 c6 9b 40 f7 35 8d 34 1c db 7b 07 de e1 70 ca

The key and nonce used are defined in Appendix C.2.4.1:

- key = K_2

- nonce = IV_2

Using the parameters above, the ciphertext CIPHERTEXT_2 can be computed:

CIPHERTEXT_2 (8 bytes)
ba 38 b9 a3 fc 1a 58 e9

C.2.4.3. message_2

From the parameter computed in Appendix C.2.4 and Appendix C.2.4.2, message_2 is computed, as the CBOR Sequence of the following items: (G_Y, C_V, CIPHERTEXT_2).

```
message_2 =
(
  h'fc3b339367a5225d53a92d380323afd035d7817b6d1be47d946f6b09a9cbdc06',
  h'c2',
  h'ba38b9a3fc1a58e9'
)
```

Which encodes to the following byte string:

message_2 (CBOR Sequence) (45 bytes)
58 20 fc 3b 33 93 67 a5 22 5d 53 a9 2d 38 03 23 af d0 35 d7 81 7b 6d 1b e4
7d 94 6f 6b 09 a9 cb dc 06 41 c2 48 ba 38 b9 a3 fc 1a 58 e9

C.2.5. Message 3

Since TYPE mod 4 equals 1, C_V is not omitted from data_3.

C_V (1 bytes)
c2

Data_3 is constructed, as the CBOR Sequence of the CBOR data item above.

```
data_3 =
(
  h'c2'
)

data_3 (CBOR Sequence) (2 bytes)
41 c2
```

From data_3, CIPHERTEXT_2 (Appendix C.2.4.2), and TH_2 (Appendix C.2.4), compute the input to the transcript hash TH_2 = H(TH_2 , CIPHERTEXT_2, data_3), as a CBOR Sequence of these 3 data items.

```
( TH_2, CIPHERTEXT_2, data_3 ) (CBOR Sequence) (45 bytes)
58 20 16 4f 44 d8 56 dd 15 22 2f a4 63 f2 02 d9 c6 0b e3 c6 9b 40 f7 35 8d
34 1c db 7b 07 de e1 70 ca 48 ba 38 b9 a3 fc 1a 58 e9 41 c2
```

And from there, compute the transcript hash TH_3 = SHA-256(TH_2 , CIPHERTEXT_2, data_3)

```
TH_3 value (32 bytes)
11 98 aa b3 ed db 61 b8 a1 b1 93 a9 e5 60 2b 5d 5f ea 76 bc 28 52 89 54 81
b5 2b 8a f5 66 d7 fe
```

When encoded as a CBOR bstr, that gives:

```
TH_3 (CBOR-encoded) (34 bytes)
58 20 11 98 aa b3 ed db 61 b8 a1 b1 93 a9 e5 60 2b 5d 5f ea 76 bc 28 52 89
54 81 b5 2b 8a f5 66 d7 fe
```

C.2.5.1. Key and Nonce Computation

The key and nonce for calculating the ciphertext are calculated as follows, as specified in Section 3.3.

HKDF SHA-256 is the HKDF used (as defined by cipher suite 0).

PRK = HMAC-SHA-256(salt, G_XY)

Since this is the symmetric case, salt is the PSK:

```
salt (16 bytes)
a1 1f 8f 12 d0 87 6f 73 6d 2d 8f d2 6e 14 c2 de
```

G_XY is the shared secret, and since the curve25519 is used, the ECDH shared secret is the output of the X25519 function.

```
G_XY (32 bytes)
d5 75 05 50 6d 8f 30 a8 60 a0 63 d0 1b 5b 7a d7 6a 09 4f 70 61 3b 4a e6 6c
5a 90 e5 c2 1f 23 11
```

From there, PRK is computed:

```
PRK (32 bytes)
aa b2 f1 3c cb 1a 4f f7 96 a9 7a 32 a4 d2 fb 62 47 ef 0b 6b 06 da 04 d3 d1
06 39 4b 28 76 e2 8c
```

Key K_3 is the output of HKDF-Expand(PRK, info, L).

