Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: June 21, 2021

G. Selander
J. Mattsson
F. Palombini
Ericsson AB

December 18, 2020

Ephemeral Diffie-Hellman Over COSE (EDHOC)
draft-ietf-lake-edhoc-03

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 size 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 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) [RFC8949]. 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]. The construction provided by EDHOC can be applied to authenticate raw public keys (RPK) and public key certificates. This version of the protocol is focusing on RPK and certificates by reference which is the initial focus for the LAKE WG (see Section 2.2 of [I-D.ietf-lake-reqs]).

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.

EDHOC is designed for highly constrained settings making it especially suitable for low-power wide area networks [RFC8376] such as Cellular IoT, 6TiSCH, and LoRaWAN. Compared to the DTLS 1.3 handshake [I-D.ietf-tls-dtls13] with ECDH and connection ID, the number of bytes in EDHOC + CoAP can be less than 1/6 when RPK authentication is used, see [I-D.ietf-lwig-security-protocol-comparison]. Figure 1 shows two examples of message sizes for EDHOC with different kinds of authentication keys and different COSE header parameters for identification: static Diffie-Hellman keys identified by 'kid' [RFC8152], and X.509 signature certificates identified by a hash value using 'x5t' [I-D.ietf-cose-x509]. Further reductions of message sizes are possible, for example by eliding redundant length indications.

	kid	x5t
message_1	37	37
message_2	46	117
message_3	20	91
Total	103	245

Figure 1: Example of message sizes in bytes.

The ECDH exchange and the key derivation follow known protocol constructions such as [SIGMA], NIST SP-800-56A [SP-800-56A], and Extract-and-Expand [RFC5869]. CBOR [RFC8949] 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 authenticated with digital signatures builds on SIGMA-I, Section 3 specifies general properties of EDHOC, including message flow and formatting of the ephemeral public keys, Section 4 specifies the key derivation, Section 5 specifies EDHOC with signature key and static Diffie-Hellman 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 constrained 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 network technologies with duty cycle, 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 impacted by a heavy key exchange protocol. A key exchange may need to be executed more than once, e.g. due to a device rebooting or for security reasons such as perfect forward secrecy.

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. Use of EDHOC

EDHOC is designed as a lightweight AKE for OSCORE, i.e. to provide authentication and session key establishment for IoT use cases such as those built on CoAP [RFC7252]. CoAP is a specialized web transfer protocol for use with constrained nodes and networks, providing a request/response interaction model between application endpoints. As such, EDHOC is targeting a large variety of use cases involving 'things' with embedded microcontrollers, sensors and actuators.

A typical setting is when one of the endpoints is constrained or in a constrained network, and the other endpoint is a node on the Internet (such as a mobile phone) or at the edge of the constrained network (such as a gateway). Thing-to-thing interactions over constrained networks are also relevant since both endpoints would then benefit from the lightweight properties of the protocol. EDHOC could e.g. be run when a device/device(s) connect(s) for the first time, or to establish fresh keys which are not compromised by a later compromise of the long-term keys. (Further security properties are described in Section 8.1.)

1.3. 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.

Readers are expected to be familiar with the terms and concepts described in CBOR [RFC8949], CBOR Sequences [RFC8742], COSE [RFC8152], and CDDL [RFC8610]. The Concise Data Definition Language (CDDL) is used to express CBOR data structures [RFC8949]. Examples of CBOR and CDDL are provided in Appendix A.1.

2. EDHOC Outline

EDHOC specifies different authentication methods of the Diffie-Hellman key exchange: digital signatures and static Diffie-Hellman keys. This section outlines the digital signature based method. Further details of protocol elements and other authentication methods are provided in the remainder of this document.

SIGMA (SIGn-and-MAc) is a family of theoretical protocols with a large number of variants [SIGMA]. Like IKEv2 [RFC7296] and (D)TLS 1.3 [RFC8446], EDHOC authenticated with digital signatures is built on a variant of the SIGMA protocol which provide identity protection of the initiator (SIGMA-I), and like IKEv2 [RFC7296], EDHOC

implements the SIGMA-I variant as MAC-then-Sign. The SIGMA-I protocol using an authenticated encryption algorithm is shown in Figure 2.

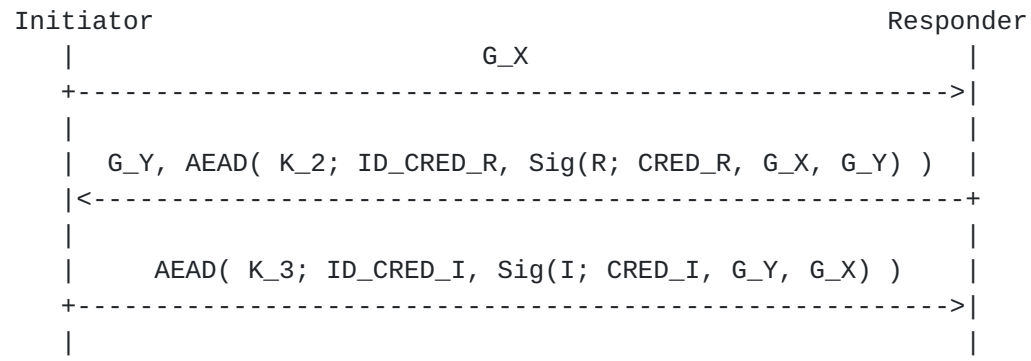

Figure 2: Authenticated encryption variant of the SIGMA-I protocol.

The parties exchanging messages are called Initiator (I) and Responder (R). They exchange ephemeral public keys, compute the shared secret, and derive symmetric application keys.

- G_X and G_Y are the ECDH ephemeral public keys of I and R, respectively.
- CRED_I and CRED_R are the credentials containing the public authentication keys of I and R, respectively.
- ID_CRED_I and ID_CRED_R are data enabling the recipient party to retrieve the credential of I and R, respectively.
- Sig(I; .) and S(R; .) denote signatures made with the private authentication key of I and R, respectively.
- AEAD(K; .) denotes authenticated encryption with additional data using a key K derived from the shared secret.

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

- Explicit connection identifiers C_I, C_R chosen by I and R, respectively, enabling the recipient to find the protocol state.
- Transcript hashes (hashes of message data) 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 authenticated encryption of different messages.

- Verification of a common preferred cipher suite:
  - The Initiator lists supported cipher suites in order of preference
  - The Responder verifies that the selected cipher suite is the first supported cipher suite
- Method types and error handling.
- Transport of opaque auxiliary 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 and COSE in EDHOC is summarized in Appendix A and test vectors including CBOR diagnostic notation are given in Appendix B.

3. Protocol Elements

3.1. General

EDHOC consists of three messages (message_1, message_2, message_3) between Initiator and Responder, plus an EDHOC error message. EDHOC messages are CBOR Sequences [RFC8742], see Figure 3. The protocol elements in the figure are introduced in the following sections. Message formatting and processing is specified in Section 5 and Section 6. An implementation may support only Initiator or only Responder.

Application data is protected using the agreed application algorithms (AEAD, hash) in the selected cipher suite (see Section 3.4) and the application can make use of the established connection identifiers C_I and C_R (see Section 3.2.4). EDHOC may be used with the media type application/edhoc defined in Section 9.

The Initiator can derive symmetric application keys after creating EDHOC message_3, see Section 4.1. Application protected data can therefore be sent in parallel with EDHOC message_3, optionally in the same CoAP message [I-D.palombini-core-oscore-edhoc].

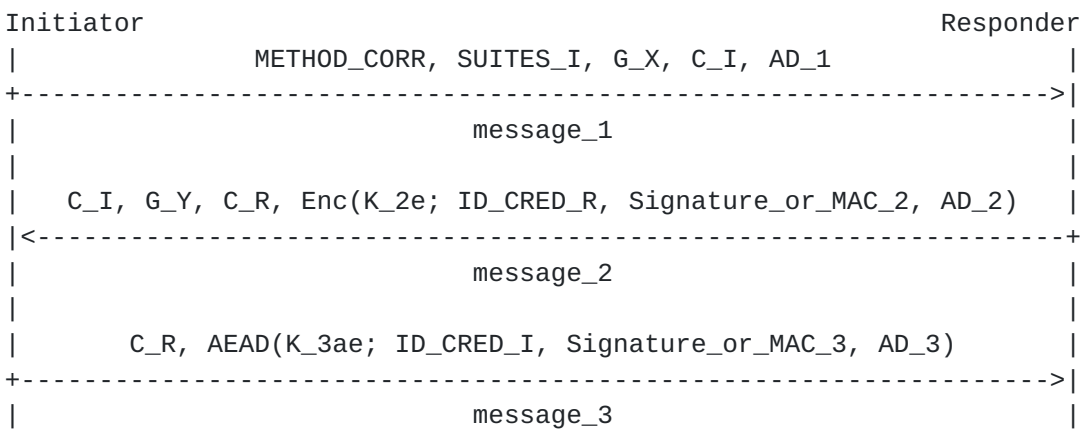

Figure 3: EDHOC Message Flow

3.2. Method and Correlation

The first data item of message_1, METHOD_CORR (see Section 5.2.1), is an integer specifying the method and the correlation properties of the transport, which are described in this section.

3.2.1. Method

EDHOC supports authentication with signature or static Diffie-Hellman keys, as defined in the four authentication methods: 0, 1, 2, and 3, see Figure 4. (Method 0 corresponds to the case outlined in Section 2 where both Initiator and Responder authenticate with signature keys.)

An implementation may support only a single method. The Initiator and the Responder need to have agreed on a single method to be used for EDHOC, see Appendix C.

Value	Initiator	Responder	Reference
0	Signature Key	Signature Key	[[this document]]
1	Signature Key	Static DH Key	[[this document]]
2	Static DH Key	Signature Key	[[this document]]
3	Static DH Key	Static DH Key	[[this document]]

Figure 4: Method Types

3.2.2. Connection Identifiers

EDHOC includes connection identifiers (C_I, C_R) to correlate messages. The connection identifiers C_I and C_R do not have any cryptographic purpose in EDHOC. They contain information facilitating retrieval of the protocol state and may therefore be very short. One byte connection identifiers are realistic in many scenarios as most constrained devices only have a few connections. In cases where a node only has one connection, the identifiers may even be the empty byte string.

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. (For OSCORE this results in the endpoint selecting its Recipient ID, see Section 3.1 of [RFC8613]).

3.2.3. Transport

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. The transport is responsible to handle message loss, reordering, message duplication, fragmentation, and denial of service protection, where necessary.

The Initiator and the Responder need to have agreed on a transport to be used for EDHOC, see Appendix C. It is recommended to transport EDHOC in CoAP payloads, see Section 7.

3.2.4. Message Correlation

If the transport provides a mechanism for correlating messages, some of the connection identifiers may be omitted. There are four cases:

- corr = 0, the transport does not provide a correlation mechanism.
- corr = 1, the transport provides a correlation mechanism that enables the Responder to correlate message_2 and message_1.
- corr = 2, the transport provides a correlation mechanism that enables the Initiator to correlate message_3 and message_2.
- corr = 3, the transport provides a correlation mechanism that enables both parties to correlate all three messages.

For example, if the key exchange is transported over CoAP, the CoAP Token can be used to correlate messages, see Section 7.1.

3.3. Authentication Parameters

3.3.1. Authentication Keys

The authentication key MUST be a signature key or static Diffie-Hellman key. The Initiator and the Responder MAY use different types of authentication keys, e.g. one uses a signature key and the other uses a static Diffie-Hellman key. When using a signature key, the authentication is provided by a signature. When using a static Diffie-Hellman key the authentication is provided by a Message Authentication Code (MAC) computed from an ephemeral-static ECDH shared secret which enables significant reductions in message sizes. The MAC is implemented with an AEAD algorithm. When using a static Diffie-Hellman keys the Initiator's and Responder's private authentication keys are called I and R, respectively, and the public authentication keys are called G_I and G_R, respectively.

- Only the Responder SHALL have access to the Responder's private authentication key.

- Only the Initiator SHALL have access to the Initiator's private authentication key.

3.3.2. Identities

EDHOC assumes the existence of mechanisms (certification authority, trusted third party, manual distribution, etc.) for specifying and distributing authentication keys and identities. Policies are set based on the identity of the other party, and parties typically only allow connections from a specific identity or a small restricted set of identities. For example, in the case of a device connecting to a network, the network may only allow connections from devices which authenticate with certificates having a particular range of serial numbers in the subject field and signed by a particular CA. On the other side, the device may only be allowed to connect to a network which authenticate with a particular public key (information of which may be provisioned, e.g., out of band or in the Auxiliary Data, see Section 3.6).

The EDHOC implementation must be able to receive and enforce information from the application about what is the intended peer endpoint, and in particular whether it is a specific identity or a set of identities.

- When a Public Key Infrastructure (PKI) is used, the trust anchor is a Certification Authority (CA) certificate, and the identity is the subject whose unique name (e.g. a domain name, NAI, or EUI) is included in the endpoint's certificate. Before running EDHOC each party needs at least one CA public key certificate, or just the public key, and a specific identity or set of identities it is allowed to communicate with. Only validated public-key certificates with an allowed subject name, as specified by the application, are to be accepted. EDHOC provides proof that the other party possesses the private authentication key corresponding to the public authentication key in its certificate. The certification path provides proof that the subject of the certificate owns the public key in the certificate.

- When public keys are used but not with a PKI (RPK, self-signed certificate), the trust anchor is the public authentication key of the other party. In this case, the identity is typically directly associated to the public authentication key of the other party. For example, the name of the subject may be a canonical representation of the public key. Alternatively, if identities can be expressed in the form of unique subject names assigned to public keys, then a binding to identity can be achieved by including both public key and associated subject name in the protocol message computation: CRED_I or CRED_R may be a self-signed certificate or COSE_Key containing the public authentication key and the subject name, see Section 3.3.3. Before running EDHOC, each endpoint needs a specific public authentication key/unique associated subject name, or a set of public authentication keys/unique associated subject names, which it is allowed to communicate with. EDHOC provides proof that the other party possesses the private authentication key corresponding to the public authentication key.

3.3.3. Authentication Credentials

The authentication credentials, CRED_I and CRED_R, contain the public authentication key of the Initiator and the Responder, respectively. The Initiator and the Responder MAY use different types of credentials, e.g. one uses an RPK and the other uses a public key certificate.

The credentials CRED_I and CRED_R are signed or MAC:ed (depending on method) by the Initiator and the Responder, respectively, see Section 5.4 and Section 5.3.

