Stream: Internet Engineering Task Force (IETF)
RFC: 9528
Category: Standards Track
Published: March 2024
ISSN: 2070-1721
Authors: G. Selander *Ericsson* J. Preuß Mattsson *Ericsson* F. Palombini *Ericsson*

RFC 9528
Ephemeral Diffie-Hellman Over COSE (EDHOC)

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, forward secrecy, and identity protection. EDHOC is intended for usage in constrained scenarios, and a main use case is to establish an Object Security for Constrained RESTful Environments (OSCORE) security context. By reusing CBOR Object Signing and Encryption (COSE) for cryptography, Concise Binary Object Representation (CBOR) for encoding, and Constrained Application Protocol (CoAP) for transport, the additional code size can be kept very low.

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc9528.

Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions

with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

Table of Contents

1. Introduction 5
1.1. Motivation 5
1.2. Message Size Examples 6
1.3. Document Structure 7
1.4. Terminology and Requirements Language 7
2. EDHOC Outline 7
3. Protocol Elements 9
3.1. General 9
3.2. Method 10
3.3. Connection Identifiers 11
3.4. Transport 12
3.5. Authentication Parameters 14
3.6. Cipher Suites 18
3.7. Ephemeral Public Keys 20
3.8. External Authorization Data (EAD) 20
3.9. Application Profile 21
4. Key Derivation 23
4.1. Keys for EDHOC Message Processing 23
4.2. Keys for EDHOC Applications 26
5. Message Formatting and Processing 27
5.1. EDHOC Message Processing Outline 27
5.2. EDHOC Message 1 28
5.3. EDHOC Message 2 29
5.4. EDHOC Message 3 31
5.5. EDHOC Message 4 33

6. Error Handling 35
- 6.1. Success 36
- 6.2. Unspecified Error 36
- 6.3. Wrong Selected Cipher Suite 36
- 6.4. Unknown Credential Referenced 38

7. EDHOC Message Deduplication 39

8. Compliance Requirements 39

9. Security Considerations 40
- 9.1. Security Properties 40
- 9.2. Cryptographic Considerations 43
- 9.3. Cipher Suites and Cryptographic Algorithms 44
- 9.4. Post-Quantum Considerations 44
- 9.5. Unprotected Data and Privacy 45
- 9.6. Updated Internet Threat Model Considerations 45
- 9.7. Denial of Service 46
- 9.8. Implementation Considerations 46

10. IANA Considerations 48
- 10.1. EDHOC Exporter Label Registry 49
- 10.2. EDHOC Cipher Suites Registry 49
- 10.3. EDHOC Method Type Registry 51
- 10.4. EDHOC Error Codes Registry 51
- 10.5. EDHOC External Authorization Data Registry 52
- 10.6. COSE Header Parameters Registry 52
- 10.7. Well-Known URI Registry 53
- 10.8. Media Types Registry 53
- 10.9. CoAP Content-Formats Registry 55
- 10.10. Resource Type (rt=) Link Target Attribute Values Registry 55
- 10.11. Expert Review Instructions 55

11. References 56
- 11.1. Normative References 56

- 11.2. Informative References 58
- Appendix A. Use with OSCORE and Transfer over CoAP 62
  - A.1. Deriving the OSCORE Security Context 62
  - A.2. Transferring EDHOC over CoAP 63
- Appendix B. Compact Representation 66
- Appendix C. Use of CBOR, CDDL, and COSE in EDHOC 68
  - C.1. CBOR and CDDL 68
  - C.2. CDDL Definitions 69
  - C.3. COSE 70
- Appendix D. Authentication-Related Verifications 72
  - D.1. Validating the Authentication Credential 72
  - D.2. Identities 72
  - D.3. Certification Path and Trust Anchors 73
  - D.4. Revocation Status 74
  - D.5. Unauthenticated Operation 74
- Appendix E. Use of External Authorization Data 74
- Appendix F. Application Profile Example 75
- Appendix G. Long PLAINTEXT_2 76
- Appendix H. EDHOC_KeyUpdate 77
- Appendix I. Example Protocol State Machine 78
  - I.1. Initiator State Machine 78
  - I.2. Responder State Machine 79
- Acknowledgments 81
- Authors' Addresses 82

1. Introduction

1.1. Motivation

Many Internet of Things (IoT) deployments require technologies that are highly performant in constrained environments [RFC7228]. IoT devices may be constrained in various ways, including memory, storage, processing capacity, and power. The connectivity for these settings may also exhibit constraints, such as unreliable and lossy channels, highly restricted bandwidth, and dynamic topology. The IETF has acknowledged this problem by standardizing a range of lightweight protocols and enablers designed for the IoT, including CoAP [RFC7252], CBOR [RFC8949], and Static Context Header Compression (SCHC) [RFC8724].

The need for special protocols targeting constrained IoT deployments extends also to the security domain [LAKE-REQS]. Important characteristics in constrained environments are the number of round trips and protocol message sizes, which (if kept low) can contribute to good performance by enabling transport over a small number of radio frames, reducing latency due to fragmentation, duty cycles, etc. Another important criterion is code size, which may be prohibitively large for certain deployments due to device capabilities or network load during firmware updates. Some IoT deployments also need to support a variety of underlying transport technologies, potentially even with a single connection.

Some security solutions for such settings exist already. COSE [RFC9052] specifies basic application-layer security services efficiently encoded in CBOR. Another example is OSCORE [RFC8613], which is a lightweight communication security extension to CoAP using CBOR and COSE. In order to establish good quality cryptographic keys for security protocols such as COSE and OSCORE, the two endpoints may run an authenticated Diffie-Hellman key exchange protocol, from which shared secret keying material can be derived. Such a key exchange protocol should also be lightweight to prevent bad performance in case of repeated use, e.g., due to device rebooting or frequent rekeying for security reasons or to avoid latencies in a network formation setting with many devices authenticating at the same time.

This document specifies Ephemeral Diffie-Hellman Over COSE (EDHOC), a lightweight authenticated key exchange protocol providing good security properties including forward secrecy, identity protection, and cipher suite negotiation. Authentication can be based on raw public keys (RPKs) or public key certificates and requires the application to provide input on how to verify that endpoints are trusted. This specification supports the referencing of credentials in order to reduce message overhead, but credentials may alternatively be embedded in the messages. EDHOC does not currently support Pre-Shared Key (PSK) authentication as authentication with static Diffie-Hellman (DH) public keys by reference produces equally small message sizes but with much simpler key distribution and identity protection.

EDHOC makes use of known protocol constructions, such as SIGn-and-MAc [SIGMA], the Noise XX pattern [Noise], and Extract-and-Expand [RFC5869]. EDHOC uses COSE for cryptography and identification of credentials (including COSE_Key, CBOR Web Token (CWT), CWT Claims Set (CCS), X.509, and CBOR-encoded X.509 (C509) certificates; see Section 3.5.2). COSE provides crypto agility and enables the use of future algorithms and credential types targeting IoT.