info is defined as follows:

```
info for K_3
[
  10,
  [ null, null, null ],
  [ null, null, null ],
  [ 128, h'', h'1198aab3eddb61b8a1b193a9e5602b5d5fea76bc2852895481b52b8af5
  66d7fe' ]
]
```

Which as a CBOR encoded data item is:

```
info (K_3) (CBOR-encoded) (48 bytes)
84 0a 83 f6 f6 f6 83 f6 f6 f6 83 18 80 40 58 20 11 98 aa b3 ed db 61 b8 a1
b1 93 a9 e5 60 2b 5d 5f ea 76 bc 28 52 89 54 81 b5 2b 8a f5 66 d7 fe
```

L is the length of K_3, so 16 bytes.

From these parameters, K_3 is computed:

```
K_3 (16 bytes)
fe 75 e3 44 27 f8 3a ad 84 16 83 c6 6f a3 8a 62
```

Nonce IV_3 is the output of HKDF-Expand(PRK, info, L).

info is defined as follows:

```
info for IV_3
[
  "IV-GENERATION",
  [ null, null, null ],
  [ null, null, null ],
  [ 104, h'', h'1198aab3eddb61b8a1b193a9e5602b5d5fea76bc2852895481b52b8af5
  66d7fe' ]
]
```

Which as a CBOR encoded data item is:

```
info (IV_3) (CBOR-encoded) (61 bytes)
84 6d 49 56 2d 47 45 4e 45 52 41 54 49 4f 4e 83 f6 f6 f6 83 f6 f6 f6 83 18
68 40 58 20 11 98 aa b3 ed db 61 b8 a1 b1 93 a9 e5 60 2b 5d 5f ea 76 bc 28
52 89 54 81 b5 2b 8a f5 66 d7 fe
```

L is the length of IV_3, so 13 bytes.

From these parameters, IV_3 is computed:

```
IV_3 (13 bytes)
60 0a 33 b4 16 de 08 23 52 67 71 ec 8a
```

C.2.5.2. Ciphertext Computation

COSE_Encrypt0 is computed with the following parameters. Note that PAD_2 is omitted.

- empty protected header
- external_aad = TH_3
- empty plaintext, since PAD_2 is omitted

From the parameters above, the Enc_structure A_3 is computed.

```
A_3 =
[
  "Encrypt0",
  h'',
  h'1198aab3eddb61b8a1b193a9e5602b5d5fea76bc2852895481b52b8af566d7fe'
]
```

Which encodes to the following byte string to be used as Additional Authenticated Data:

```
A_3 (CBOR-encoded) (45 bytes)
83 68 45 6e 63 72 79 70 74 30 40 58 20 11 98 aa b3 ed db 61 b8 a1 b1 93 a9
e5 60 2b 5d 5f ea 76 bc 28 52 89 54 81 b5 2b 8a f5 66 d7 fe
```

The key and nonce used are defined in Appendix C.2.5.1:

- key = K_3
- nonce = IV_3

Using the parameters above, the ciphertext CIPHERTEXT_3 can be computed:

CIPHERTEXT_3 (8 bytes)
51 29 07 92 61 45 40 04

C.2.5.3. message_3

From the parameter computed in Appendix C.2.5 and Appendix C.2.5.2, message_3 is computed, as the CBOR Sequence of the following items: (C_V, CIPHERTEXT_3).

```
message_3 =
(
  h'c2',
  h'5129079261454004'
)
```

Which encodes to the following byte string:

message_3 (CBOR Sequence) (11 bytes)
41 c2 48 51 29 07 92 61 45 40 04

C.2.5.4. OSCORE Security Context Derivation

From the previous message exchange, the Common Security Context for OSCORE [RFC8613] can be derived, as specified in Section 3.3.1.