When the credential is a certificate, CRED_x is an end-entity certificate (i.e. not the certificate chain) encoded as a CBOR bstr. In X.509 certificates, signature keys typically have key usage

"digitalSignature" and Diffie-Hellman keys typically have key usage "keyAgreement"

When the credential is a COSE_Key, CRED_x is a CBOR map only containing specific fields from the COSE_Key:

- For COSE_Keys of type OKP the CBOR map SHALL only include the parameters 1 (kty), -1 (crv), and -2 (x-coordinate).

- For COSE_Keys of type EC2 the CBOR map SHALL only include the parameters 1 (kty), -1 (crv), -2 (x-coordinate), and -3 (y-coordinate).

To prevent misbinding attacks in systems where an attacker can register public keys without proving knowledge of the private key, SIGMA [SIGMA] enforces a MAC to be calculated over the "Identity", which in case of a X.509 certificate would be the 'subject' and 'subjectAltName' fields. EDHOC follows SIGMA by calculating a MAC over the whole certificate. While SIGMA paper only focuses on the identity, the same principle is true for any information such as policies connected to the public key.

If the parties have agreed on an identity besides the public key, the identity is included in the CBOR map with the label "subject name", otherwise the subject name is the empty text string. The parameters SHALL be encoded in decreasing order with int labels first and text string labels last. An example of CRED_x when the RPK contains an X25519 static Diffie-Hellman key and the parties have agreed on an EUI-64 identity is shown below:

```
CRED_x = {
  1:  1,
 -1:  4,
 -2:  h'b1a3e89460e88d3a8d54211dc95f0b90
        3ff205eb71912d6db8f4af980d2db83a',
 "subject name" : "42-50-31-FF-EF-37-32-39"
}
```

3.3.4. Identification of Credentials

ID_CRED_I and ID_CRED_R are identifiers of the public authentication keys of the Initiator and the Responder, respectively. ID_CRED_I and ID_CRED_R do not have any cryptographic purpose in EDHOC.

- ID_CRED_R is intended to facilitate for the Initiator to retrieve the Responder's public authentication key.

- ID_CRED_I is intended to facilitate for the Responder to retrieve the Initiator's public authentication key.

The identifiers ID_CRED_I and ID_CRED_R are COSE header_maps, i.e. CBOR maps containing COSE Common Header Parameters, see Section 3.1 of [RFC8152]). In the following we give some examples of COSE header_maps.

Raw public keys are most optimally stored as COSE_Key objects and identified with a 'kid' parameter:

- ID_CRED_x = { 4 : kid_x }, where kid_x : bstr, for x = I or R.

Public key certificates can be identified in different ways. Header parameters for identifying X.509 certificates are defined in [I-D.ietf-cose-x509], for example:

- by a hash value with the 'x5t' parameter;
  - ID_CRED_x = { 34 : COSE_CertHash }, for x = I or R,
- by a URL with the 'x5u' parameter;
  - ID_CRED_x = { 35 : uri }, for x = I or R,

ID_CRED_x MAY contain the actual credential used for authentication, CRED_x. It is RECOMMENDED that they uniquely identify the public authentication key as the recipient may otherwise have to try several keys. ID_CRED_I and ID_CRED_R are transported in the ciphertext, see Section 5.4 and Section 5.3.

When ID_CRED_x does not contain the actual credential it may be very short. One byte credential identifiers are realistic in many scenarios as most constrained devices only have a few keys. In cases where a node only has one key, the identifier may even be the empty byte string.

3.4. Cipher Suites

An EDHOC cipher suite consists of an ordered set of COSE code points from the "COSE Algorithms" and "COSE Elliptic Curves" registries:

- EDHOC AEAD algorithm
- EDHOC hash algorithm
- EDHOC ECDH curve

- EDHOC signature algorithm
- EDHOC signature algorithm curve
- Application AEAD algorithm
- Application hash algorithm

Each cipher suite is identified with a pre-defined int label.

EDHOC can be used with all algorithms and curves defined for COSE. Implementation can either use one of the pre-defined cipher suites (Section 9.1) or use any combination of COSE algorithms to define their own private cipher suite. Private cipher suites can be identified with any of the four values -24, -23, -22, -21.

The following cipher suites are for constrained IoT where message overhead is a very important factor:

0. (10, -16, 4, -8, 6, 10, -16)
   (AES-CCM-16-64-128, SHA-256, X25519, EdDSA, Ed25519,
    AES-CCM-16-64-128, SHA-256)

1. (30, -16, 4, -8, 6, 10, -16)
   (AES-CCM-16-128-128, SHA-256, X25519, EdDSA, Ed25519,
    AES-CCM-16-64-128, SHA-256)

2. (10, -16, 1, -7, 1, 10, -16)
   (AES-CCM-16-64-128, SHA-256, P-256, ES256, P-256,
    AES-CCM-16-64-128, SHA-256)

3. (30, -16, 1, -7, 1, 10, -16)
   (AES-CCM-16-128-128, SHA-256, P-256, ES256, P-256,
    AES-CCM-16-64-128, SHA-256)

The following cipher suite is for general non-constrained applications. It uses very high performance algorithms that also are widely supported:

4. (1, -16, 4, -7, 1, 1, -16)
   (A128GCM, SHA-256, X25519, ES256, P-256,
    A128GCM, SHA-256)

The following cipher suite is for high security application such as government use and financial applications. It is compatible with the CNSA suite [CNSA].

5. (3, -43, 2, -35, 2, 3, -43)
   (A256GCM, SHA-384, P-384, ES384, P-384,
    A256GCM, SHA-384)

The different methods use the same cipher suites, but some algorithms are not used in some methods. The EDHOC signature algorithm and the EDHOC signature algorithm curve are not used in methods without signature authentication.

The Initiator needs to have a list of cipher suites it supports in order of preference. The Responder needs to have a list of cipher suites it supports. SUITES_I is a CBOR array containing cipher suites that the Initiator supports. SUITES_I is formatted and processed as detailed in Section 5.2.1 to secure the cipher suite negotation.

3.5. 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' parameter is included G_X and G_Y. 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' parameter, 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.

3.6. Auxiliary Data

In order to reduce round trips and number of messages, and in some cases also streamline processing, certain security applications may be integrated into EDHOC by transporting auxiliary data together with the messages. One example is the transport of third-party authorization information protected outside of EDHOC [I-D.selander-ace-ake-authz]. Another example is the embedding of a certificate enrolment request or a newly issued certificate.

EDHOC allows opaque auxiliary data (AD) to be sent in the EDHOC messages. Unprotected Auxiliary Data (AD_1, AD_2) may be sent in message_1 and message_2, respectively. Protected Auxiliary Data (AD_3) may be sent in message_3.

Since data carried in AD_1 and AD_2 may not be protected, and the content of AD_3 is available to both the Initiator and the Responder, special considerations need to be made such that the availability of the data a) does not violate security and privacy requirements of the service which uses this data, and b) does not violate the security properties of EDHOC.

3.7. Communication of Protocol Features

EDHOC allows the communication or negotiation of various protocol features during the execution of the protocol.

- The Initiator proposes a cipher suite (see Section 3.4), and the Responder either accepts or rejects, and may make a counter proposal.

- The Initiator decides on the correlation parameter corr (see Section 3.2.4). This is typically given by the transport which the Initiator and the Responder have agreed on beforehand. The Responder either accepts or rejects.

- The Initiator decides on the method parameter, see Figure 4. The Responder either accepts or rejects.

- The Initiator and the Responder decide on the representation of the identifier of their respective credentials, ID_CRED_I and ID_CRED_R. The decision is reflected by the label used in the CBOR map, see for example Section 3.3.4.

Editor's note: This section needs to be aligned with Appendix C.

4. Key Derivation

EDHOC uses Extract-and-Expand [RFC5869] with the EDHOC hash algorithm in the selected cipher suite to derive keys. Extract is used to derive fixed-length uniformly pseudorandom keys (PRK) from ECDH shared secrets. Expand is used to derive additional output keying material (OKM) from the PRKs. The PRKs are derived using Extract.

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

If the EDHOC hash algorithm is SHA-2, then Extract(salt, IKM) = HKDF-Extract(salt, IKM) [RFC5869]. If the EDHOC hash algorithm is SHAKE128, then Extract(salt, IKM) = KMAC128(salt, IKM, 256, ""). If the EDHOC hash algorithm is SHAKE256, then Extract(salt, IKM) = KMAC256(salt, IKM, 512, "").

PRK_2e is used to derive key and IV to encrypt message_2. PRK_3e2m is used to derive keys and IVs produce a MAC in message_2 and to encrypt message_3. PRK_4x3m is used to derive keys and IVs produce a MAC in message_3 and to derive application specific data.

PRK_2e is derived with the following input:

- The salt SHALL be the empty byte string. 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 (calculated from G_X and Y or G_Y and X) as defined in Section 12.4.1 of [RFC8152].

Example: Assuming the use of SHA-256 the extract phase of HKDF produces PRK_2e as follows:

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

where salt = 0x (the empty byte string).

The pseudorandom keys PRK_3e2m and PRK_4x3m are defined as follow:

- If the Responder authenticates with a static Diffie-Hellman key, then PRK_3e2m = Extract(PRK_2e, G_RX), where G_RX is the ECDH shared secret calculated from G_R and X, or G_X and R, else PRK_3e2m = PRK_2e.

- If the Initiator authenticates with a static Diffie-Hellman key, then PRK_4x3m = Extract(PRK_3e2m, G_IY), where G_IY is the ECDH shared secret calculated from G_I and Y, or G_Y and I, else PRK_4x3m = PRK_3e2m.

Example: Assuming the use of curve25519, the ECDH shared secrets G_XY, G_RX, and G_IY are the outputs of the X25519 function [RFC7748]:

```
G_XY = X25519( Y, G_X ) = X25519( X, G_Y )
```

The keys and IVs used in EDHOC are derived from PRK using Expand [RFC5869] where the EDHOC-KDF is instantiated with the EDHOC AEAD algorithm in the selected cipher suite.

```
OKM = EDHOC-KDF( PRK, transcript_hash, label, length )
    = Expand( PRK, info, length )
```

where info is the CBOR encoding of

```
info = [
   edhoc_aead_id : int / tstr,
   transcript_hash : bstr,
   label : tstr,
   length : uint
]
```

where

- edhoc_aead_id is an int or tstr containing the algorithm identifier of the EDHOC AEAD algorithm in the selected cipher suite encoded as defined in [RFC8152]. Note that a single fixed edhoc_aead_id is used in all invocations of EDHOC-KDF, including the derivation of K_2e and invocations of the EDHOC-Exporter.

- transcript_hash is a bstr set to one of the transcript hashes TH_2, TH_3, or TH_4 as defined in Sections 5.3.1, 5.4.1, and 4.1.

- label is a tstr set to the name of the derived key or IV, i.e. "K_2m", "IV_2m", "K_2e", "IV_2e", "K_3m", "IV_3m", "K_3ae", or "IV_3ae".

- length is the length of output keying material (OKM) in bytes

If the EDHOC hash algorithm is SHA-2, then Expand(PRK, info, length) = HKDF-Expand(PRK, info, length) [RFC5869]. If the EDHOC hash algorithm is SHAKE128, then Expand(PRK, info, length) = KMAC128(PRK, info, L, ""). If the EDHOC hash algorithm is SHAKE256, then Expand(PRK, info, length) = KMAC256(PRK, info, L, "").

K_2e and IV_2e are derived using the transcript hash TH_2 and the pseudorandom key PRK_2e. K_2m and IV_2m are derived using the transcript hash TH_2 and the pseudorandom key PRK_3e2m. K_3ae and IV_3ae are derived using the transcript hash TH_3 and the pseudorandom key PRK_3e2m. K_3m and IV_3m are derived using the transcript hash TH_3 and the pseudorandom key PRK_4x3m. IVs are only used if the EDHOC AEAD algorithm uses IVs.

4.1. EDHOC-Exporter Interface

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

```
EDHOC-Exporter(label, length)
  = EDHOC-KDF(PRK_4x3m, TH_4, label, length)
```

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 selected cipher suite. Example use of the EDHOC-Exporter is given in Sections 7.1.1.

To provide forward secrecy in an even more efficient way than re-running EDHOC, EDHOC provides the function EDHOC-Exporter-FS. When EHDOC-Exporter-FS is called the old PRK_4x3m is deleted and the new PRk_4x3m is calculated as a "hash" of the old key using the Extract function as illustrated by the following pseudocode:

```
EHDOC-Exporter-FS( nonce ):
   PRK_4x3m = Extract( [ "TH_4", nonce ], PRK_4x3m )
```

5. Message Formatting and Processing

This section specifies formatting of the messages and processing steps. Error messages are specified in Section 6.

An EDHOC message is encoded as a sequence of CBOR data (CBOR Sequence, [RFC8742]). Additional optimizations are made to reduce message overhead.

While EDHOC uses the COSE_Key, COSE_Sign1, and COSE_Encrypt0 structures, only a subset of the parameters is included in the EDHOC messages. The unprotected COSE header in COSE_Sign1, and COSE_Encrypt0 (not included in the EDHOC message) MAY contain parameters (e.g. 'alg').

5.1. Encoding of bstr_identifier

Byte strings are encoded in CBOR as two or more bytes, whereas integers in the interval -24 to 23 are encoded in CBOR as one byte.

bstr_identifier is a special encoding of byte strings, used throughout the protocol to enable the encoding of the shortest byte strings as integers that only require one byte of CBOR encoding.

The bstr_identifier encoding is defined as follows: Byte strings in the interval h'00' to h'2f' are encoded as the corresponding integer minus 24, which are all represented by one byte CBOR ints. Other byte strings are encoded as CBOR byte strings.

For example, the byte string h'59e9' encoded as a bstr_identifier is equal to h'59e9', while the byte string h'2a' is encoded as the integer 18.

The CDDL definition of the bstr_identifier is given below:

```
bstr_identifier = bstr / int
```

Note that, despite what could be interpreted by the CDDL definition only, bstr_identifier once decoded are always byte strings.