EDHOC is designed for highly constrained settings, making it especially suitable for low-power networks [RFC8376] such as Cellular IoT, IPv6 over the TSCH mode of IEEE 802.15.4e (6TiSCH), and LoRaWAN. A main objective for EDHOC is to be a lightweight authenticated key exchange for OSCORE, i.e., to provide authentication and session key establishment for IoT use cases such as those built on CoAP [RFC7252] involving 'things' with embedded microcontrollers, sensors, and actuators. By reusing the same lightweight primitives as OSCORE (CBOR, COSE, and CoAP), the additional code size can be kept very low. Note that while CBOR and COSE primitives are built into the protocol messages, EDHOC is not bound to a particular transport.

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). 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 connects for the first time or to establish fresh keys that are not revealed by a later compromise of the long-term keys.

1.2. Message Size Examples

Examples of EDHOC message sizes are shown in Table 1, which use different kinds of authentication keys and COSE header parameters for identification, including static Diffie-Hellman keys or signature keys, either in CWT/CCS [RFC8392] identified by a key identifier using 'kid' [RFC9052] or in X.509 certificates identified by a hash value using 'x5t' [RFC9360]. EDHOC always uses ephemeral-ephemeral key exchange. As a comparison, in the case of RPK authentication and when transferred in CoAP, the EDHOC message size can be less than 1/7 of the DTLS 1.3 handshake [RFC9147] with Ephemeral Elliptic Curve Diffie-Hellman (ECDHE) and connection ID; see [CoAP-SEC-PROT].

	Static DH Keys		Signature Keys	
	kid	**x5t**	**kid**	**x5t**
message_1	37	37	37	37
message_2	45	58	102	115
message_3	19	33	77	90
Total	101	128	216	242

Table 1: Examples of EDHOC Message Sizes in Bytes

1.3. Document Structure

The remainder of the document is organized as follows: Section 2 outlines EDHOC authenticated with signature keys; Section 3 describes the protocol elements of EDHOC, including formatting of the ephemeral public keys; Section 4 specifies the key derivation; Section 5 specifies message processing for EDHOC authenticated with signature keys or static Diffie-Hellman keys; Section 6 describes the error messages; Section 7 describes EDHOC support for transport that does not handle message duplication; and Section 8 lists compliance requirements. Note that normative text is also used in appendices, in particular Appendix A.

1.4. 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 Structures and Processing [RFC9052], COSE Algorithms [RFC9053], CWT and CCS [RFC8392], and the Concise Data Definition Language (CDDL) [RFC8610], which is used to express CBOR data structures. Examples of CBOR and CDDL are provided in Appendix C.1. When referring to CBOR, this specification always refers to Deterministically Encoded CBOR, as specified in Sections 4.2.1 and 4.2.2 of [RFC8949]. The single output from authenticated encryption (including the authentication tag) is called "ciphertext", following [RFC5116].

2. EDHOC Outline

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

SIGn-and-MAc (SIGMA) is a family of theoretical protocols with a number of variants [SIGMA]. Like in Internet Key Exchange Protocol Version 2 (IKEv2) [RFC7296] and (D)TLS 1.3 [RFC8446] [RFC9147], EDHOC authenticated with signature keys is built on a variant of the SIGMA protocol, SIGMA-I, which provides identity protection against active attacks on the party initiating the protocol. Also like IKEv2, EDHOC implements the MAC-then-Sign variant of the SIGMA-I protocol. The message flow (excluding an optional fourth message) is shown in Figure 1.

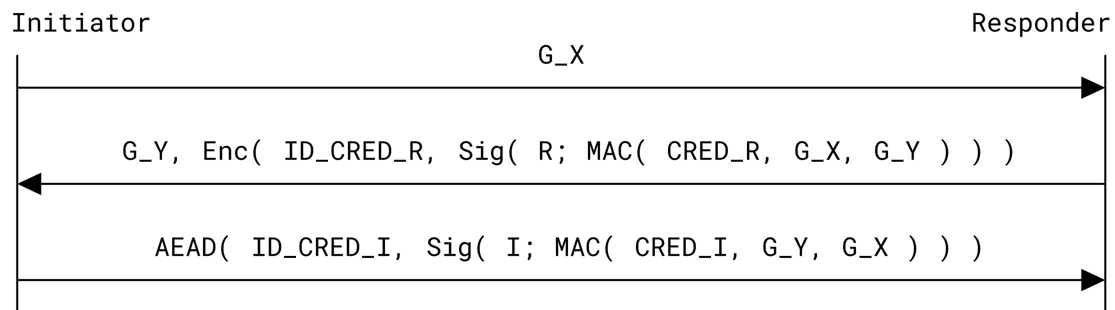

Figure 1: MAC-then-Sign Variant of the SIGMA-I Protocol Used by the EDHOC Method 0

The parties exchanging messages in an EDHOC session are called the Initiator (I) and the Responder (R), where the Initiator sends message_1 (see Section 3). They exchange ephemeral public keys, compute a shared secret session key PRK_out, and derive symmetric application keys used to protect application data.

- G_X and G_Y are the Elliptic Curve Diffie-Hellman (ECDH) ephemeral public keys of I and R, respectively.
- CRED_I and CRED_R are the authentication credentials containing the public authentication keys of I and R, respectively.
- ID_CRED_I and ID_CRED_R are used to identify and optionally transport the credentials of I and R, respectively.
- Sig(I; .) and Sig(R; .) denote signatures made with the private authentication key of I and R, respectively.
- Enc(), AEAD(), and MAC() denote encryption, Authenticated Encryption with Associated Data, and Message Authentication Code -- crypto algorithms applied with keys derived from one or more shared secrets calculated during the protocol.

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

- transcript hashes (hashes of message data), TH_2, TH_3, and 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,
- an optional fourth message giving key confirmation to I in deployments where no protected application data is sent from R to I,
- a keying material exporter and a key update function with forward secrecy,
- secure negotiation of the cipher suite,
- method types, error handling, and padding,
- the selection of connection identifiers, C_I and C_R, which may be used in EDHOC to identify the protocol state, and

- transport of external authorization data.

EDHOC is designed to encrypt and integrity protect as much information as possible. Symmetric keys and random material used in EDHOC are derived using EDHOC_KDF with as much previous information as possible; see Figure 6. 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. Like in (D)TLS, authentication is the responsibility of the application. EDHOC identifies (and optionally transports) authentication credentials and provides proof-of-possession of the private authentication key.