First af all, TH_4 is computed: TH_4 = H(TH_3, CIPHERTEXT_3), where the input to the hash function is the CBOR Sequence of TH_3 and CIPHERTEXT_3

(TH_3, CIPHERTEXT_3)
(CBOR Sequence) (43 bytes)
58 20 11 98 aa b3 ed db 61 b8 a1 b1 93 a9 e5 60 2b 5d 5f ea 76 bc 28 52 89
54 81 b5 2b 8a f5 66 d7 fe 48 51 29 07 92 61 45 40 04

And from there, compute the transcript hash TH_4 = SHA-256(TH_3, CIPHERTEXT_3)

TH_4 value (32 bytes)
df 7c 9b 06 f5 dc 0e e8 86 0b 39 6c 78 c5 be b7 57 41 3f a7 b6 a9 cf 28 3d
db 4c d4 c1 fd e4 3c

When encoded as a CBOR bstr, that gives:

```
TH_4 (CBOR-encoded) (34 bytes)
58 20 df 7c 9b 06 f5 dc 0e e8 86 0b 39 6c 78 c5 be b7 57 41 3f a7 b6 a9 cf
28 3d db 4c d4 c1 fd e4 3c
```

To derive the Master Secret and Master Salt the same HKDF-Expand (PRK, info, L) is used, with different info and L.

For Master Secret:

L for Master Secret = 16

```
Info for Master Secret =
[
  "OSCORE Master Secret",
  [ null, null, null ],
  [ null, null, null ],
  [ 128, h'', h'df7c9b06f5dc0ee8860b396c78c5beb757413fa7b6a9cf283ddb4cd4c1
  fde43c' ]
]
```

When encoded as a CBOR bstr, that gives:

```
info (OSCORE Master Secret) (CBOR-encoded) (68 bytes)
84 74 4f 53 43 4f 52 45 20 4d 61 73 74 65 72 20 53 65 63 72 65 74 83 f6 f6
f6 83 f6 f6 f6 83 18 80 40 58 20 df 7c 9b 06 f5 dc 0e e8 86 0b 39 6c 78 c5
be b7 57 41 3f a7 b6 a9 cf 28 3d db 4c d4 c1 fd e4 3c
```

Finally, the Master Secret value computed is:

```
OSCORE Master Secret (16 bytes)
8d 36 8f 09 26 2d c5 52 7f e7 19 e6 6c 91 63 75
```

For Master Salt:

L for Master Secret = 8

```
Info for Master Salt =
[
  "OSCORE Master Salt",
  [ null, null, null ],
  [ null, null, null ],
  [ 64, h'', h'df7c9b06f5dc0ee8860b396c78c5beb757413fa7b6a9cf283ddb4cd4c1f
  de43c' ]
]
```

When encoded as a CBOR bstr, that gives:

info (OSCORE Master Salt) (CBOR-encoded) (66 bytes)
84 72 4f 53 43 4f 52 45 20 4d 61 73 74 65 72 20 53 61 6c 74 83 f6 f6 f6 83
f6 f6 f6 83 18 40 40 58 20 df 7c 9b 06 f5 dc 0e e8 86 0b 39 6c 78 c5 be b7
57 41 3f a7 b6 a9 cf 28 3d db 4c d4 c1 fd e4 3c

Finally, the Master Secret value computed is:

OSCORE Master Salt (8 bytes)
4d b7 06 58 c5 e9 9f b6

The Client's Sender ID takes the value of C_V:

Client's OSCORE Sender ID (1 bytes)
c2

The Server's Sender ID takes the value of C_U:

Server's OSCORE Sender ID (1 bytes)
c1

The algorithms are those negociated in the cipher suite:

AEAD Algorithm
10

HMAC Algorithm
5

Acknowledgments

The authors want to thank Alessandro Bruni, Martin Disch, Theis Groenbech Petersen, Dan Harkins, Klaus Hartke, Russ Housley, Alexandros Krontiris, Ilari Liusvaara, Karl Norrman, Salvador Perez, Eric Rescorla, Michael Richardson, Thorvald Sahl Joergensen, Jim Schaad, Carsten Schuermann, Ludwig Seitz, Stanislav Smyshlyaev, Valery Smyslov, Rene Struik, and Erik Thormarker for reviewing and commenting on intermediate versions of the draft. We are especially indebted to Jim Schaad for his continuous reviewing and implementation of different versions of the draft.

Authors' Addresses

Goeran Selander
Ericsson AB

Email: goran.selander@ericsson.com

John Mattsson
Ericsson AB

Email: john.mattsson@ericsson.com

Francesca Palombini
Ericsson AB

Email: francesca.palombini@ericsson.com