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 = (
  METHOD_CORR : int,
  SUITES_I : [ selected : suite, supported : 2* suite ] / suite,
  G_X : bstr,
  C_I : bstr_identifier,
  ? AD_1 : bstr,
)

suite = int
```

where:

- METHOD_CORR = 4 * method + corr, where method = 0, 1, 2, or 3 (see Figure 4) and the correlation parameter corr is chosen based on the transport and determines which connection identifiers that are omitted (see Section 3.2.4).

- SUITES_I - cipher suites which the Initiator supports in order of (decreasing) preference. The list of supported cipher suites can be truncated at the end, as is detailed in the processing steps below. One of the supported cipher suites is selected. The selected suite is the first suite in the SUITES_I CBOR array. If a single supported cipher suite is conveyed then that cipher suite is selected and the selected cipher suite is encoded as an int instead of an array.

- G_X - the ephemeral public key of the Initiator

- C_I - variable length connection identifier, encoded as a bstr_identifier (see Section 5.1).

- AD_1 - bstr containing unprotected opaque auxiliary data

5.2.2. Initiator Processing of Message 1

The Initiator 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_I sent to the Responder 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 the Responder in message_1. If the Initiator previously received from the Responder an error message to a message_1 with diagnostic payload identifying a cipher suite that the Initiator supports, then the Initiator SHALL use that cipher suite. Otherwise the first supported (i.e. the most preferred) cipher suite in SUITES_I 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 and format it as a COSE_Key. Let G_X be the 'x' parameter of the COSE_Key.

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

- Encode message_1 as a sequence of CBOR encoded data items as specified in Section 5.2.1

5.2.3. Responder Processing of Message 1

The Responder 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 suite in SUITES_I is supported.

- Pass AD_1 to the security application.

If any verification step fails, the Responder MUST send an EDHOC error message back, formatted as defined in Section 6, and the protocol MUST be discontinued. If the Responder does not support the selected cipher suite, then SUITES_R MUST include one or more

supported cipher suites. If the Responder does not support the selected cipher suite, but supports another cipher suite in SUITES_I, then SUITES_R MUST include the first supported cipher suite in SUITES_I.

5.3. EDHOC Message 2

5.3.1. 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_I : bstr_identifier,
  G_Y : bstr,
  C_R : bstr_identifier,
)
```

where:

- G_Y - the ephemeral public key of the Responder
- C_R - variable length connection identifier, encoded as a bstr_identifier (see Section 5.1).

5.3.2. Responder Processing of Message 2

The Responder SHALL compose message_2 as follows:

- If corr (METHOD_CORR mod 4) equals 1 or 3, C_I is omitted, otherwise C_I 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 and format it as a COSE_Key. Let G_Y be the 'x' parameter of the COSE_Key.
- Choose a connection identifier C_R 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 selected cipher suite. The transcript

hash TH_2 is a CBOR encoded bstr and the input to the hash function is a CBOR Sequence.

- Compute an inner COSE_Encrypt0 as defined in Section 5.3 of [RFC8152], with the EDHOC AEAD algorithm in the selected cipher suite, K_2m, IV_2m, and the following parameters:

  * protected = << ID_CRED_R >>

    + ID_CRED_R - identifier to facilitate retrieval of CRED_R, see Section 3.3.4

  * external_aad = << TH_2, CRED_R, ? AD_2 >>

    + CRED_R - bstr containing the credential of the Responder, see Section 3.3.4.

    + AD_2 = bstr containing opaque unprotected auxiliary data

  * plaintext = h''

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

  * Key K = EDHOC-KDF(PRK_3e2m, TH_2, "K_2m", length)

  * Nonce N = EDHOC-KDF(PRK_3e2m, TH_2, "IV_2m", length)

  * Plaintext P = 0x (the empty string)

  * Associated data A =

    ["Encrypt0", << ID_CRED_R >>, << TH_2, CRED_R, ? AD_2 >>]

  MAC_2 is the 'ciphertext' of the inner COSE_Encrypt0.

- If the Responder authenticates with a static Diffie-Hellman key (method equals 1 or 3), then Signature_or_MAC_2 is MAC_2. If the Responder authenticates with a signature key (method equals 0 or 2), then Signature_or_MAC_2 is the 'signature' of a COSE_Sign1 object as defined in Section 4.4 of [RFC8152] using the signature algorithm in the selected cipher suite, the private authentication key of the Responder, and the following parameters:

  * protected = << ID_CRED_R >>

  * external_aad = << TH_2, CRED_R, ? AD_2 >>

  * payload = MAC_2

COSE constructs the input to the Signature Algorithm as:

* The key is the private authentication key of the Responder.

* The message M to be signed =

  ["Signature1", << ID_CRED_R >>, << TH_2, CRED_R, ? AD_2 >>, MAC_2]

o Compute an outer COSE_Encrypt0 as defined in Section 5.3 of [RFC8152], with the EDHOC AEAD algorithm in the selected cipher suite, K_2e, IV_2e, and the following parameters. The protected header SHALL be empty.

* plaintext = (ID_CRED_R / bstr_identifier, Signature_or_MAC_2, ? AD_2)

  + Note that if ID_CRED_R contains a single 'kid' parameter, i.e., ID_CRED_R = { 4 : kid_R }, only the byte string kid_R is conveyed in the plaintext encoded as a bstr_identifier, see Section 3.3.4 and Section 5.1.

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

* Key K = EDHOC-KDF(PRK_2e, TH_2, "K_2e", length)

* Nonce N = EDHOC-KDF(PRK_2e, TH_2, "IV_2e", length)

* Plaintext P = (ID_CRED_R / bstr_identifier, Signature_or_MAC_2, ? AD_2)

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

CIPHERTEXT_2 is the 'ciphertext' of the outer COSE_Encrypt0 with the tag removed.

o Encode message_2 as a sequence of CBOR encoded data items as specified in Section 5.3.1.

5.3.3. Initiator Processing of Message 2

The Initiator SHALL process message_2 as follows:

o Decode message_2 (see Appendix A.1).

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

- Decrypt CIPHERTEXT_2 by computing an outer COSE_Encrypt0 as defined in see Section 5.3.2 and XORing CIPHERTEXT_2 with the 'ciphertext' of the outer COSE_Encrypt0 with the tag removed.

- Verify that the identity of the Responder is an allowed identity for this connection, see Section 3.3.

- Verify Signature_or_MAC_2 using the algorithm in the selected cipher suite. The verification process depends on the method, see Section 5.3.2.

- Pass AD_2 to the security application.

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

5.4. EDHOC Message 3

5.4.1. 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_R : bstr_identifier,
)
```

5.4.2. Initiator Processing of Message 3

The Initiator SHALL compose message_3 as follows:

- If corr (METHOD_CORR mod 4) equals 2 or 3, C_R is omitted, otherwise C_R is not omitted.

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

- Compute an inner COSE_Encrypt0 as defined in Section 5.3 of [RFC8152], with the EDHOC AEAD algorithm in the selected cipher suite, K_3m, IV_3m, and the following parameters:

* protected = << ID_CRED_I >>

  + ID_CRED_I - identifier to facilitate retrieval of CRED_I, see Section 3.3.4

* external_aad = << TH_3, CRED_I, ? AD_3 >>

  + CRED_I - bstr containing the credential of the Initiator, see Section 3.3.4.

  + AD_3 = bstr containing opaque protected auxiliary data

* plaintext = h''

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

* Key K = EDHOC-KDF(PRK_4x3m, TH_3, "K_3m", length)

* Nonce N = EDHOC-KDF(PRK_4x3m, TH_3, "IV_3m", length)

* Plaintext P = 0x (the empty string)

* Associated data A =

  ["Encrypt0", << ID_CRED_I >>, << TH_3, CRED_I, ? AD_3 >>]

MAC_3 is the 'ciphertext' of the inner COSE_Encrypt0.

o If the Initiator authenticates with a static Diffie-Hellman key (method equals 2 or 3), then Signature_or_MAC_3 is MAC_3. If the Initiator authenticates with a signature key (method equals 0 or 1), then Signature_or_MAC_3 is the 'signature' of a COSE_Sign1 object as defined in Section 4.4 of [RFC8152] using the signature algorithm in the selected cipher suite, the private authentication key of the Initiator, and the following parameters:

* protected = << ID_CRED_I >>

* external_aad = << TH_3, CRED_I, ? AD_3 >>

* payload = MAC_3

COSE constructs the input to the Signature Algorithm as:

* The key is the private authentication key of the Initiator.

* The message M to be signed =

["Signature1", << ID_CRED_I >>, << TH_3, CRED_I, ? AD_3 >>, MAC_3]

- Compute an outer COSE_Encrypt0 as defined in Section 5.3 of [RFC8152], with the EDHOC AEAD algorithm in the selected cipher suite, K_3ae, IV_3ae, and the following parameters. The protected header SHALL be empty.

  * external_aad = TH_3

  * plaintext = (ID_CRED_I / bstr_identifier, Signature_or_MAC_3, ? AD_3)

    + Note that if ID_CRED_I contains a single 'kid' parameter, i.e., ID_CRED_I = { 4 : kid_I }, only the byte string kid_I is conveyed in the plaintext encoded as a bstr_identifier, see Section 3.3.4 and Section 5.1.

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

  * Key K = EDHOC-KDF(PRK_3e2m, TH_3, "K_3ae", length)

  * Nonce N = EDHOC-KDF(PRK_3e2m, TH_3, "IV_3ae", length)

  * Plaintext P = (ID_CRED_I / bstr_identifier, Signature_or_MAC_3, ? AD_3)

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

  CIPHERTEXT_3 is the 'ciphertext' of the outer COSE_Encrypt0.

- Encode message_3 as a sequence of CBOR encoded data items as specified in Section 5.4.1.

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

After sending message_3, the Initiator is assured that no other party than the Responder can compute the key PRK_4x3m (implicit key authentication). The Initiator does however not know that the Responder has actually computed the key PRK_4x3m. While the Initiator can securely send protected application data, the Initiator SHOULD NOT store the keying material PRK_4x3m and TH_4 until the Initiator is assured that the Responder has actually computed the key PRK_4x3m (explicit key confirmation). Explicit key confirmation is

e.g. assured when the Initiator has verified an OSCORE message from the Responder.

5.4.3. Responder Processing of Message 3

The Responder SHALL process message_3 as follows:

- Decode message_3 (see Appendix A.1).
- Retrieve the protocol state using the connection identifier C_R and/or other external information such as the CoAP Token and the 5-tuple.
- Decrypt and verify the outer COSE_Encrypt0 as defined in Section 5.3 of [RFC8152], with the EDHOC AEAD algorithm in the selected cipher suite, K_3ae, and IV_3ae.
- Verify that the identity of the Initiator is an allowed identity for this connection, see Section 3.3.
- Verify Signature_or_MAC_3 using the algorithm in the selected cipher suite. The verification process depends on the method, see Section 5.4.2.
- Pass AD_3, the connection identifiers (C_I, C_R), and the application algorithms in the selected cipher suite to the security application. The application can now derive application keys using the EDHOC-Exporter interface.

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

After verifying message_3, the Responder is assured that the Initiator has calculated the key PRK_4x3m (explicit key confirmation) and that no other party than the Responder can compute the key. The Responder can securely send protected application data and store the keying material PRK_4x3m and TH_4.

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_identifier,
  ERR_MSG : tstr,
  ? SUITES_R : [ supported : 2* suite ] / suite,
)
```

where:

- C_x - (optional) variable length connection identifier, encoded as a bstr_identifier (see Section 5.1). If error is sent by the Responder and corr (METHOD_CORR mod 4) equals 0 or 2 then C_x is set to C_I, else if error is sent by the Initiator and corr (METHOD_CORR mod 4) equals 0 or 1 then C_x is set to C_R, 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. This text string is mandatory and characteristic for error messages, which enables the receiver to distinguish between a normal message and an error message of the protocol.

- SUITES_R - (optional) cipher suites from SUITES_I or the EDHOC cipher suites registry that the Responder supports. SUITES_R MUST only be included in replies to message_1. If a single supported cipher suite is conveyed then the supported cipher suite is encoded as an int instead of an array.

After receiving SUITES_R, the Initiator can determine which selected cipher suite to use for the next EDHOC run with the Responder. If the Initiator intends to contact the Responder in the future, the Initiator SHOULD remember which selected cipher suite to use until the next message_1 has been sent, otherwise the Initiator and Responder will likely run into an infinite loop. After a successful run of EDHOC, the Initiator MAY remember the selected cipher suite to use in future EDHOC runs. Note that if the Initiator or Responder is updated with new cipher suite policies, any cached information may be outdated.

6.1.1. Example Use of EDHOC Error Message with SUITES_R

Assuming that the Initiator supports the five cipher suites 5, 6, 7, 8, and 9 in decreasing order of preference, Figures 5 and 6 show examples of how the Initiator can truncate SUITES_I and how SUITES_R is used by the Responder to give the Initiator information about the cipher suites that the Responder supports.

In Figure 5, the Responder supports cipher suite 6 but not the initially selected cipher suite 5.

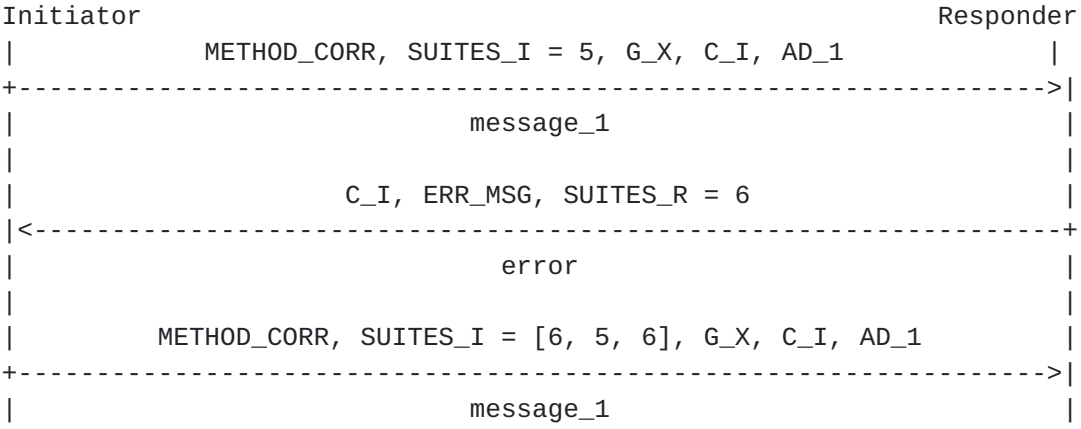

Figure 5: Example use of error message with SUITES_R.

In Figure 6, the Responder supports cipher suite 7 and 9 but not the more preferred (by the Initiator) cipher suites 5 and 6. The order of cipher suites in SUITES_R does not matter.