To simplify for implementors, the use of CBOR, CDDL, and COSE in EDHOC is summarized in Appendix C. Test vectors, including CBOR diagnostic notation, are provided in [RFC9529].

3. Protocol Elements

3.1. General

The EDHOC protocol consists of three mandatory messages (message_1, message_2, and message_3), an optional fourth message (message_4), and an error message, between an Initiator (I) and a Responder (R). The odd messages are sent by I, the even by R. Both I and R can send error messages. The roles have slightly different security properties that should be considered when the roles are assigned; see Section 9.1. All EDHOC messages are CBOR Sequences [RFC8742] and are defined to be deterministically encoded CBOR as specified in Section 4.2.1 of [RFC8949]. Figure 2 illustrates an EDHOC message flow with the optional fourth message as well as the content of each message. The protocol elements in the figure are introduced in Sections 3 and 5. Message formatting and processing are specified in Sections 5 and 6.

Application data may be protected using the agreed application algorithms (AEAD, hash) in the selected cipher suite (see Section 3.6), and the application can make use of the established connection identifiers C_I and C_R (see Section 3.3). Media types that may be used for EDHOC are defined in Section 10.8.

The Initiator can derive symmetric application keys after creating EDHOC message_3; see Section 4.2.1. Protected application data can therefore be sent in parallel or together with EDHOC message_3. EDHOC message_4 is typically not sent.

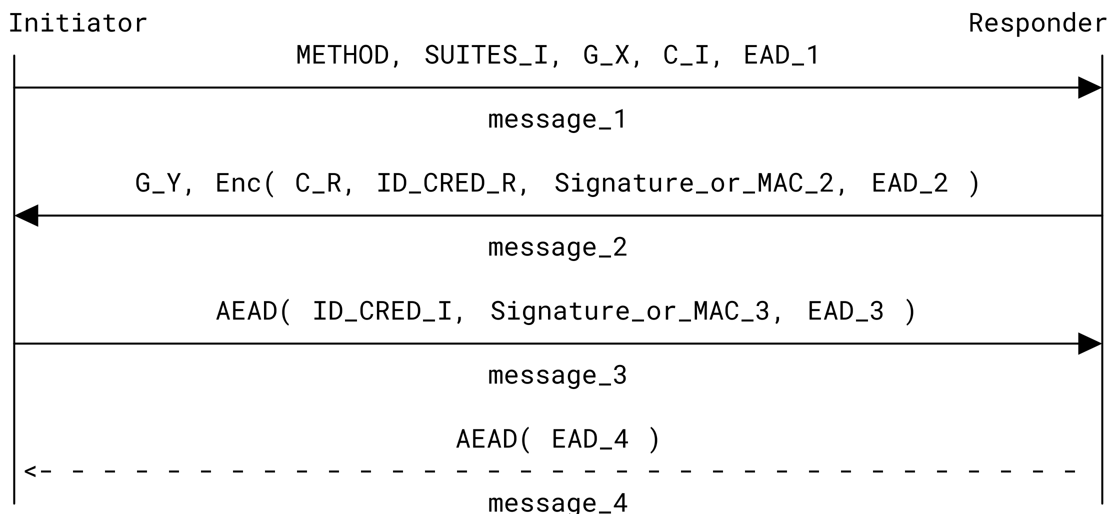

Figure 2: EDHOC Message Flow Including the Optional Fourth Message

3.2. Method

The data item METHOD in message_1 (see Section 5.2.1) is an integer specifying the authentication method. EDHOC currently supports authentication with signature or static Diffie-Hellman keys, as defined in the four authentication methods: 0, 1, 2, and 3; see Table 2. 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 that enables significant reductions in message sizes. Note that, also in the static Diffie-Hellman-based authentication methods, there is an ephemeral-ephemeral Diffie-Hellman key exchange.

The Initiator and Responder need to have agreed on a single method to be used for EDHOC; see Section 3.9.

Method Type Value	Initiator Authentication Key	Responder Authentication Key
0	Signature Key	Signature Key
1	Signature Key	Static DH Key
2	Static DH Key	Signature Key
3	Static DH Key	Static DH Key
23	Reserved	Reserved

Table 2: Authentication Keys for Method Types

EDHOC does not have a dedicated message field to indicate the protocol version. Breaking changes to EDHOC can be introduced by specifying and registering new methods.

3.3. Connection Identifiers

EDHOC includes the selection of connection identifiers (C_I and C_R) identifying a connection for which keys are agreed.

Connection identifiers may be used to correlate EDHOC messages and facilitate the retrieval of protocol state during an EDHOC session (see Section 3.4) or may be used in applications of EDHOC, e.g., in OSCORE (see Section 3.3.3). The connection identifiers do not have any cryptographic purpose in EDHOC and only facilitate the retrieval of security data associated with the protocol state.

Connection identifiers in EDHOC are intrinsically byte strings. Most constrained devices only have a few connections for which short identifiers may be sufficient. In some cases, minimum length identifiers are necessary to comply with overhead requirements. However, CBOR byte strings -- with the exception of the empty byte string h'', which encodes as one byte (0x40) -- are encoded as two or more bytes. To enable one-byte encoding of certain byte strings while maintaining CBOR encoding, EDHOC represents certain identifiers as CBOR integers on the wire; see Section 3.3.2.

3.3.1. Selection of Connection Identifiers

C_I and C_R are chosen by I and R, respectively. The Initiator selects C_I and sends it in message_1 for the Responder to use as a reference to the connection in communications with the Initiator. The Responder selects C_R and sends it in message_2 for the Initiator to use as a reference to the connection in communications with the Responder.

If connection identifiers are used by an application protocol for which EDHOC establishes keys, then the selected connection identifiers **SHALL** adhere to the requirements for that protocol; see Section 3.3.3 for an example.

3.3.2. Representation of Byte String Identifiers

To allow identifiers with minimal overhead on the wire, certain byte strings used in connection identifiers and credential identifiers (see Section 3.5.3) are defined to have integer representations.

The integers with one-byte CBOR encoding are -24, ..., 23; see Figure 3.

```
    Integer:  -24  -23  ... -11  ...  -2   -1    0    1  ...  15  ...  23
    Encoding:  37   36  ...  2A  ...  21   20   00   01  ...  0F  ...  17
```

Figure 3: One-Byte CBOR-Encoded Integers

The byte strings that coincide with a one-byte CBOR encoding of an integer **MUST** be represented by the CBOR encoding of that integer. Other byte strings are simply encoded as CBOR byte strings.

For example:

- 0x21 is represented by 0x21 (CBOR encoding of the integer -2), not by 0x4121 (CBOR encoding of the byte string 0x21).
- 0x0D is represented by 0x0D (CBOR encoding of the integer 13), not by 0x410D (CBOR encoding of the byte string 0x0D).
- 0x18 is represented by 0x4118 (CBOR encoding of the byte string 0x18).
- 0x38 is represented by 0x4138 (CBOR encoding of the byte string 0x38).
- 0xABCD is represented by 0x42ABCD (CBOR encoding of the byte string 0xABCD).

One may view this representation of byte strings as a transport encoding, i.e., a byte string that parses as the one-byte CBOR encoding of an integer (i.e., integer in the interval -24, ..., 23) is just copied directly into the message, and a byte string that does not is encoded as a CBOR byte string during transport.

> Implementation Note: When implementing the byte string identifier representation, in some programming languages, it can help to define a new type or other data structure, which (in its user-facing API) behaves like a byte string but when serializing to CBOR produces a CBOR byte string or a CBOR integer depending on its value.

3.3.3. Use of Connection Identifiers with OSCORE

For OSCORE, the choice of connection identifier results in the endpoint selecting its Recipient ID (see Section 3.1 of [RFC8613]) for which certain uniqueness requirements apply (see Section 3.3 of [RFC8613]). Therefore, the Initiator and Responder **MUST NOT** select connection identifiers such that it results in the same OSCORE Recipient ID. Since the connection identifier is a byte string, it is converted to an OSCORE Recipient ID equal to the byte string.

Examples:

- A connection identifier 0xFF (represented in the EDHOC message as 0x41FF; see Section 3.3.2) is converted to the OSCORE Recipient ID 0xFF.
- A connection identifier 0x21 (represented in the EDHOC message as 0x21; see Section 3.3.2) is converted to the OSCORE Recipient ID 0x21.

3.4. Transport

Cryptographically, EDHOC does not put requirements on the underlying layers. Received messages are processed as the expected next message according to the protocol state; see Section 5. If processing fails for any reason, then typically an error message is attempted to be sent and the EDHOC session is aborted.

EDHOC is not bound to a particular transport layer and can even be used in environments without IP. Ultimately, the application is free to choose how to transport EDHOC messages including errors. In order to avoid unnecessary message processing or protocol termination, it is **RECOMMENDED** to use reliable transport, such as CoAP in reliable mode, which is the default transport; see Appendix A.2. In general, the transport **SHOULD** handle:

- message loss,
- message duplication (see Section 7 for an alternative),
- flow control,
- congestion control,
- fragmentation and reassembly,
- demultiplexing EDHOC messages from other types of messages,
- denial-of-service mitigation, and
- message correlation (see Section 3.4.1).

EDHOC does not require error-free transport since a change in message content is detected through the transcript hashes in a subsequent integrity verification; see Section 5. The transport does not require additional means to handle message reordering because of the lockstep processing of EDHOC.

EDHOC is designed to enable an authenticated key exchange with small messages, where the minimum message sizes are of the order illustrated in the first column of Table 1. There is no maximum message size specified by the protocol; for example, this is dependent on the size of the authentication credentials (if they are transported, see Section 3.5). The encryption of very large content in message_2 when using certain hash algorithms is described in Appendix G.

The use of transport is specified in the application profile, which in particular, may specify limitations in message sizes; see Section 3.9.

3.4.1. EDHOC Message Correlation

Correlation between EDHOC messages is needed to facilitate the retrieval of the protocol state and security context during an EDHOC session. It is also helpful for the Responder to get an indication that a received EDHOC message is the beginning of a new EDHOC session, such that no existing protocol state or security context needs to be retrieved.

Correlation may be based on existing mechanisms in the transport protocol; for example, the CoAP Token may be used to correlate EDHOC messages in a CoAP response and in an associated CoAP request. The connection identifiers may also be used to correlate EDHOC messages.

If correlation between consecutive messages is not provided by other means, then the transport binding **SHOULD** mandate prepending of an appropriate connection identifier (when available from the EDHOC protocol) to the EDHOC message. If message_1 indication is not provided by other means, then the transport binding **SHOULD** mandate prepending of message_1 with the CBOR simple value `true` (0xf5).

Transport of EDHOC in CoAP payloads is described in Appendix A.2, including how to use connection identifiers and message_1 indication with CoAP. A similar construction is possible for other client-server protocols. Protocols that do not provide any correlation at all can prescribe prepending of the peer's connection identifier to all messages.

Note that correlation between EDHOC messages may be obtained without transport support or connection identifiers, for example, if the endpoints only accept a single instance of the protocol at a time and execute conditionally on a correct sequence of messages.

3.5. Authentication Parameters

EDHOC supports various settings for how the other endpoint's public key for authentication may be transported, identified, and trusted. We shall use the term "authentication key" to mean key used for authentication in general, or specifically, the public key, when there is no risk for confusion.

EDHOC performs the following authentication-related operations:

- EDHOC transports information about credentials in ID_CRED_I and ID_CRED_R (described in Section 3.5.3). Based on this information, the authentication credentials CRED_I and CRED_R (described in Section 3.5.2) can be obtained. EDHOC may also transport certain authentication-related information as external authorization data (see Section 3.8).
- EDHOC uses the authentication credentials in two ways (see Sections 5.3.2 and 5.4.2):
  - The authentication credential is input to the integrity verification using the MAC fields.
  - The authentication key of the authentication credential is used with the Signature_or_MAC field to verify proof-of-possession of the private key.

Other authentication-related verifications are out of scope for EDHOC and are the responsibility of the application. In particular, the authentication credential needs to be validated in the context of the connection for which EDHOC is used; see Appendix D. EDHOC **MUST** allow the application to read received information about credentials in ID_CRED_R and ID_CRED_I. EDHOC **MUST** have access to the authentication key and the authentication credential.

Note that the type of authentication key, the type of authentication credential, and the identification of the credential have a large impact on the message size. For example, the Signature_or_MAC field is much smaller with a static DH key than with a signature key. A CWT Claims Set (CCS) is much smaller than a self-signed certificate / CWT, but if it is possible to reference the credential with a COSE header like 'kid', then that is in turn much smaller than a CCS.

3.5.1. Authentication Keys

The authentication key **MUST** be a signature key or a static Diffie-Hellman key. The Initiator and Responder **MAY** use different types of authentication keys, e.g., one uses a signature key and the other uses a static Diffie-Hellman key.

The authentication key algorithm needs to be compatible with the method and the selected cipher suite (see Section 3.6). The authentication key algorithm needs to be compatible with the EDHOC key exchange algorithm when static Diffie-Hellman authentication is used and compatible with the EDHOC signature algorithm when signature authentication is used.

Note that for most signature algorithms, the signature is determined jointly by the signature algorithm and the authentication key algorithm. When using static Diffie-Hellman keys, the Initiator's and the Responder's private authentication keys are denoted as I and R, respectively, and the public authentication keys are denoted G_I and G_R, respectively.