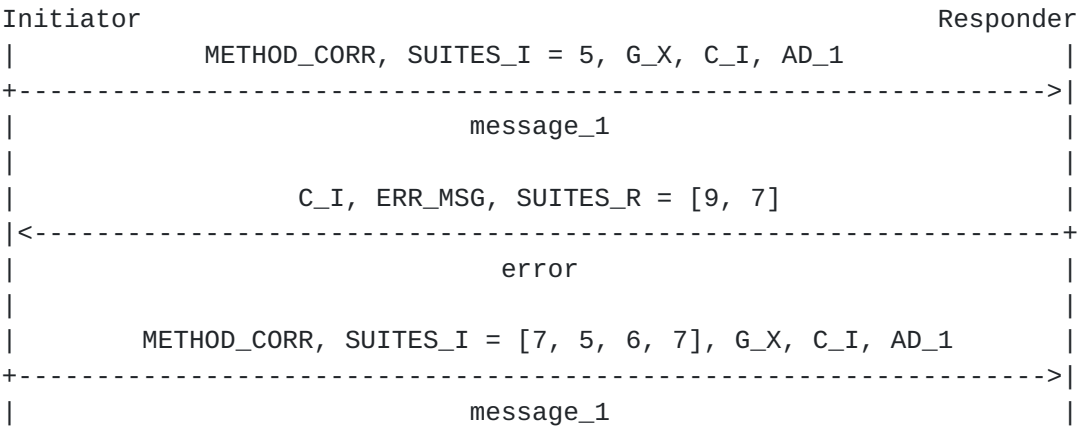

Figure 6: Example use of error message with SUITES_R.

Note that the Initiator's list of supported cipher suites and order of preference is fixed (see Section 5.2.1 and Section 5.2.2). Furthermore, the Responder shall only accept message_1 if the

selected cipher suite is the first cipher suite in SUITES_I that the Responder supports (see Section 5.2.3). Following this procedure ensures that the selected cipher suite is the most preferred (by the Initiator) cipher suite supported by both parties.

If the selected cipher suite is not the first cipher suite which the Responder supports in SUITES_I received in message_1, then Responder MUST discontinue the protocol, see Section 5.2.3. If SUITES_I in message_1 is manipulated then the integrity verification of message_2 containing the transcript hash TH_2 = H(message_1, data_2) will fail and the Initiator will discontinue the protocol.

7. Transferring EDHOC and Deriving an OSCORE Context

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 messages in Confirmable messages, especially if fragmentation is used.

By default, the CoAP client is the Initiator and the CoAP server is the Responder, 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 7. In this case the CoAP Token enables the Initiator to correlate message_1 and message_2 so the correlation parameter corr = 1.

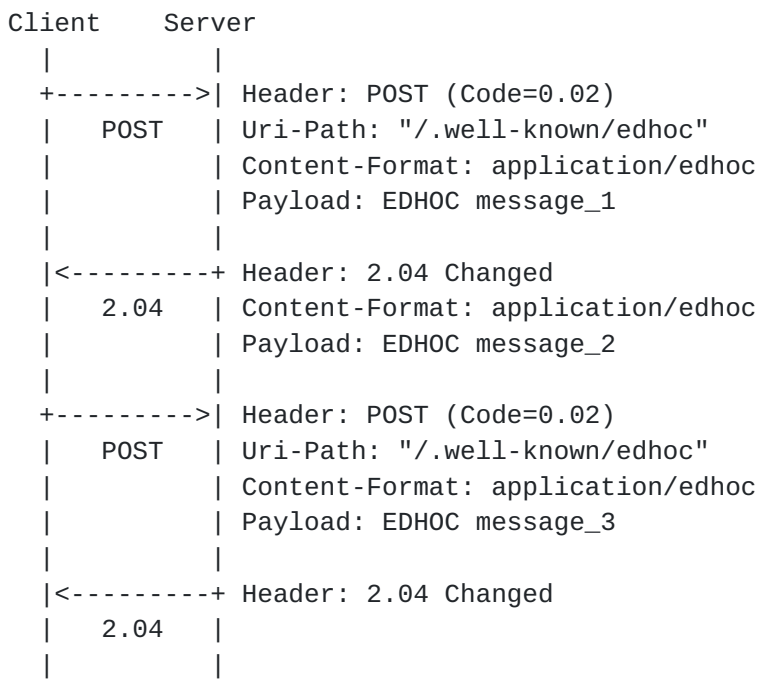

Figure 7: Transferring EDHOC in CoAP when the Initiator is CoAP Client

The exchange in Figure 7 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 8. In this case the CoAP Token enables the Responder to correlate message_2 and message_3 so the correlation parameter corr = 2.

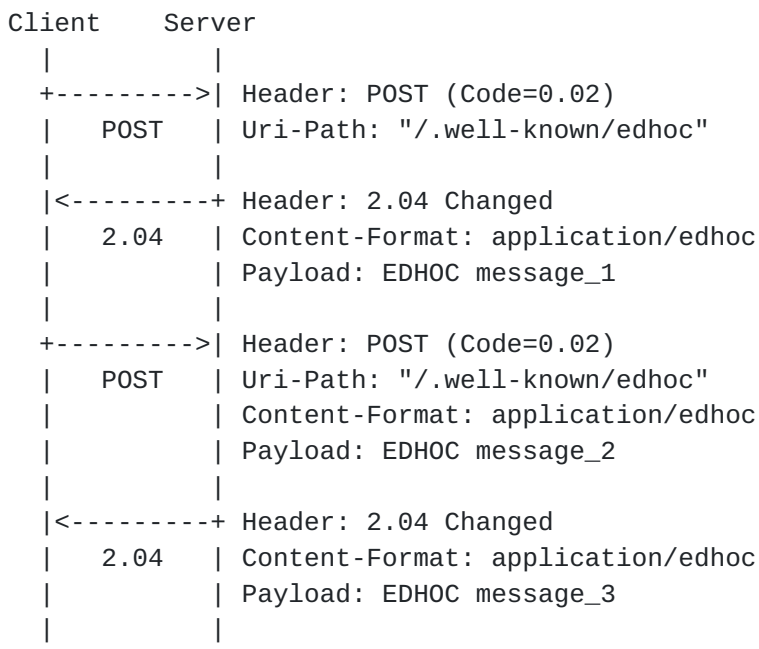

Figure 8: Transferring EDHOC in CoAP when the Initiator is CoAP Server

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 make sure that the EDHOC connection identifiers are unique, i.e. C_R MUST NOT be equal to C_I. 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 the Initiator and the CoAP server is the Responder:

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

- The AEAD Algorithm and the hash algorithm are the application AEAD and hash 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 application AEAD Algorithm.

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

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. As described in [SIGMA], peer awareness is provided to the Responder, but not to the Initiator.

EDHOC protects the credential identifier of the Initiator against active attacks and the credential identifier of the Responder against passive attacks. The roles should be assigned to protect the most sensitive identity/identifier, 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, auxiliary 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 the Responder to verify that the selected cipher suite is the most preferred cipher suite by the Initiator which is supported by both the Initiator and the Responder.

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 (private signature or static DH keys) does not compromise the security of completed EDHOC exchanges. Compromising the private authentication keys of one party lets an active 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. Compromise of the long-term keys does not enable a passive attacker to compromise future session keys. Compromise of the HDKF input parameters (ECDH shared secret) leads to compromise of all session keys derived from that compromised shared secret. Compromise of one session key does not compromise other session keys.

If supported by the device, it is RECOMMENDED that at least the long-term private keys is stored in a Trusted Execution Environment (TEE) and that sensitive operations using these keys are performed inside the TEE. To achieve even higher security additional operation such as ephemeral key generation, all computations of shared secrets, and storage of the PRK keys can be done inside the TEE. Optimally, the whole EDHOC protocol can be implemented inside the TEE. Typically an adversary with physical access to a device can be assumed to gain access to all information outside of the TEE, but none of the information inside the TEE.

Key compromise impersonation (KCI): In EDHOC authenticated with signature keys, EDHOC provides KCI protection against an attacker having access to the long term key or the ephemeral secret key. With static 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.

Repudiation: In EDHOC authenticated with signature keys, the Initiator could theoretically prove that the Responder 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 static Diffie-Hellman key authentication, both parties can always deny having participated in the protocol.

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. The Initiator and the Responder 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-16-64-128, SHA-256, X25519, EdDSA, Ed25519, AES-CCM-16-64-128, SHA-256) is mandatory to implement. Implementations only need to implement the algorithms needed for their supported methods. 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

The Initiator and the Responder 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 AD_1, ID_CRED_R, AD_2, and ERR_MSG. Using the same AD_1 in several EDHOC sessions allows passive eavesdroppers to correlate the different sessions. Another consideration is that the list of supported cipher suites may potentially be used to identify the application.

The Initiator and the Responder must also make sure that unauthenticated data does not trigger any harmful actions. In particular, this applies to AD_1 and ERR_MSG.

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 the Responder 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 pseudorandom number must only be used once, an implementation need to get a new truly random seed after reboot, or continuously 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, and IVs MUST be secret. Implementations should provide countermeasures to side-channel attacks such as timing attacks. Depending on the selected curve, the parties should perform various validations of each other's public keys, see e.g. Section 5 of [SP-800-56A].

The Initiator and the Responder 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 MUST be kept secret.

The Initiator and the Responder are allowed to select the connection identifiers C_I and C_R, 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).

The Responder MUST finish the verification step of message_3 before passing AD_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: -24
Algorithms: N/A
Desc: Reserved for Private Use
Reference: [[this document]]

Value: -23
Algorithms: N/A
Desc: Reserved for Private Use
Reference: [[this document]]

```
   Value: -22
   Algorithms: N/A
   Desc: Reserved for Private Use
   Reference: [[this document]]

   Value: -21
   Algorithms: N/A
   Desc: Reserved for Private Use
   Reference: [[this document]]

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

   Value: 1
   Array: 30, 5, 4, -8, 6, 10, 5
   Desc: AES-CCM-16-128-128, SHA-256, X25519, EdDSA, Ed25519,
         AES-CCM-16-64-128, SHA-256
   Reference: [[this document]]

   Value: 2
   Array: 10, 5, 1, -7, 1, 10, 5
   Desc: AES-CCM-16-64-128, SHA-256, P-256, ES256, P-256,
         AES-CCM-16-64-128, SHA-256
   Reference: [[this document]]

   Value: 3
   Array: 30, 5, 1, -7, 1, 10, 5
   Desc: AES-CCM-16-128-128, SHA-256, P-256, ES256, P-256,
         AES-CCM-16-64-128, SHA-256
   Reference: [[this document]]

   Value: 4
   Array: 1, -16, 4, -7, 1, 1, -16
   Desc: A128GCM, SHA-256, X25519, ES256, P-256,
         A128GCM, SHA-256
   Reference: [[this document]]

   Value: 5
   Array: 3, -43, 2, -35, 2, 3, -43
   Desc: A256GCM, SHA-384, P-384, ES384, P-384,
         A256GCM, SHA-384
   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 is shown in Figure 4.

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-core-echo-request-tag]
Amsuess, C., Mattsson, J., and G. Selander, "CoAP: Echo, Request-Tag, and Token Processing", draft-ietf-core-echo-request-tag-11 (work in progress), November 2020.

[I-D.ietf-cose-x509]
Schaad, J., "CBOR Object Signing and Encryption (COSE): Header parameters for carrying and referencing X.509 certificates", draft-ietf-cose-x509-08 (work in progress), December 2020.

[I-D.ietf-lake-reqs]
Vucinic, M., Selander, G., Mattsson, J., and D. Garcia-Carillo, "Requirements for a Lightweight AKE for OSCORE", draft-ietf-lake-reqs-04 (work in progress), June 2020.

[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>.

[RFC8376] Farrell, S., Ed., "Low-Power Wide Area Network (LPWAN) Overview", RFC 8376, DOI 10.17487/RFC8376, May 2018, <https://www.rfc-editor.org/info/rfc8376>.

[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>.

[RFC8742] Bormann, C., "Concise Binary Object Representation (CBOR) Sequences", RFC 8742, DOI 10.17487/RFC8742, February 2020, <https://www.rfc-editor.org/info/rfc8742>.

[RFC8949] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", STD 94, RFC 8949, DOI 10.17487/RFC8949, December 2020, <https://www.rfc-editor.org/info/rfc8949>.

10.2. Informative References

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

[CNSA] (Placeholder), ., "Commercial National Security Algorithm Suite", August 2015, <https://apps.nsa.gov/iaarchive/programs/iad-initiatives/cnsa-suite.cfm>.

[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-36 (work in progress), November 2020.

[I-D.ietf-core-resource-directory] Amsuess, C., Shelby, Z., Koster, M., Bormann, C., and P. Stok, "CoRE Resource Directory", draft-ietf-core-resource-directory-26 (work in progress), November 2020.

[I-D.ietf-lwig-security-protocol-comparison] Mattsson, J., Palombini, F., and M. Vucinic, "Comparison of CoAP Security Protocols", draft-ietf-lwig-security-protocol-comparison-05 (work in progress), November 2020.

[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-dtls13-39 (work in progress), November 2020.

[I-D.palombini-core-oscore-edhoc] Palombini, F., Tiloca, M., Hoeglund, R., Hristozov, S., and G. Selander, "Combining EDHOC and OSCORE", draft-palombini-core-oscore-edhoc-01 (work in progress), November 2020.

[I-D.selander-ace-ake-authz]
Selander, G., Mattsson, J., Vucinic, M., Richardson, M., and A. Schellenbaum, "Lightweight Authorization for Authenticated Key Exchange.", draft-selander-ace-ake-authz-02 (work in progress), November 2020.

[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>.

[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 K", 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>.

[RFC7296] Kaufman, C., Hoffman, P., Nir, Y., Eronen, P., and T. Kivinen, "Internet Key Exchange Protocol Version 2 (IKEv2)", STD 79, RFC 7296, DOI 10.17487/RFC7296, October 2014, <https://www.rfc-editor.org/info/rfc7296>.

[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>.

[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>.

[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 [RFC8949], CDDL [RFC8610], COSE [RFC8152], and HKDF [RFC5869].

A.1. CBOR and CDDL

The Concise Binary Object Representation (CBOR) [RFC8949] 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 [RFC8742] of data

items. Some examples are given below. For a complete specification and more examples, see [RFC8949] 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

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, and COSE_Sign1 objects.

Appendix B. 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 the default mapping to CoAP where the Initiator acts as CoAP client (this means that corr = 1).

A more extensive test vector suite covering more combinations of authentication method used between Initiator and Responder and related code to generate them can be found at https://github.com/lake-wg/edhoc/tree/master/test-vectors .

B.1. Test Vectors for EDHOC Authenticated with Signature Keys (x5t)

EDHOC with signature authentication and X.509 certificates is used. In this test vector, the hash value 'x5t' is used to identify the certificate.

method (Signature Authentication)
0

CoAP is used as transport and the Initiator acts as CoAP client:

corr (the Initiator can correlate message_1 and message_2)
1

From there, METHOD_CORR has the following value:

METHOD_CORR (4 * method + corr) (int)
1

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

The list of supported cipher suites of the Initiator in order of preference is the following:

Supported Cipher Suites (4 bytes)
00 01 02 03

The cipher suite selected by the Initiator is the most preferred:

Selected Cipher Suite (int)
0

The mandatory-to-implement cipher suite 0 is supported by both the Initiator and the Responder, see Section 8.3.

B.1.1. Message_1

X (Initiator's ephemeral private key) (32 bytes)
8f 78 1a 09 53 72 f8 5b 6d 9f 61 09 ae 42 26 11 73 4d 7d bf a0 06 9a 2d
f2 93 5b b2 e0 53 bf 35

G_X (Initiator's ephemeral public key) (32 bytes)
89 8f f7 9a 02 06 7a 16 ea 1e cc b9 0f a5 22 46 f5 aa 4d d6 ec 07 6b ba
02 59 d9 04 b7 ec 8b 0c

The Initiator chooses a connection identifier C_I:

Connection identifier chosen by Initiator (0 bytes)

Since no unprotected opaque auxiliary data is sent in the message exchanges:

AD_1 (0 bytes)

Since the list of supported cipher suites needs to contain the selected cipher suite, the initiator truncates the list of supported cipher suites to one cipher suite only, 00.

Because one single selected cipher suite is conveyed, it is encoded as an int instead of an array:

SUITES_I (int)
0

With SUITES_I = 0, message_1 is constructed, as the CBOR Sequence of the CBOR data items above.