For X.509 certificates, the authentication key is represented by a SubjectPublicKeyInfo field, which also contains information about authentication key algorithm. For CWT and CCS (see Section 3.5.2), the authentication key is represented by a 'cnf' claim [RFC8747] containing a COSE_Key [RFC9052], which contains information about authentication key algorithm. In EDHOC, a raw public key (RPK) is an authentication key encoded as a COSE_Key wrapped in a CCS, an example is given in Figure 4.

3.5.2. Authentication Credentials

The authentication credentials, CRED_I and CRED_R, contain the public authentication key of the Initiator and Responder, respectively. We use the notation CRED_x to refer to CRED_I or CRED_R. Requirements on CRED_x applies both to CRED_I and to CRED_R. The authentication credential typically also contains other parameters that needs to be verified by the application (see Appendix D) and in particular information about the identity ("subject") of the endpoint to prevent misbinding attacks (see Appendix D.2).

EDHOC relies on COSE for identification of credentials (see Section 3.5.3), for example, X.509 certificates [RFC9360], C509 certificates [C509-CERTS], CWTs [RFC8392], and CCSs [RFC8392]. When the identified credential is a chain or a bag, the authentication credential CRED_x is just the end entity X.509 or C509 certificate / CWT. In the choice between a chain or a bag, it is **RECOMMENDED** to use a chain, since the certificates in a bag are unordered and may contain self-signed and extraneous certificates, which can add complexity to the process of extracting the end entity certificate. The Initiator and Responder **MAY** use different types of authentication credentials, e.g., one uses an RPK and the other uses a public key certificate.

Since CRED_R is used in the integrity verification (see Section 5.3.2), it needs to be specified such that it is identical when used by the Initiator or Responder. Similarly for CRED_I, see Section 5.4.2. The Initiator and Responder are expected to agree on the specific encoding of the authentication credentials; see Section 3.9. It is **RECOMMENDED** that the COSE 'kid' parameter, when used to identify the authentication credential, refers to such a specific encoding of the authentication credential. The Initiator and Responder **SHOULD** use an available authentication credential without re-encoding, i.e. an authentication credential transported in EDHOC by value, or otherwise provisioned, **SHOULD** be used as is. If for some reason re-encoding of an

authentication credential passed by reference may occur, then a potential common encoding for CBOR-based credentials is deterministically encoded CBOR, as specified in Sections 4.2.1 and 4.2.2 of [RFC8949].

- When the authentication credential is an X.509 certificate, CRED_x **SHALL** be the DER-encoded certificate, encoded as a bstr [RFC9360].
- When the authentication credential is a C509 certificate, CRED_x **SHALL** be the C509 certificate [C509-CERTS].
- When the authentication credential is a CWT including a COSE_Key, CRED_x **SHALL** be the untagged CWT.
- When the authentication credential includes a COSE_Key but is not in a CWT, CRED_x **SHALL** be an untagged CCS. This is how RPKs are encoded, see Figure 4 for an example.
  - Naked COSE_Keys are thus dressed as CCS when used in EDHOC, in its simplest form by prefixing the COSE_Key with 0xA108A101 (a map with a 'cnf' claim). In that case, the resulting authentication credential contains no other identity than the public key itself; see Appendix D.2.

An example of CRED_x is shown below:

```
{                                              /CCS/
  2 : "42-50-31-FF-EF-37-32-39",               /sub/
  8 : {                                        /cnf/
    1 : {                                      /COSE_Key/
      1 : 1,                                   /kty/
      2 : h'00',                               /kid/
     -1 : 4,                                   /crv/
     -2 : h'b1a3e89460e88d3a8d54211dc95f0b90   /x/
            3ff205eb71912d6db8f4af980d2db83a'
    }
  }
}
```

Figure 4: CCS Containing an X25519 Static Diffie-Hellman Key and an EUI-64 Identity

3.5.3. Identification of Credentials

The ID_CRED fields, ID_CRED_R and ID_CRED_I, are transported in message_2 and message_3, respectively; see Sections 5.3.2 and 5.4.2. We use the notation ID_CRED_x to refer to ID_CRED_I or ID_CRED_R. Requirements on ID_CRED_x applies both to ID_CRED_I and to ID_CRED_R. The ID_CRED fields are used to identify and optionally transport credentials:

- ID_CRED_R is intended to facilitate for the Initiator retrieving the authentication credential CRED_R and the authentication key of R.
- ID_CRED_I is intended to facilitate for the Responder retrieving the authentication credential CRED_I and the authentication key of I.

ID_CRED_x may contain the authentication credential CRED_x, for x = I or R, but for many settings, it is not necessary to transport the authentication credential within EDHOC. For example, it may be pre-provisioned or acquired out-of-band over less constrained links. ID_CRED_I and ID_CRED_R do not have any cryptographic purpose in EDHOC since the authentication credentials are integrity protected by the Signature_or_MAC field.

EDHOC relies on COSE for identification of credentials and supports all credential types for which COSE header parameters are defined, including X.509 certificates [RFC9360], C509 certificates [C509-CERTS], CWTs (Section 3.5.3.1) and CCSs (Section 3.5.3.1).

ID_CRED_I and ID_CRED_R are of type COSE header_map, as defined in Section 3 of [RFC9052], and contain one or more COSE header parameters. If a map contains several header parameters, the labels do not need to be sorted in bytewise lexicographic order. ID_CRED_I and ID_CRED_R **MAY** contain different header parameters. The header parameters typically provide some information about the format of the credential.

Example: X.509 certificates can be identified by a hash value using the 'x5t' parameter; see Section 2 of [RFC9360]:

- ID_CRED_x = { 34 : COSE_CertHash }, for x = I or R

Example: CWT or CCS can be identified by a key identifier using the 'kid' parameter; see Section 3.1 of [RFC9052]:

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

Note that COSE header parameters in ID_CRED_x are used to identify the message sender's credential. Therefore, there is no reason to use the "-sender" header parameters, such as x5t-sender, defined in Section 3 of [RFC9360]. Instead, the corresponding parameter without "-sender", such as x5t, **SHOULD** be used.

As stated in Section 3.1 of [RFC9052], applications **MUST NOT** assume that 'kid' values are unique and several keys associated with a 'kid' may need to be checked before the correct one is found. Applications might use additional information such as 'kid context' or lower layers to determine which key to try first. Applications should strive to make ID_CRED_x as unique as possible, since the recipient may otherwise have to try several keys.

See Appendix C.3 for more examples.

3.5.3.1. COSE Header Parameters for CWT and CWT Claims Set

This document registers two new COSE header parameters, 'kcwt' and 'kccs', for use with CBOR Web Token (CWT) [RFC8392] and CWT Claims Set (CCS) [RFC8392], respectively. The CWT/CCS **MUST** contain a COSE_Key in a 'cnf' claim [RFC8747]. There may be any number of additional claims present in the CWT/CCS.

CWTs sent in 'kcwt' are protected using a MAC or a signature and are similar to a certificate (when used with public key cryptography) or a Kerberos ticket (when used with symmetric key cryptography). CCSs sent in 'kccs' are not protected and are therefore similar to raw public keys or self-signed certificates.

Security considerations for 'kcwt' and 'kccs' are made in Section 9.8.

3.5.3.2. Compact Encoding of ID_CRED Fields for 'kid'

To comply with the Lightweight Authenticated Key Exchange (LAKE) message size requirements (see [LAKE-REQS]), two optimizations are made for the case when ID_CRED_x, for x = I or R, contains a single 'kid' parameter.

1. The CBOR map { 4 : kid_x } is replaced by the byte string kid_x.
2. The representation of identifiers specified in Section 3.3.2 is applied to kid_x.

These optimizations **MUST** be applied if and only if ID_CRED_x = { 4 : kid_x } and ID_CRED_x in PLAINTEXT_y of message_y, y = 2 or 3; see Sections 5.3.2 and 5.4.2. Note that these optimizations are not applied to instances of ID_CRED_x that have no impact on message size, e.g., context_y, or the COSE protected header. For example:

- For ID_CRED_x = { 4 : h'FF' }, the encoding in PLAINTEXT_y is not the CBOR map 0xA10441FF but the CBOR byte string h'FF', i.e., 0x41FF.
- For ID_CRED_x = { 4 : h'21' }, the encoding in PLAINTEXT_y is neither the CBOR map 0xA1044121 nor the CBOR byte string h'21', i.e., 0x4121, but the CBOR integer 0x21.

3.6. Cipher Suites

An EDHOC cipher suite consists of an ordered set of algorithms from the "COSE Algorithms" and "COSE Elliptic Curves" registries as well as the EDHOC MAC length. All algorithm names and definitions follow COSE Algorithms [RFC9053]. Note that COSE sometimes uses peculiar names such as ES256 for Elliptic Curve Digital Signature Algorithm (ECDSA) with SHA-256, A128 for AES-128, and Ed25519 for the curve edwards25519. Algorithms need to be specified with enough parameters to make them completely determined. The EDHOC MAC length **MUST** be at least 8 bytes. Any cryptographic algorithm used in the COSE header parameters in ID_CRED fields is selected independently of the selected cipher suite. EDHOC is currently only specified for use with key exchange algorithms of type ECDH curves, but any Key Encapsulation Mechanism (KEM), including Post-Quantum Cryptography (PQC) KEMs, can be used in method 0; see Section 9.4. Use of other types of key exchange algorithms to replace static DH authentication (methods 1, 2, and 3) would likely require a specification updating EDHOC with new methods.

EDHOC supports all signature algorithms defined by COSE. Just like in (D)TLS 1.3 [RFC8446] [RFC9147] and IKEv2 [RFC7296], a signature in COSE is determined jointly by the signature algorithm and the authentication key algorithm; see Section 3.5.1. The exact details of the authentication key algorithm depend on the type of authentication credential. COSE supports

different formats for storing the public authentication keys including COSE_Key and X.509, which use different names and ways to represent the authentication key and the authentication key algorithm.

An EDHOC cipher suite consists of the following parameters:

- EDHOC AEAD algorithm,
- EDHOC hash algorithm,
- EDHOC MAC length in bytes (Static DH),
- EDHOC key exchange algorithm (ECDH curve),
- EDHOC signature algorithm,
- application AEAD algorithm, and
- application hash algorithm.

Each cipher suite is identified with a predefined integer label.

EDHOC can be used with all algorithms and curves defined for COSE. Implementations can either use any combination of COSE algorithms and parameters to define their own private cipher suite or use one of the predefined cipher suites. Private cipher suites can be identified with any of the four values: -24, -23, -22, and -21. The predefined cipher suites are listed in the IANA registry (Section 10.2) with the initial content outlined here:

- Cipher suites 0-3, based on AES-CCM, are intended for constrained IoT where message overhead is a very important factor. Note that AES-CCM-16-64-128 and AES-CCM-16-128-128 are compatible with the IEEE AES-CCM* mode of operation defined in Annex B of [IEEE.802.15.4-2015].
    - Cipher suites 1 and 3 use a larger tag length (128 bits) in EDHOC than in the application AEAD algorithm (64 bits).
- Cipher suites 4 and 5, based on ChaCha20, are intended for less constrained applications and only use 128-bit tag lengths.
- Cipher suite 6, based on AES-GCM, is for general non-constrained applications. It consists of high-performance algorithms that are widely used in non-constrained applications.
- Cipher suites 24 and 25 are intended for high security applications such as government use and financial applications. These cipher suites do not share any algorithms. Cipher suite 24 consists of algorithms from the Commercial National Security Algorithm (CNSA) 1.0 suite [CNSA].

The different methods (Section 3.2) use the same cipher suites, but some algorithms are not used in some methods. The EDHOC signature algorithm is 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 contains cipher suites supported by the Initiator and formatted and processed as detailed in Section 5.2.1 to secure the cipher suite negotiation. Examples of cipher suite negotiation are given in Section 6.3.2.

3.7. Ephemeral Public Keys

The ephemeral public keys in EDHOC (G_X and G_Y) use compact representation of elliptic curve points; see Appendix B. In COSE, compact representation is achieved by formatting the ECDH ephemeral public keys as COSE_Keys of type EC2 or Octet Key Pair (OKP) according to Sections 7.1 and 7.2 of [RFC9053] but only including the 'x' parameter in G_X and G_Y. For Elliptic Curve Keys of type EC2, compact representation **MAY** be used also in the COSE_Key. COSE always uses compact output for Elliptic Curve Keys of type EC2. If the COSE implementation requires a 'y' parameter, the value y = false or a calculated y-coordinate can be used; see Appendix B.

3.8. External Authorization Data (EAD)

In order to reduce round trips and the number of messages or to simplify processing, external security applications may be integrated into EDHOC by transporting authorization-related data in the messages.

EDHOC allows processing of external authorization data (EAD) to be defined in a separate specification and sent in dedicated fields of the four EDHOC messages: EAD_1, EAD_2, EAD_3, and EAD_4. EAD is opaque data to EDHOC.

Each EAD field, EAD_x, for x = 1, 2, 3, or 4, is a CBOR sequence (see Appendix C.1) consisting of one or more EAD items. EAD item ead is a CBOR sequence of an ead_label and an optional ead_value; see Figure 5 and Appendix C.2 for the CDDL definitions.

```
ead = (
  ead_label : int,
  ? ead_value : bstr,
)
```

Figure 5: EAD Item

A security application may register one or more EAD labels (see Section 10.5) and specify the associated processing and security considerations. The IANA registry contains the absolute value of the ead_label, |ead_label|; the same ead_value applies independently of the sign of the ead_label.