```
message_1 =
(
  1,
  0,
  h'898ff79a02067a16ea1eccb90fa52246f5aa4dd6ec076bba0259d904b7ec8b0c',
  h''
)
```

```
message_1 (CBOR Sequence) (37 bytes)
01 00 58 20 89 8f f7 9a 02 06 7a 16 ea 1e cc b9 0f a5 22 46 f5 aa 4d d6
ec 07 6b ba 02 59 d9 04 b7 ec 8b 0c 40
```

B.1.2. Message_2

Since METHOD_CORR mod 4 equals 1, C_I is omitted from data_2.

```
Y (Responder's ephemeral private key) (32 bytes)
fd 8c d8 77 c9 ea 38 6e 6a f3 4f f7 e6 06 c4 b6 4c a8 31 c8 ba 33 13 4f
d4 cd 71 67 ca ba ec da

G_Y (Responder's ephemeral public key) (32 bytes)
71 a3 d5 99 c2 1d a1 89 02 a1 ae a8 10 b2 b6 38 2c cd 8d 5f 9b f0 19 52
81 75 4c 5e bc af 30 1e
```

From G_X and Y or from G_Y and X the ECDH shared secret is computed:

```
G_XY (ECDH shared secret) (32 bytes)
2b b7 fa 6e 13 5b c3 35 d0 22 d6 34 cb fb 14 b3 f5 82 f3 e2 e3 af b2 b3
15 04 91 49 5c 61 78 2b
```

The key and nonce for calculating the ciphertext are calculated as follows, as specified in Section 4.

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

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

Salt is the empty byte string.

salt (0 bytes)

From there, PRK_2e is computed:

PRK_2e (32 bytes)
ec 62 92 a0 67 f1 37 fc 7f 59 62 9d 22 6f bf c4 e0 68 89 49 f6 62 a9 7f
d8 2f be b7 99 71 39 4a

SK_R (Responders's private authentication key) (32 bytes)
df 69 27 4d 71 32 96 e2 46 30 63 65 37 2b 46 83 ce d5 38 1b fc ad cd 44
0a 24 c3 91 d2 fe db 94

Since neither the Initiator nor the Responder authenticates with a static Diffie-Hellman key, PRK_3e2m = PRK_2e

PRK_3e2m (32 bytes)
ec 62 92 a0 67 f1 37 fc 7f 59 62 9d 22 6f bf c4 e0 68 89 49 f6 62 a9 7f
d8 2f be b7 99 71 39 4a

The Responder chooses a connection identifier C_R.

Connection identifier chosen by Responder (1 byte)
2b

Note that since C_R is a byte string of length one, it is encoded as the corresponding integer subtracted by 24 (see bstr_identifier in Section 5.1). Thus 0x2b = 43, 43 - 24 = 19, and 19 in CBOR encoding is equal to 0x13.

C_R (1 byte)
13

Data_2 is constructed, as the CBOR Sequence of G_Y and C_R.

```
data_2 =
(
  h'71a3d599c21da18902a1aea810b2b6382ccd8d5f9bf0195281754c5ebcaf301e',
  h'13'
)
```

data_2 (CBOR Sequence) (35 bytes)
58 20 71 a3 d5 99 c2 1d a1 89 02 a1 ae a8 10 b2 b6 38 2c cd 8d 5f 9b f0
19 52 81 75 4c 5e bc af 30 1e 13

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

Input to calculate TH_2 (CBOR Sequence) (72 bytes)
01 00 58 20 89 8f f7 9a 02 06 7a 16 ea 1e cc b9 0f a5 22 46 f5 aa 4d d6
ec 07 6b ba 02 59 d9 04 b7 ec 8b 0c 40 58 20 71 a3 d5 99 c2 1d a1 89 02
a1 ae a8 10 b2 b6 38 2c cd 8d 5f 9b f0 19 52 81 75 4c 5e bc af 30 1e 13

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

TH_2 (32 bytes)
b0 dc 6c 1b a0 ba e6 e2 88 86 10 fa 0b 27 bf c5 2e 31 1a 47 b9 ca fb 60
9d e4 f6 a1 76 0d 6c f7

The Responder's subject name is the empty string:

Responders's subject name (text string)
""

CRED_R is the certificate (X509_R) encoded as a CBOR byte string:

X509_R (110 bytes)
47 62 4d c9 cd c6 82 4b 2a 4c 52 e9 5e c9 d6 b0 53 4b 71 c2 b4 9e 4b f9
03 15 00 ce e6 86 99 79 c2 97 bb 5a 8b 38 1e 98 db 71 41 08 41 5e 5c 50
db 78 97 4c 27 15 79 b0 16 33 a3 ef 62 71 be 5c 22 5e b2 8f 9c f6 18 0b
5a 6a f3 1e 80 20 9a 08 5c fb f9 5f 3f dc f9 b1 8b 69 3d 6c 0e 0d 0f fb
8e 3f 9a 32 a5 08 59 ec d0 bf cf f2 c2 18

CRED_R (112 bytes)
58 6e 47 62 4d c9 cd c6 82 4b 2a 4c 52 e9 5e c9 d6 b0 53 4b 71 c2 b4 9e
4b f9 03 15 00 ce e6 86 99 79 c2 97 bb 5a 8b 38 1e 98 db 71 41 08 41 5e
5c 50 db 78 97 4c 27 15 79 b0 16 33 a3 ef 62 71 be 5c 22 5e b2 8f 9c f6
18 0b 5a 6a f3 1e 80 20 9a 08 5c fb f9 5f 3f dc f9 b1 8b 69 3d 6c 0e 0d
0f fb 8e 3f 9a 32 a5 08 59 ec d0 bf cf f2 c2 18

And because certificates are identified by a hash value with the 'x5t' parameter, ID_CRED_R is the following:

ID_CRED_R = { 34 : COSE_CertHash }. In this example, the hash algorithm used is SHA-2 256-bit with hash truncated to 64-bits (value -15). The hash value is calculated over the certificate X509_R.

```
   ID_CRED_R =
   {
     34: [-15, h'FC79990F2431A3F5']
   }

   ID_CRED_R (14 bytes)
   a1 18 22 82 2e 48 fc 79 99 0f 24 31 a3 f5
```

Since no unprotected opaque auxiliary data is sent in the message exchanges:

AD_2 (0 bytes)

The Plaintext is defined as the empty string:

P_2m (0 bytes)

The Enc_structure is defined as follows: ["Encrypt0", << ID_CRED_R >>, << TH_2, CRED_R >>]

```
A_2m =
[
  "Encrypt0",
  h'A11822822E48FC79990F2431A3F5',
  h'5820B0DC6C1BA0BAE6E2888610FA0B27BFC52E311A47B9CAFB609DE4F6A1760D6CF
  7586E47624DC9CDC6824B2A4C52E95EC9D6B0534B71C2B49E4BF9031500CEE6869979
  C297BB5A8B381E98DB714108415E5C50DB78974C271579B01633A3EF6271BE5C225EB
  28F9CF6180B5A6AF31E80209A085CFBF95F3FDCF9B18B693D6C0E0D0FFB8E3F9A32A5
  0859ECD0BFCFF2C218'
  ]
```

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

```
A_2m (CBOR-encoded) (173 bytes)
83 68 45 6e 63 72 79 70 74 30 4e a1 18 22 82 2e 48 fc 79 99 0f 24 31 a3
f5 58 92 58 20 b0 dc 6c 1b a0 ba e6 e2 88 86 10 fa 0b 27 bf c5 2e 31 1a
47 b9 ca fb 60 9d e4 f6 a1 76 0d 6c f7 58 6e 47 62 4d c9 cd c6 82 4b 2a
4c 52 e9 5e c9 d6 b0 53 4b 71 c2 b4 9e 4b f9 03 15 00 ce e6 86 99 79 c2
97 bb 5a 8b 38 1e 98 db 71 41 08 41 5e 5c 50 db 78 97 4c 27 15 79 b0 16
33 a3 ef 62 71 be 5c 22 5e b2 8f 9c f6 18 0b 5a 6a f3 1e 80 20 9a 08 5c
fb f9 5f 3f dc f9 b1 8b 69 3d 6c 0e 0d 0f fb 8e 3f 9a 32 a5 08 59 ec d0
bf cf f2 c2 18
```

info for K_2m is defined as follows:

```
 info for K_2m =
 [
   10,
   h'B0DC6C1BA0BAE6E2888610FA0B27BFC52E311A47B9CAFB609DE4F6A1760D6CF7',
   "K_2m",
   16
 ]
```

Which as a CBOR encoded data item is:

```
info for K_2m (CBOR-encoded) (42 bytes)
84 0a 58 20 b0 dc 6c 1b a0 ba e6 e2 88 86 10 fa 0b 27 bf c5 2e 31 1a 47
b9 ca fb 60 9d e4 f6 a1 76 0d 6c f7 64 4b 5f 32 6d 10
```

From these parameters, K_2m is computed. Key K_2m is the output of HKDF-Expand(PRK_3e2m, info, L), where L is the length of K_2m, so 16 bytes.

```
K_2m (16 bytes)
b7 48 6a 94 a3 6c f6 9e 67 3f c4 57 55 ee 6b 95
```

info for IV_2m is defined as follows:

```
 info for IV_2m =
 [
   10,
   h'B0DC6C1BA0BAE6E2888610FA0B27BFC52E311A47B9CAFB609DE4F6A1760D6CF7',
   "IV_2m",
   13
 ]
```

Which as a CBOR encoded data item is:

```
info for IV_2m (CBOR-encoded) (43 bytes)
84 0a 58 20 b0 dc 6c 1b a0 ba e6 e2 88 86 10 fa 0b 27 bf c5 2e 31 1a 47
b9 ca fb 60 9d e4 f6 a1 76 0d 6c f7 65 49 56 5f 32 6d 0d
```

From these parameters, IV_2m is computed. IV_2m is the output of HKDF-Expand(PRK_3e2m, info, L), where L is the length of IV_2m, so 13 bytes.

```
IV_2m (13 bytes)
c5 b7 17 0e 65 d5 4f 1a e0 5d 10 af 56
```

Finally, COSE_Encrypt0 is computed from the parameters above.

- protected header = CBOR-encoded ID_CRED_R

   o external_aad = A_2m

   o empty plaintext = P_2m

   MAC_2 (8 bytes)
   cf 99 99 ae 75 9e c0 d8

   To compute the Signature_or_MAC_2, the key is the private
   authentication key of the Responder and the message M_2 to be signed
   = ["Signature1", << ID_CRED_R >>, << TH_2, CRED_R, ? AD_2 >>, MAC_2
   ]

```
M_2 =
[
  "Signature1",
  h'A11822822E48FC79990F2431A3F5',
  h'5820B0DC6C1BA0BAE6E2888610FA0B27BFC52E311A47B9CAFB609DE4F6A1760D6CF
  7586E47624DC9CDC6824B2A4C52E95EC9D6B0534B71C2B49E4BF9031500CEE6869979
  C297BB5A8B381E98DB714108415E5C50DB78974C271579B01633A3EF6271BE5C225EB
  28F9CF6180B5A6AF31E80209A085CFBF95F3FDCF9B18B693D6C0E0D0FFB8E3F9A32A5
  0859ECD0BFCFF2C218',
  h'CF9999AE759EC0D8'
]
```

   Which as a CBOR encoded data item is:

```
M_2 (184 bytes)
84 6a 53 69 67 6e 61 74 75 72 65 31 4e a1 18 22 82 2e 48 fc 79 99 0f 24
31 a3 f5 58 92 58 20 b0 dc 6c 1b a0 ba e6 e2 88 86 10 fa 0b 27 bf c5 2e
31 1a 47 b9 ca fb 60 9d e4 f6 a1 76 0d 6c f7 58 6e 47 62 4d c9 cd c6 82
4b 2a 4c 52 e9 5e c9 d6 b0 53 4b 71 c2 b4 9e 4b f9 03 15 00 ce e6 86 99
79 c2 97 bb 5a 8b 38 1e 98 db 71 41 08 41 5e 5c 50 db 78 97 4c 27 15 79
b0 16 33 a3 ef 62 71 be 5c 22 5e b2 8f 9c f6 18 0b 5a 6a f3 1e 80 20 9a
08 5c fb f9 5f 3f dc f9 b1 8b 69 3d 6c 0e 0d 0f fb 8e 3f 9a 32 a5 08 59
ec d0 bf cf f2 c2 18 48 cf 99 99 ae 75 9e c0 d8
```

   From there Signature_or_MAC_2 is a signature (since method = 0):

```
Signature_or_MAC_2 (64 bytes)
45 47 81 ec ef eb b4 83 e6 90 83 9d 57 83 8d fe 24 a8 cf 3f 66 42 8a a0
16 20 4a 22 61 84 4a f8 4f 98 b8 c6 83 4f 38 7f dd 60 6a 29 41 3a dd e3
a2 07 74 02 13 74 01 19 6f 6a 50 24 06 6f ac 0e
```

   CIPHERTEXT_2 is the ciphertext resulting from XOR encrypting a
   plaintext constructed from the following parameters and the key K_2e.

   o plaintext = CBOR Sequence of the items ID_CRED_R and
      Singature_or_MAC_2, in this order.

The plaintext is the following:

```
P_2e (CBOR Sequence) (80 bytes)
a1 18 22 82 2e 48 fc 79 99 0f 24 31 a3 f5 58 40 45 47 81 ec ef eb b4 83
e6 90 83 9d 57 83 8d fe 24 a8 cf 3f 66 42 8a a0 16 20 4a 22 61 84 4a f8
4f 98 b8 c6 83 4f 38 7f dd 60 6a 29 41 3a dd e3 a2 07 74 02 13 74 01 19
6f 6a 50 24 06 6f ac 0e
```

K_2e = HKDF-Expand(PRK, info, length), where length is the length of the plaintext, so 80.