An EAD item can be either critical or non-critical, determined by the sign of the ead_label in the EAD item transported in the EAD field. A negative value indicates that the EAD item is critical, and a nonnegative value indicates that the EAD item is non-critical.

If an endpoint receives a critical EAD item it does not recognize or a critical EAD item that contains information that it cannot process, then the endpoint **MUST** send an EDHOC error message back as defined in Section 6, and the EDHOC session **MUST** be aborted. The EAD item specification defines the error processing. A non-critical EAD item can be ignored.

The security application registering a new EAD item needs to describe under what conditions the EAD item is critical or non-critical, and thus whether the ead_label is used with a negative or positive sign. ead_label = 0 is used for padding; see Section 3.8.1.

The security application may define multiple uses of certain EAD items, e.g., the same EAD item may be used in different EDHOC messages. Multiple occurrences of an EAD item in one EAD field may also be specified, but the criticality of the repeated EAD item is expected to be the same.

The EAD fields of EDHOC **MUST** only be used with registered EAD items; see Section 10.5. Examples of the use of EAD are provided in Appendix E.

3.8.1. Padding

EDHOC message_1 and the plaintext of message_2, message_3, and message_4 can be padded with the use of the corresponding EAD_x field, for x = 1, 2, 3, or 4. Padding in EAD_1 mitigates amplification attacks (see Section 9.7), and padding in EAD_2, EAD_3, and EAD_4 hides the true length of the plaintext (see Section 9.6). Padding **MUST** be ignored and discarded by the receiving application.

Padding is obtained by using an EAD item with ead_label = 0 and a (pseudo)randomly generated byte string of appropriate length as ead_value, noting that the ead_label and the CBOR encoding of ead_value also add bytes. For example:

- One-byte padding (optional ead_value omitted):

  EAD_x = 0x00

- Two-byte padding, using the empty byte string (0x40) as ead_value:

  EAD_x = 0x0040

- Three-byte padding, constructed from the pseudorandomly generated ead_value 0xe9 encoded as byte string:

  EAD_x = 0x0041e9

Multiple occurrences of EAD items with ead_label = 0 are allowed. Certain padding lengths require the use of at least two such EAD items.

Note that padding is non-critical because the intended behavior when receiving is to ignore it.

3.9. Application Profile

EDHOC requires certain parameters to be agreed upon between the Initiator and Responder. Some parameters can be negotiated through the protocol execution (specifically, cipher suite; see Section 3.6), but other parameters are only communicated and may not be negotiated (e.g., which authentication method is used; see Section 3.2). Yet, other parameters need to be known out-of-band to ensure successful completion, e.g., whether message_4 is used or not. The application decides which endpoint is the Initiator and which is the Responder.

The purpose of an application profile is to describe the intended use of EDHOC to allow for the relevant processing and verifications to be made, including things like the following:

1. How the endpoint detects that an EDHOC message is received. This includes how EDHOC messages are transported, for example, in the payload of a CoAP message with a certain Uri-Path or Content-Format; see Appendix A.2.

   The method of transporting EDHOC messages may also describe data carried along with the messages that are needed for the transport to satisfy the requirements of Section 3.4, e.g., connection identifiers used with certain messages; see Appendix A.2.
2. Authentication method (METHOD; see Section 3.2).
3. Profile for authentication credentials (CRED_I and CRED_R; see Section 3.5.2), e.g., profile for certificate or CCS, including supported authentication key algorithms (subject public key algorithm in X.509 or C509 certificate).
4. Type used to identify credentials (ID_CRED_I and ID_CRED_R; see Section 3.5.3).
5. Use and type of external authorization data (EAD_1, EAD_2, EAD_3, and EAD_4; see Section 3.8).
6. Identifier used as the identity of the endpoint; see Appendix D.2.
7. If message_4 shall be sent/expected, and if not, how to ensure a protected application message is sent from the Responder to the Initiator; see Section 5.5.

The application profile may also contain information about supported cipher suites. The procedure for selecting and verifying a cipher suite is still performed as described in Sections 5.2.1 and 6.3, but it may become simplified by this knowledge. EDHOC messages can be processed without the application profile, i.e., the EDHOC messages include information about the type and length of all fields.

An example of an application profile is shown in Appendix F.

For some parameters, like METHOD, the type of the ID_CRED field, or EAD, the receiver of an EDHOC message is able to verify compliance with the application profile and, if it needs to fail because of the lack of compliance, to infer the reason why the EDHOC session failed.

For other encodings, like the profiling of CRED_x in the case that it is not transported, it may not be possible to verify that the lack of compliance with the application profile was the reason for failure. For example, integrity verification in message_2 or message_3 may fail not only because of a wrong credential. For example, in case the Initiator uses a public key certificate by reference (i.e., not transported within the protocol), then both endpoints need to use an identical data structure as CRED_I or else the integrity verification will fail.

Note that it is not necessary for the endpoints to specify a single transport for the EDHOC messages. For example, a mix of CoAP and HTTP may be used along the path, and this may still allow correlation between messages.

The application profile may be dependent on the identity of the other endpoint or other information carried in an EDHOC message, but it then applies only to the later phases of the protocol when such information is known. (The Initiator does not know the identity of the Responder before having verified message_2, and the Responder does not know the identity of the Initiator before having verified message_3.)

Other conditions may be part of the application profile, such as what is the target application or use (if there is more than one application/use) to the extent that EDHOC can distinguish between them. In case multiple application profiles are used, the receiver needs to be able to determine which is applicable for a given EDHOC session, for example, based on the URI to which the EDHOC message is sent, or external authorization data type.

4. Key Derivation

4.1. Keys for EDHOC Message Processing

EDHOC uses Extract-and-Expand [RFC5869] with the EDHOC hash algorithm in the selected cipher suite to derive keys used in message processing. This section defines EDHOC_Extract (Section 4.1.1) and EDHOC_Expand (Section 4.1.2) and how to use them to derive PRK_out (Section 4.1.3), which is the shared secret session key resulting from a completed EDHOC session.

EDHOC_Extract is used to derive fixed-length uniformly pseudorandom keys (PRKs) from ECDH shared secrets. EDHOC_Expand is used to define EDHOC_KDF for generating MACs and for deriving output keying material (OKM) from PRKs.

In EDHOC, a specific message is protected with a certain PRK, but how the key is derived depends on the authentication method (Section 3.2), as detailed in Section 5.