```
info for K_2e =
[
  10,
  h'B0DC6C1BA0BAE6E2888610FA0B27BFC52E311A47B9CAFB609DE4F6A1760D6CF7',
  "K_2e",
  80
]
```

Which as a CBOR encoded data item is:

```
info for K_2e (CBOR-encoded) (43 bytes)
84 0a 58 20 b0 dc 6c 1b a0 ba e6 e2 88 86 10 fa 0b 27 bf c5 2e 31 1a 47
b9 ca fb 60 9d e4 f6 a1 76 0d 6c f7 64 4b 5f 32 65 18 50
```

From there, K_2e is computed:

```
K_2e (80 bytes)
38 cd 1a 83 89 6d 43 af 3d e8 39 35 27 42 0d ac 7d 7a 76 96 7e 85 74 58
26 bb 39 e1 76 21 8d 7e 5f e7 97 60 14 c9 ed ba c0 58 ee 18 cd 57 71 80
a4 4d de 0b 83 00 fe 8e 09 66 9a 34 d6 3e 3a e6 10 12 26 ab f8 5c eb 28
05 dc 00 13 d1 78 2a 20
```

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

```
CIPHERTEXT_2 (80 bytes)
99 d5 38 01 a7 25 bf d6 a4 e7 1d 04 84 b7 55 ec 38 3d f7 7a 91 6e c0 db
c0 2b ba 7c 21 a2 00 80 7b 4f 58 5f 72 8b 67 1a d6 78 a4 3a ac d3 3b 78
eb d5 66 cd 00 4f c6 f1 d4 06 f0 1d 97 04 e7 05 b2 15 52 a9 eb 28 ea 31
6a b6 50 37 d7 17 86 2e
```

message_2 is the CBOR Sequence of data_2 and CIPHERTEXT_2, in this order:

```
message_2 =
(
 data_2,
 h'99d53801a725bfd6a4e71d0484b755ec383df77a916ec0dbc02bba7c21a200807b4f
585f728b671ad678a43aacd33b78ebd566cd004fc6f1d406f01d9704e705b21552a9eb
28ea316ab65037d717862e'
)
```

Which as a CBOR encoded data item is:

```
message_2 (CBOR Sequence) (117 bytes)
58 20 71 a3 d5 99 c2 1d a1 89 02 a1 ae a8 10 b2 b6 38 2c cd 8d 5f 9b f0
19 52 81 75 4c 5e bc af 30 1e 13 58 50 99 d5 38 01 a7 25 bf d6 a4 e7 1d
04 84 b7 55 ec 38 3d f7 7a 91 6e c0 db c0 2b ba 7c 21 a2 00 80 7b 4f 58
5f 72 8b 67 1a d6 78 a4 3a ac d3 3b 78 eb d5 66 cd 00 4f c6 f1 d4 06 f0
1d 97 04 e7 05 b2 15 52 a9 eb 28 ea 31 6a b6 50 37 d7 17 86 2e
```

B.1.3. Message_3

Since corr equals 1, C_R is not omitted from data_3.

```
SK_I (Initiator's private authentication key) (32 bytes)
2f fc e7 a0 b2 b8 25 d3 97 d0 cb 54 f7 46 e3 da 3f 27 59 6e e0 6b 53 71
48 1d c0 e0 12 bc 34 d7
```

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

PRK_4x3m = HMAC-SHA-256 (PRK_3e2m, G_IY)

```
PRK_4x3m (32 bytes)
ec 62 92 a0 67 f1 37 fc 7f 59 62 9d 22 6f bf c4 e0 68 89 49 f6 62 a9 7f
d8 2f be b7 99 71 39 4a
```

data 3 is equal to C_R.

```
data_3 (CBOR Sequence) (1 bytes)
13
```

From data_3, CIPHERTEXT_2, and TH_2, compute the input to the transcript hash TH_3 = H(TH_2 , CIPHERTEXT_2, data_3), as a CBOR Sequence of these 3 data items.

```
Input to calculate TH_3 (CBOR Sequence) (117 bytes)
58 20 b0 dc 6c 1b a0 ba e6 e2 88 86 10 fa 0b 27 bf c5 2e 31 1a 47 b9 ca
fb 60 9d e4 f6 a1 76 0d 6c f7 58 50 99 d5 38 01 a7 25 bf d6 a4 e7 1d 04
84 b7 55 ec 38 3d f7 7a 91 6e c0 db c0 2b ba 7c 21 a2 00 80 7b 4f 58 5f
72 8b 67 1a d6 78 a4 3a ac d3 3b 78 eb d5 66 cd 00 4f c6 f1 d4 06 f0 1d
97 04 e7 05 b2 15 52 a9 eb 28 ea 31 6a b6 50 37 d7 17 86 2e 13
```

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

TH_3 (32 bytes)
a2 39 a6 27 ad a3 80 2d b8 da e5 1e c3 92 bf eb 92 6d 39 3e f6 ee e4 dd
b3 2e 4a 27 ce 93 58 da

The initiator's subject name is the empty string:

Initiator's subject name (text string)
""

CRED_I is the certificate (X509_I) encoded as a CBOR byte string:

X509_I (101 bytes)
fa 34 b2 2a 9c a4 a1 e1 29 24 ea e1 d1 76 60 88 09 84 49 cb 84 8f fc 79
5f 88 af c4 9c be 8a fd d1 ba 00 9f 21 67 5e 8f 6c 77 a4 a2 c3 01 95 60
1f 6f 0a 08 52 97 8b d4 3d 28 20 7d 44 48 65 02 ff 7b dd a6 32 c7 88 37
00 16 b8 96 5b db 20 74 bf f8 2e 5a 20 e0 9b ec 21 f8 40 6e 86 44 2b 87
ec 3f f2 45 b7

CRED_I (103 bytes)
58 65 fa 34 b2 2a 9c a4 a1 e1 29 24 ea e1 d1 76 60 88 09 84 49 cb 84 8f
fc 79 5f 88 af c4 9c be 8a fd d1 ba 00 9f 21 67 5e 8f 6c 77 a4 a2 c3 01
95 60 1f 6f 0a 08 52 97 8b d4 3d 28 20 7d 44 48 65 02 ff 7b dd a6 32 c7
88 37 00 16 b8 96 5b db 20 74 bf f8 2e 5a 20 e0 9b ec 21 f8 40 6e 86 44
2b 87 ec 3f f2 45 b7

And because certificates are identified by a hash value with the 'x5t' parameter, ID_CRED_I is the following:

ID_CRED_I = { 34 : COSE_CertHash }. In this example, the hash algorithm used is SHA-2 256-bit with hash truncated to 64-bits (value -15). The hash value is calculated over the certificate X509_I.

ID_CRED_I =
{
  34: [-15, h'FC79990F2431A3F5']
}

ID_CRED_I (14 bytes)
a1 18 22 82 2e 48 5b 78 69 88 43 9e bc f2

Since no opaque auxiliary data is exchanged:

AD_3 (0 bytes)

The Plaintext of the COSE_Encrypt is the empty string:

P_3m (0 bytes)

The external_aad is the CBOR Sequence od CRED_I and TH_3, in this order:

A_3m (CBOR-encoded) (164 bytes)
```
83 68 45 6e 63 72 79 70 74 30 4e a1 18 22 82 2e 48 5b 78 69 88 43 9e bc
f2 58 89 58 20 a2 39 a6 27 ad a3 80 2d b8 da e5 1e c3 92 bf eb 92 6d 39
3e f6 ee e4 dd b3 2e 4a 27 ce 93 58 da 58 65 fa 34 b2 2a 9c a4 a1 e1 29
24 ea e1 d1 76 60 88 09 84 49 cb 84 8f fc 79 5f 88 af c4 9c be 8a fd d1
ba 00 9f 21 67 5e 8f 6c 77 a4 a2 c3 01 95 60 1f 6f 0a 08 52 97 8b d4 3d
28 20 7d 44 48 65 02 ff 7b dd a6 32 c7 88 37 00 16 b8 96 5b db 20 74 bf
f8 2e 5a 20 e0 9b ec 21 f8 40 6e 86 44 2b 87 ec 3f f2 45 b7
```

Info for K_3m is computed as follows:

```
info for K_3m =
[
  10,
  h'A239A627ADA3802DB8DAE51EC392BFEB926D393EF6EEE4DDB32E4A27CE9358DA',
  "K_3m",
  16
]
```

Which as a CBOR encoded data item is:

info for K_3m (CBOR-encoded) (42 bytes)
```
84 0a 58 20 a2 39 a6 27 ad a3 80 2d b8 da e5 1e c3 92 bf eb 92 6d 39 3e
f6 ee e4 dd b3 2e 4a 27 ce 93 58 da 64 4b 5f 33 6d 10
```

From these parameters, K_3m is computed. Key K_3m is the output of HKDF-Expand(PRK_4x3m, info, L), where L is the length of K_2m, so 16 bytes.

K_3m (16 bytes)
```
3d bb f0 d6 01 03 26 e8 27 3f c6 c6 c3 b0 de cd
```

Nonce IV_3m is the output of HKDF-Expand(PRK_4x3m, info, L), where L = 13 bytes.

Info for IV_3m is defined as follows:

```
 info for IV_3m =
 [
   10,
   h'A239A627ADA3802DB8DAE51EC392BFEB926D393EF6EEE4DDB32E4A27CE9358DA',
   "IV_3m",
   13
 ]
```

Which as a CBOR encoded data item is:

```
info for IV_3m (CBOR-encoded) (43 bytes)
84 0a 58 20 a2 39 a6 27 ad a3 80 2d b8 da e5 1e c3 92 bf eb 92 6d 39 3e
f6 ee e4 dd b3 2e 4a 27 ce 93 58 da 65 49 56 5f 33 6d 0d
```

From these parameters, IV_3m is computed:

```
IV_3m (13 bytes)
10 b6 f4 41 4a 2c 91 3c cd a1 96 42 e3
```

MAC_3 is the ciphertext of the COSE_Encrypt0:

```
MAC_3 (8 bytes)
5e ef b8 85 98 3c 22 d9
```

Since the method = 0, Signature_or_Mac_3 is a signature:

- The message M_3 to be signed = ["Signature1", << ID_CRED_I >>, << TH_3, CRED_I >>, MAC_3]

- The signing key is the private authentication key of the Initiator.

```
M_3 =
[
  "Signature1",
  h'A11822822E485B786988439EBCF2',
  h'5820A239A627ADA3802DB8DAE51EC392BFEB926D393EF6EEE4DDB32E4A27CE9358D
  A5865FA34B22A9CA4A1E12924EAE1D1766088098449CB848FFC795F88AFC49CBE8AFD
  D1BA009F21675E8F6C77A4A2C30195601F6F0A0852978BD43D28207D44486502FF7BD
  DA632C788370016B8965BDB2074BFF82E5A20E09BEC21F8406E86442B87EC3FF245
  B7',
  h'5EEFB885983C22D9']
```

Which as a CBOR encoded data item is:

M_3 (175 bytes)
84 6a 53 69 67 6e 61 74 75 72 65 31 4e a1 18 22 82 2e 48 5b 78 69 88 43
9e bc f2 58 89 58 20 a2 39 a6 27 ad a3 80 2d b8 da e5 1e c3 92 bf eb 92
6d 39 3e f6 ee e4 dd b3 2e 4a 27 ce 93 58 da 58 65 fa 34 b2 2a 9c a4 a1
e1 29 24 ea e1 d1 76 60 88 09 84 49 cb 84 8f fc 79 5f 88 af c4 9c be 8a
fd d1 ba 00 9f 21 67 5e 8f 6c 77 a4 a2 c3 01 95 60 1f 6f 0a 08 52 97 8b
d4 3d 28 20 7d 44 48 65 02 ff 7b dd a6 32 c7 88 37 00 16 b8 96 5b db 20
74 bf f8 2e 5a 20 e0 9b ec 21 f8 40 6e 86 44 2b 87 ec 3f f2 45 b7 48 5e
ef b8 85 98 3c 22 d9

From there, the signature can be computed:

Signature_or_MAC_3 (64 bytes)
b3 31 76 33 fa eb c7 f4 24 9c f3 ab 95 96 fd ae 2b eb c8 e7 27 5d 39 9f
42 00 04 f3 76 7b 88 d6 0f fe 37 dc f3 90 a0 00 d8 5a b0 ad b0 d7 24 e3
a5 7c 4d fe 24 14 a4 1e 79 78 91 b9 55 35 89 06

Finally, the outer COSE_Encrypt0 is computed.

The Plaintext is the following CBOR Sequence: plaintext = (ID_CRED_I
, Signature_or_MAC_3)

P_3ae (CBOR Sequence) (80 bytes)
a1 18 22 82 2e 48 5b 78 69 88 43 9e bc f2 58 40 b3 31 76 33 fa eb c7 f4
24 9c f3 ab 95 96 fd ae 2b eb c8 e7 27 5d 39 9f 42 00 04 f3 76 7b 88 d6
0f fe 37 dc f3 90 a0 00 d8 5a b0 ad b0 d7 24 e3 a5 7c 4d fe 24 14 a4 1e
79 78 91 b9 55 35 89 06

The Associated data A is the following: Associated data A = [
"Encrypt0", h'', TH_3]

A_3ae (CBOR-encoded) (45 bytes)
83 68 45 6e 63 72 79 70 74 30 40 58 20 a2 39 a6 27 ad a3 80 2d b8 da e5
1e c3 92 bf eb 92 6d 39 3e f6 ee e4 dd b3 2e 4a 27 ce 93 58 da

Key K_3ae is the output of HKDF-Expand(PRK_3e2m, info, L).

info is defined as follows:

info for K_3ae =
[
  10,
  h'A239A627ADA3802DB8DAE51EC392BFEB926D393EF6EEE4DDB32E4A27CE9358DA',
  "K_3ae",
  16
]

Which as a CBOR encoded data item is:

```
 info for K_3ae (CBOR-encoded) (43 bytes)
 84 0a 58 20 a2 39 a6 27 ad a3 80 2d b8 da e5 1e c3 92 bf eb 92 6d 39 3e
 f6 ee e4 dd b3 2e 4a 27 ce 93 58 da 65 4b 5f 33 61 65 10
```

L is the length of K_3ae, so 16 bytes.

From these parameters, K_3ae is computed:

```
K_3ae (16 bytes)
58 b5 2f 94 5b 30 9d 85 4c a7 36 cd 06 a9 62 95
```

Nonce IV_3ae is the output of HKDF-Expand(PRK_3e2m, info, L).

info is defined as follows:

```
 info for IV_3ae =
 [
   10,
   h'A239A627ADA3802DB8DAE51EC392BFEB926D393EF6EEE4DDB32E4A27CE9358DA',
   "IV_3ae",
   13
 ]
```

Which as a CBOR encoded data item is:

```
 info for IV_3ae (CBOR-encoded) (44 bytes)
 84 0a 58 20 a2 39 a6 27 ad a3 80 2d b8 da e5 1e c3 92 bf eb 92 6d 39 3e
 f6 ee e4 dd b3 2e 4a 27 ce 93 58 da 66 49 56 5f 33 61 65 0d
```

L is the length of IV_3ae, so 13 bytes.

From these parameters, IV_3ae is computed:

```
IV_3ae (13 bytes)
cf a9 a5 85 58 10 d6 dc e9 74 3c 3b c3
```

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

```
 CIPHERTEXT_3 (88 bytes)
 2d 88 ff 86 da 47 48 2c 0d fa 55 9a c8 24 a4 a7 83 d8 70 c9 db a4 78 05
 e8 aa fb ad 69 74 c4 96 46 58 65 03 fa 9b bf 3e 00 01 2c 03 7e af 56 e4
 5e 30 19 20 83 9b 81 3a 53 f6 d4 c5 57 48 0f 6c 79 7d 5b 76 f0 e4 62 f5
 f5 7a 3d b6 d2 b5 0c 32 31 9f 34 0f 4a c5 af 9a
```

From the parameter above, message_3 is computed, as the CBOR Sequence of the following items: (C_R, CIPHERTEXT_3).

```
message_3 =
(
  h'13',
  h''
)
```

Which encodes to the following byte string:

```
message_3 (CBOR Sequence) (91 bytes)
13 58 58 2d 88 ff 86 da 47 48 2c 0d fa 55 9a c8 24 a4 a7 83 d8 70 c9 db
a4 78 05 e8 aa fb ad 69 74 c4 96 46 58 65 03 fa 9b bf 3e 00 01 2c 03 7e
af 56 e4 5e 30 19 20 83 9b 81 3a 53 f6 d4 c5 57 48 0f 6c 79 7d 5b 76 f0
e4 62 f5 f5 7a 3d b6 d2 b5 0c 32 31 9f 34 0f 4a c5 af 9a
```

B.2. Test Vectors for EDHOC Authenticated with Static Diffie-Hellman Keys

EDHOC with static Diffie-Hellman keys is used.

```
method (Static DH Based Authentication)
3
```

CoAP is used as transport and the Initiator acts as CoAP client:

```
corr (the Initiator can correlate message_1 and message_2)
1
```

From there, METHOD_CORR has the following value:

```
METHOD_CORR (4 * method + corr) (int)
13
```

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

The list of supported cipher suites of the Initiator in order of preference is the following:

```
Supported Cipher Suites (4 bytes)
00 01 02 03
```

The cipher suite selected by the Initiator is the most preferred:

```
Selected Cipher Suite (int)
0
```

The mandatory-to-implement cipher suite 0 is supported by both the Initiator and the Responder, see Section 8.3.

B.2.1. Message_1

```
X (Initiator's ephemeral private key) (32 bytes)
ae 11 a0 db 86 3c 02 27 e5 39 92 fe b8 f5 92 4c 50 d0 a7 ba 6e ea b4 ad
1f f2 45 72 f4 f5 7c fa

G_X (Initiator's ephemeral public key) (32 bytes)
8d 3e f5 6d 1b 75 0a 43 51 d6 8a c2 50 a0 e8 83 79 0e fc 80 a5 38 a4 44
ee 9e 2b 57 e2 44 1a 7c
```

The Initiator chooses a connection identifier C_I:

```
Connection identifier chosen by Initiator (1 bytes)
16
```

Note that since C_I is a byte strings of length one, it is encoded as the corresponding integer - 24 (see bstr_identifier in Section 5.1), i.e. 0x16 = 22, 22 - 24 = -2, and -2 in CBOR encoding is equal to 0x21.

```
C_I (1 byte)
21
```

Since no unprotected opaque auxiliary data is sent in the message exchanges:

```
AD_1 (0 bytes)
```

Since the list of supported cipher suites needs to contain the selected cipher suite, the initiator truncates the list of supported cipher suites to one cipher suite only, 00.

Because one single selected cipher suite is conveyed, it is encoded as an int instead of an array:

```
SUITES_I (int)
0
```

With SUITES_I = 0, message_1 is constructed, as the CBOR Sequence of the CBOR data items above.

```
message_1 =
(
  13,
  0,
  h'8D3EF56D1B750A4351D68AC250A0E883790EFC80A538A444EE9E2B57E2441A7C',
  -2
)
```

message_1 (CBOR Sequence) (37 bytes)
0d 00 58 20 8d 3e f5 6d 1b 75 0a 43 51 d6 8a c2 50 a0 e8 83 79 0e fc 80
a5 38 a4 44 ee 9e 2b 57 e2 44 1a 7c 21

B.2.2. Message_2

Since METHOD_CORR mod 4 equals 1, C_I is omitted from data_2.

Y (Responder's ephemeral private key) (32 bytes)
c6 46 cd dc 58 12 6e 18 10 5f 01 ce 35 05 6e 5e bc 35 f4 d4 cc 51 07 49
a3 a5 e0 69 c1 16 16 9a

G_Y (Responder's ephemeral public key) (32 bytes)
52 fb a0 bd c8 d9 53 dd 86 ce 1a b2 fd 7c 05 a4 65 8c 7c 30 af db fc 33
01 04 70 69 45 1b af 35

From G_X and Y or from G_Y and X the ECDH shared secret is computed:

G_XY (ECDH shared secret) (32 bytes)
de fc 2f 35 69 10 9b 3d 1f a4 a7 3d c5 e2 fe b9 e1 15 0d 90 c2 5e e2 f0
66 c2 d8 85 f4 f8 ac 4e

The key and nonce for calculating the ciphertext are calculated as follows, as specified in Section 4.

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

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

Salt is the empty byte string.

salt (0 bytes)

From there, PRK_2e is computed:

PRK_2e (32 bytes)
93 9f cb 05 6d 2e 41 4f 1b ec 61 04 61 99 c2 c7 63 d2 7f 0c 3d 15 fa 16
71 fa 13 4e 0d c5 a0 4d

Since the Responder authenticates with a static Diffie-Hellman key, PRK_3e2m = HKDF-Extract(PRK_2e, G_RX), where G_RX is the ECDH shared secret calculated from G_R and X, or G_X and R.

R (Responder's private authentication key) (32 bytes)
bb 50 1a ac 67 b9 a9 5f 97 e0 ed ed 6b 82 a6 62 93 4f bb fc 7a d1 b7 4c
1f ca d6 6a 07 94 22 d0

G_R (Responder's public authentication key) (32 bytes)
a3 ff 26 35 95 be b3 77 d1 a0 ce 1d 04 da d2 d4 09 66 ac 6b cb 62 20 51
b8 46 59 18 4d 5d 9a 32

From the Responder's authentication key and the Initiator's ephemeral key (see Appendix B.2.1), the ECDH shared secret G_RX is calculated.

G_RX (ECDH shared secret) (32 bytes)
21 c7 ef f4 fb 69 fa 4b 67 97 d0 58 84 31 5d 84 11 a3 fd a5 4f 6d ad a6
1d 4f cd 85 e7 90 66 68

PRK_3e2m (32 bytes)
75 07 7c 69 1e 35 01 2d 48 bc 24 c8 4f 2b ab 89 f5 2f ac 03 fe dd 81 3e
43 8c 93 b1 0b 39 93 07

The Responder chooses a connection identifier C_R.

Connection identifier chosen by Responder (1 byte)
20

Note that since C_R is a byte strings of length one, it is encoded as the corresponding integer - 24 (see bstr_identifier in Section 5.1), i.e. 0x20 = 32, 32 - 24 = 8, and 8 in CBOR encoding is equal to 0x08.

C_R (1 byte)
08

Data_2 is constructed, as the CBOR Sequence of G_Y and C_R.

```
data_2 =
(
  h'52FBA0BDC8D953DD86CE1AB2FD7C05A4658C7C30AFDBFC3301047069451BAF35',
  08
)
```

data_2 (CBOR Sequence) (35 bytes)
58 20 52 fb a0 bd c8 d9 53 dd 86 ce 1a b2 fd 7c 05 a4 65 8c 7c 30 af db
fc 33 01 04 70 69 45 1b af 35 08

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

Input to calculate TH_2 (CBOR Sequence) (72 bytes)
0d 00 58 20 8d 3e f5 6d 1b 75 0a 43 51 d6 8a c2 50 a0 e8 83 79 0e fc 80
a5 38 a4 44 ee 9e 2b 57 e2 44 1a 7c 21 58 20 52 fb a0 bd c8 d9 53 dd 86
ce 1a b2 fd 7c 05 a4 65 8c 7c 30 af db fc 33 01 04 70 69 45 1b af 35 08

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

 TH_2 (32 bytes)
 6a 28 78 e8 4b 2c c0 21 cc 1a eb a2 96 52 53 ef 42 f7 fa 30 0c af 9c 49
 1a 52 e6 83 6a 25 64 ff

   The Responder's subject name is the empty string:

   Responders's subject name (text string)
   ""

   ID_CRED_R is the following:

   ID_CRED_R =
   {
     4: h'07'
   }

   ID_CRED_R (4 bytes)
   a1 04 41 07

   CRED_R is the following COSE_Key:

```
{
  1: 1,
  -1: 4,
  -2: h'A3FF263595BEB377D1A0CE1D04DAD2D40966AC6BCB622051B84659184D5D9A32',
  "subject name": ""
}
```

   Which encodes to the following byte string:

 CRED_R (54 bytes)
 a4 01 01 20 04 21 58 20 a3 ff 26 35 95 be b3 77 d1 a0 ce 1d 04 da d2 d4
 09 66 ac 6b cb 62 20 51 b8 46 59 18 4d 5d 9a 32 6c 73 75 62 6a 65 63 74
 20 6e 61 6d 65 60

   Since no unprotected opaque auxiliary data is sent in the message
   exchanges:

   AD_2 (0 bytes)

   The Plaintext is defined as the empty string:

   P_2m (0 bytes)

The Enc_structure is defined as follows: ["Encrypt0",
<< ID_CRED_R >>, << TH_2, CRED_R >>]

```
A_2m =
[
  "Encrypt0",
  h'A1044107',

h'58206A2878E84B2CC021CC1AEBA2965253EF42F7FA300CAF9C491A52E6836A2564FFA401012004215820A3FF263595
]
```

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

```
A_2m (CBOR-encoded) (105 bytes)
83 68 45 6e 63 72 79 70 74 30 44 a1 04 41 07 58 58 58 20 6a 28 78 e8 4b
2c c0 21 cc 1a eb a2 96 52 53 ef 42 f7 fa 30 0c af 9c 49 1a 52 e6 83 6a
25 64 ff a4 01 01 20 04 21 58 20 a3 ff 26 35 95 be b3 77 d1 a0 ce 1d 04
da d2 d4 09 66 ac 6b cb 62 20 51 b8 46 59 18 4d 5d 9a 32 6c 73 75 62 6a
65 63 74 20 6e 61 6d 65 60
```

info for K_2m is defined as follows:

```
info for K_2m =
[
  10,
  h'6A2878E84B2CC021CC1AEBA2965253EF42F7FA300CAF9C491A52E6836A2564FF',
  "K_2m",
  16
]
```

Which as a CBOR encoded data item is:

```
info for K_2m (CBOR-encoded) (42 bytes)
84 0a 58 20 6a 28 78 e8 4b 2c c0 21 cc 1a eb a2 96 52 53 ef 42 f7 fa 30
0c af 9c 49 1a 52 e6 83 6a 25 64 ff 64 4b 5f 32 6d 10
```

From these parameters, K_2m is computed. Key K_2m is the output of HKDF-Expand(PRK_3e2m, info, L), where L is the length of K_2m, so 16 bytes.

```
K_2m (16 bytes)
81 2a 48 87 d1 90 ff ed 2b 10 0b a7 a5 c2 5e 67
```

info for IV_2m is defined as follows:

```
 info for IV_2m =
 [
   10,
   h'6A2878E84B2CC021CC1AEBA2965253EF42F7FA300CAF9C491A52E6836A2564FF',
   "IV_2m",
   13
 ]
```

Which as a CBOR encoded data item is:

```
info for IV_2m (CBOR-encoded) (43 bytes)
84 0a 58 20 6a 28 78 e8 4b 2c c0 21 cc 1a eb a2 96 52 53 ef 42 f7 fa 30
0c af 9c 49 1a 52 e6 83 6a 25 64 ff 65 49 56 5f 32 6d 0d
```

From these parameters, IV_2m is computed. IV_2m is the output of HKDF-Expand(PRK_3e2m, info, L), where L is the length of IV_2m, so 13 bytes.

```
IV_2m (13 bytes)
92 3c 0f 94 31 51 5b 69 21 30 49 2b 7f
```

Finally, COSE_Encrypt0 is computed from the parameters above.

- protected header = CBOR-encoded ID_CRED_R

- external_aad = A_2m

- empty plaintext = P_2m

```
MAC_2 (8 bytes)
64 21 0d 2e 18 b9 28 cd
```

From there Signature_or_MAC_2 is the MAC (since method = 3):

```
Signature_or_MAC_2 (8 bytes)
64 21 0d 2e 18 b9 28 cd
```

CIPHERTEXT_2 is the ciphertext resulting from XOR encrypting a plaintext constructed from the following parameters and the key K_2e.

- plaintext = CBOR Sequence of the items ID_CRED_R and the CBOR encoded Signature_or_MAC_2, in this order. Note that since ID_CRED_R contains a single 'kid' parameter, i.e., ID_CRED_R = { 4 : kid_R }, only the byte string kid_R is conveyed in the plaintext encoded as a bstr_identifier. kid_R is encoded as the corresponding integer - 24 (see bstr_identifier in Section 5.1), i.e. 0x07 = 7, 7 - 24 = -17, and -17 in CBOR encoding is equal to 0x30.

The plaintext is the following:

P_2e (CBOR Sequence) (10 bytes)
30 48 64 21 0d 2e 18 b9 28 cd

K_2e = HKDF-Expand(PRK, info, length), where length is the length of the plaintext, so 10.

```
info for K_2e =
[
  10,
  h'6A2878E84B2CC021CC1AEBA2965253EF42F7FA300CAF9C491A52E6836A2564FF',
  "K_2e",
  10
]
```

Which as a CBOR encoded data item is:

```
info for K_2e (CBOR-encoded) (42 bytes)
84 0a 58 20 6a 28 78 e8 4b 2c c0 21 cc 1a eb a2 96 52 53 ef 42 f7 fa 30
0c af 9c 49 1a 52 e6 83 6a 25 64 ff 64 4b 5f 32 65 0a
```

From there, K_2e is computed:

K_2e (10 bytes)
ec be 9a bd 5f 62 3a fc 65 26

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

CIPHERTEXT_2 (10 bytes)
dc f6 fe 9c 52 4c 22 45 4d eb

message_2 is the CBOR Sequence of data_2 and CIPHERTEXT_2, in this order:

```
message_2 =
(
 data_2,
 h'DCF6FE9C524C22454DEB'
)
```

Which as a CBOR encoded data item is:

```
message_2 (CBOR Sequence) (46 bytes)
58 20 52 fb a0 bd c8 d9 53 dd 86 ce 1a b2 fd 7c 05 a4 65 8c 7c 30 af db
fc 33 01 04 70 69 45 1b af 35 08 4a dc f6 fe 9c 52 4c 22 45 4d eb
```

B.2.3. Message_3

Since corr equals 1, C_R is not omitted from data_3.

SK_I (Initiator's private authentication key) (32 bytes)
2b be a6 55 c2 33 71 c3 29 cf bd 3b 1f 02 c6 c0 62 03 38 37 b8 b5 90 99
a4 43 6f 66 60 81 b0 8e

G_I (Initiator's public authentication key) (32 bytes)
2c 44 0c c1 21 f8 d7 f2 4c 3b 0e 41 ae da fe 9c aa 4f 4e 7a bb 83 5e c3
0f 1d e8 8a db 96 ff 71

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

From the Initiator's authentication key and the Responder's ephemeral key (see Appendix B.2.2), the ECDH shared secret G_IY is calculated.

G_IY (ECDH shared secret) (32 bytes)
cb ff 8c d3 4a 81 df ec 4c b6 5d 9a 57 2e bd 09 64 45 0c 78 56 3d a4 98
1d 80 d3 6c 8b 1a 75 2a

PRK_4x3m = HMAC-SHA-256 (PRK_3e2m, G_IY).

PRK_4x3m (32 bytes)
ec 62 92 a0 67 f1 37 fc 7f 59 62 9d 22 6f bf c4 e0 68 89 49 f6 62 a9 7f
d8 2f be b7 99 71 39 4a

data 3 is equal to C_R.

data_3 (CBOR Sequence) (1 bytes)
08

From data_3, CIPHERTEXT_2, and TH_2, compute the input to the transcript hash TH_3 = H(TH_2 , CIPHERTEXT_2, data_3), as a CBOR Sequence of these 3 data items.

Input to calculate TH_3 (CBOR Sequence) (46 bytes)
58 20 6a 28 78 e8 4b 2c c0 21 cc 1a eb a2 96 52 53 ef 42 f7 fa 30 0c af
9c 49 1a 52 e6 83 6a 25 64 ff 4a dc f6 fe 9c 52 4c 22 45 4d eb 08

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

TH_3 (32 bytes)
51 dd 22 43 a6 b8 3f 13 16 dc 53 29 1a e1 91 cd 93 b4 44 cc e4 80 16 07
03 ee d9 c4 a1 bc b6 11

The initiator's subject name is the empty string:

   Initiator's subject name (text string)
   ""

   And its credential is:

   ID_CRED_I =
   {
     4: h'24'
   }

   ID_CRED_I (4 bytes)
   a1 04 41 24

   CRED_I is the following COSE_Key:

```
{
  1: 1,
  -1: 4,
   -2: h'2C440CC121F8D7F24C3B0E41AEDAFE9CAA4F4E7ABB835EC30F1DE88ADB96FF71',
   "subject name": ""
 }
```

   Which encodes to the following byte string:

```
 CRED_I (54 bytes)
 a4 01 01 20 04 21 58 20 2c 44 0c c1 21 f8 d7 f2 4c 3b 0e 41 ae da fe 9c
 aa 4f 4e 7a bb 83 5e c3 0f 1d e8 8a db 96 ff 71 6c 73 75 62 6a 65 63 74
 20 6e 61 6d 65 60
```

   Since no opaque auxiliary data is exchanged:

   AD_3 (0 bytes)

   The Plaintext of the COSE_Encrypt is the empty string:

   P_3m (0 bytes)

   The external_aad is the CBOR Sequence of CRED_I and TH_3, in this
   order:

```
 A_3m (CBOR-encoded) (105 bytes)
 83 68 45 6e 63 72 79 70 74 30 44 a1 04 41 24 58 58 58 20 51 dd 22 43 a6
 b8 3f 13 16 dc 53 29 1a e1 91 cd 93 b4 44 cc e4 80 16 07 03 ee d9 c4 a1
 bc b6 11 a4 01 01 20 04 21 58 20 2c 44 0c c1 21 f8 d7 f2 4c 3b 0e 41 ae
 da fe 9c aa 4f 4e 7a bb 83 5e c3 0f 1d e8 8a db 96 ff 71 6c 73 75 62 6a
 65 63 74 20 6e 61 6d 65 60
```

   Info for K_3m is computed as follows:

```
 info for K_3m =
 [
   10,
   h'51DD2243A6B83F1316DC53291AE191CD93B444CCE480160703EED9C4A1BCB611',
   "K_3m",
   16
 ]
```

Which as a CBOR encoded data item is:

```
info for K_3m (CBOR-encoded) (42 bytes)
84 0a 58 20 51 dd 22 43 a6 b8 3f 13 16 dc 53 29 1a e1 91 cd 93 b4 44 cc
e4 80 16 07 03 ee d9 c4 a1 bc b6 11 64 4b 5f 33 6d 10
```

From these parameters, K_3m is computed. Key K_3m is the output of HKDF-Expand(PRK_4x3m, info, L), where L is the length of K_2m, so 16 bytes.

K_3m (16 bytes)

```
84 85 31 8a a3 08 6f d5 86 7a 02 8e 99 e2 40 30
```

Nonce IV_3m is the output of HKDF-Expand(PRK_4x3m, info, L), where L = 13 bytes.

Info for IV_3m is defined as follows:

```
 info for IV_3m =
 [
   10,
   h'51DD2243A6B83F1316DC53291AE191CD93B444CCE480160703EED9C4A1BCB611',
   "IV_3m",
   13
 ]
```

Which as a CBOR encoded data item is:

```
info for IV_3m (CBOR-encoded) (43 bytes)
84 0a 58 20 51 dd 22 43 a6 b8 3f 13 16 dc 53 29 1a e1 91 cd 93 b4 44 cc
e4 80 16 07 03 ee d9 c4 a1 bc b6 11 65 49 56 5f 33 6d 0d
```

From these parameters, IV_3m is computed:

IV_3m (13 bytes)

```
1e 10 5b 88 50 0e d5 ae b0 5d 00 6b ea
```

MAC_3 is the ciphertext of the COSE_Encrypt0:

MAC_3 (8 bytes)
1f b7 5a c1 aa d2 34 25

Since the method = 3, Signature_or_Mac_3 is the MAC_3:

Signature_or_MAC_3 (8 bytes)
1f b7 5a c1 aa d2 34 25

Finally, the outer COSE_Encrypt0 is computed.

The Plaintext is the following CBOR Sequence: plaintext = (ID_CRED_I , Signature_or_MAC_3). Note that since ID_CRED_I contains a single 'kid' parameter, i.e., ID_CRED_I = { 4 : kid_I }, only the byte string kid_I is conveyed in the plaintext encoded as a bstr_identifier. kid_I is encoded as the corresponding integer - 24 (see bstr_identifier in Section 5.1), i.e. 0x24 = 36, 36 - 24 = 12, and 12 in CBOR encoding is equal to 0x0c.

P_3ae (CBOR Sequence) (10 bytes)
0c 48 1f b7 5a c1 aa d2 34 25

The Associated data A is the following: Associated data A = ["Encrypt0", h'', TH_3]

A_3ae (CBOR-encoded) (45 bytes)
83 68 45 6e 63 72 79 70 74 30 40 58 20 51 dd 22 43 a6 b8 3f 13 16 dc 53 29 1a e1 91 cd 93 b4 44 cc e4 80 16 07 03 ee d9 c4 a1 bc b6 11

Key K_3ae is the output of HKDF-Expand(PRK_3e2m, info, L).

info is defined as follows:

info for K_3ae =
[
  10,
  h'51DD2243A6B83F1316DC53291AE191CD93B444CCE480160703EED9C4A1BCB611',
  "K_3ae",
  16
]

Which as a CBOR encoded data item is:

info for K_3ae (CBOR-encoded) (43 bytes)
84 0a 58 20 51 dd 22 43 a6 b8 3f 13 16 dc 53 29 1a e1 91 cd 93 b4 44 cc e4 80 16 07 03 ee d9 c4 a1 bc b6 11 65 4b 5f 33 61 65 10

L is the length of K_3ae, so 16 bytes.

From these parameters, K_3ae is computed:

K_3ae (16 bytes)
bf 29 0b 7e e0 4b 86 5d e1 01 0a 81 1b 36 00 64

Nonce IV_3ae is the output of HKDF-Expand(PRK_3e2m, info, L).

info is defined as follows:

```
info for IV_3ae =
[
  10,
  h'51DD2243A6B83F1316DC53291AE191CD93B444CCE480160703EED9C4A1BCB611',
  "IV_3ae",
  13
]
```

Which as a CBOR encoded data item is:

```
info for IV_3ae (CBOR-encoded) (44 bytes)
84 0a 58 20 51 dd 22 43 a6 b8 3f 13 16 dc 53 29 1a e1 91 cd 93 b4 44 cc
e4 80 16 07 03 ee d9 c4 a1 bc b6 11 66 49 56 5f 33 61 65 0d
```

L is the length of IV_3ae, so 13 bytes.

From these parameters, IV_3ae is computed:

IV_3ae (13 bytes)
0e 74 45 0a fc ec e9 73 af 64 e9 4d 46

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

CIPHERTEXT_3 (18 bytes)
53 c3 99 19 99 a5 ff b8 69 21 e9 9b 60 7c 06 77 70 e0

From the parameter above, message_3 is computed, as the CBOR Sequence of the following items: (C_R, CIPHERTEXT_3).

```
message_3 =
(
  h'08',
  h'53C3991999A5FFB86921E99B607C067770E0'
)
```

Which encodes to the following byte string:

message_3 (CBOR Sequence) (20 bytes)
08 52 53 c3 99 19 99 a5 ff b8 69 21 e9 9b 60 7c 06 77 70 e0

Appendix C. Applicability Statement Template

EDHOC requires certain parameters to be agreed upon between Initiator and Responder. A cipher suite is negotiated with the protocol, but certain other parameters need to be agreed beforehand:

1. Method and correlation of underlying transport messages (METHOD_CORR, see Section 3.2.1 and Section 3.2.4).

2. Type of authentication credentials (CRED_I, CRED_R, see Section 3.3.4).

3. Type for identifying authentication credentials (ID_CRED_I, ID_CRED_R, see Section 3.3.4).

4. Type and use of Auxiliary Data AD_1, AD_2, AD_3 (see Section 3.6).

5. Identifier used as identity of endpoint (see Section 3.3).

An example of an applicability statement is shown in the next section.

Note that for some of the parameters, like METHOD_CORR, ID_CRED_x, type of AD_x, the receiver is able to assert whether it supports the parameter or not and thus, if it fails, to infer why.

For other parameters, like type of authentication credential, it may be more difficult to detect if the receiver got the wrong type since the credential is not necessarily transported, and a failed integrity of the received message may be caused by other circumstances. For example in the case of public key certificates there is a large variety of profiles and alternative encodings, which the applicability statement needs to nail down.

Note also that it is not always necessary for the endpoints to agree on the transport for the EDHOC messages. For example, a mix of CoAP and HTTP may be used along the path and still allow correlation between message_1 and message_2.

C.1. Use of EDHOC in the XX Protocol

For use of EDHOC in the XX protocol, the following assumptions are made on the parameters.

- METHOD_CORR = 5
  - method = 1 (I uses signature key, R uses static DH key.)
  - corr = 1 (CoAP Token or other transport data enables correlation between message_1 and message_2.)
- CRED_I is an 802.1AR IDevID encoded as a CBOR Certificate of type 0
  - R acquires CRED_I out-of-band, indicated in AD_1
  - ID_CRED_I = {4: h''} is a kid with value empty byte string
- CRED_R is a COSE_Key of type OKP as specified in Section 3.3.4.
  - The CBOR map has parameters 1 (kty), -1 (crv), and -2 (x-coordinate).
- ID_CRED_R = CRED_R
- AD_1 contains Auxiliary Data of type A (TBD)
- AD_2 contains Auxiliary Data of type B (TBD)

Auxiliary Data is processed as specified in [I-D.ietf-ace-oauth-authz].

- Need to specify use of C_I/C_R ? (TBD)

Acknowledgments

The authors want to thank Alessandro Bruni, Karthikeyan Bhargavan, 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, Vaishnavi Sundararajan, Erik Thormarker, and Michel Veillette 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 Preuss Mattsson
Ericsson AB

Email: john.mattsson@ericsson.com

Francesca Palombini
Ericsson AB

Email: francesca.palombini@ericsson.com