4.1.1. EDHOC_Extract

The pseudorandom keys (PRKs) used for EDHOC message processing are derived using EDHOC_Extract:

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

where the input keying material (IKM) and salt are defined for each PRK below.

The definition of EDHOC_Extract depends on the EDHOC hash algorithm of the selected cipher suite:

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

where the Keccak Message Authentication Code (KMAC) is specified in [SP800-185].

The rest of the section defines the pseudorandom keys PRK_2e, PRK_3e2m, and PRK_4e3m; their use is shown in Figure 6. The index of a PRK indicates its use or in what message protection operation it is involved. For example, PRK_3e2m is involved in the encryption of message 3 and in calculating the MAC of message 2.

4.1.1.1. PRK_2e

The pseudorandom key PRK_2e is derived with the following input:

- The salt **SHALL** be TH_2.
- The IKM **SHALL** be the ephemeral-ephemeral ECDH shared secret G_XY (calculated from G_X and Y or G_Y and X) as defined in Section 6.3.1 of [RFC9053]. The use of G_XY gives forward secrecy in the sense that compromise of the private authentication keys does not compromise past session keys.

Example: Assuming the use of curve25519, the ECDH shared secret G_XY is the output of the X25519 function [RFC7748]:

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

Example: Assuming the use of SHA-256, the extract phase of the key derivation function is HKDF-Extract, which produces PRK_2e as follows:

```
PRK_2e = HMAC-SHA-256( TH_2, G_XY )
```

4.1.1.2. PRK_3e2m

The pseudorandom key PRK_3e2m is derived as follows:

If the Responder authenticates with a static Diffie-Hellman key, then PRK_3e2m = EDHOC_Extract(SALT_3e2m, G_RX), where

- SALT_3e2m is derived from PRK_2e (see Section 4.1.2) and
- G_RX is the ECDH shared secret calculated from G_R and X, or G_X and R (the Responder's private authentication key; see Section 3.5.1),

else PRK_3e2m = PRK_2e.

4.1.1.3. PRK_4e3m

The pseudorandom key PRK_4e3m is derived as follows:

If the Initiator authenticates with a static Diffie-Hellman key, then PRK_4e3m = EDHOC_Extract(SALT_4e3m, G_IY), where

- SALT_4e3m is derived from PRK_3e2m (see Section 4.1.2) and

- G_IY is the ECDH shared secret calculated from G_I and Y, or G_Y and I (the Initiator's private authentication key; see Section 3.5.1),

else PRK_4e3m = PRK_3e2m.

4.1.2. EDHOC_Expand and EDHOC_KDF

The output keying material (OKM) -- including keys, initialization vectors (IVs), and salts -- are derived from the PRKs using the EDHOC_KDF, which is defined through EDHOC_Expand:

```
   OKM = EDHOC_KDF( PRK, info_label, context, length )
       = EDHOC_Expand( PRK, info, length )
```

where info is encoded as the CBOR sequence:

```
info = (
  info_label : int,
  context : bstr,
  length : uint,
)
```

where:

- info_label is an int,
- context is a bstr, and
- length is the length of OKM in bytes.

When EDHOC_KDF is used to derive OKM for EDHOC message processing, then the context includes one of the transcript hashes, TH_2, TH_3, or TH_4, defined in Sections 5.3.2 and 5.4.2.

The definition of EDHOC_Expand depends on the EDHOC hash algorithm of the selected cipher suite:

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

where L = 8 · length, the output length in bits.

Figure 6 lists derivations made with EDHOC_KDF, where:

- hash_length is the length of output size of the EDHOC hash algorithm of the selected cipher suite,

- key_length is the length of the encryption key of the EDHOC AEAD algorithm of the selected cipher suite, and
- iv_length is the length of the initialization vector of the EDHOC AEAD algorithm of the selected cipher suite

Further details of the key derivation and how the output keying material is used are specified in Section 5.

```
      KEYSTREAM_2   = EDHOC_KDF( PRK_2e,   0, TH_2,      plaintext_length )
      SALT_3e2m     = EDHOC_KDF( PRK_2e,   1, TH_2,      hash_length )
      MAC_2         = EDHOC_KDF( PRK_3e2m, 2, context_2, mac_length_2 )
      K_3           = EDHOC_KDF( PRK_3e2m, 3, TH_3,      key_length )
      IV_3          = EDHOC_KDF( PRK_3e2m, 4, TH_3,      iv_length )
      SALT_4e3m     = EDHOC_KDF( PRK_3e2m, 5, TH_3,      hash_length )
      MAC_3         = EDHOC_KDF( PRK_4e3m, 6, context_3, mac_length_3 )
      PRK_out       = EDHOC_KDF( PRK_4e3m, 7, TH_4,      hash_length )
      K_4           = EDHOC_KDF( PRK_4e3m, 8, TH_4,      key_length )
      IV_4          = EDHOC_KDF( PRK_4e3m, 9, TH_4,      iv_length )
      PRK_exporter  = EDHOC_KDF( PRK_out, 10, h'',       hash_length )
```

Figure 6: Key Derivations Using EDHOC_KDF

h'' is CBOR diagnostic notation for the empty byte string, 0x40.

4.1.3. PRK_out

The pseudorandom key PRK_out, derived as shown in Figure 6, is the output session key of a completed EDHOC session.

Keys for applications are derived using EDHOC_Exporter (see Section 4.2.1) from PRK_exporter, which in turn is derived from PRK_out as shown in Figure 6. For the purpose of generating application keys, it is sufficient to store PRK_out or PRK_exporter. (Note that the word "store" used here does not imply that the application has access to the plaintext PRK_out since that may be reserved for code within a Trusted Execution Environment (TEE); see Section 9.8.)

4.2. Keys for EDHOC Applications

This section defines EDHOC_Exporter in terms of EDHOC_KDF and PRK_exporter. A key update function is defined in Appendix H.

4.2.1. EDHOC_Exporter

Keying material for the application can be derived using the EDHOC_Exporter interface defined as:

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

where:

- exporter_label is a registered uint from the "EDHOC Exporter Labels" registry (Section 10.1),
- context is a bstr defined by the application, and
- length is a uint defined by the application.

The (exporter_label, context) pair used in EDHOC_Exporter must be unique, i.e., an (exporter_label, context) **MUST NOT** be used for two different purposes. However, an application can re-derive the same key several times as long as it is done securely. For example, in most encryption algorithms, the same key can be reused with different nonces. The context can, for example, be the empty CBOR byte string.

Examples of use of the EDHOC_Exporter are given in Appendix A.

5. Message Formatting and Processing

This section specifies formatting of the messages and processing steps. Error messages are specified in Section 6. Annotated traces of EDHOC sessions are provided in [RFC9529].

An EDHOC message is encoded as a sequence of CBOR data items (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; see Appendix C.3. In order to recreate the COSE object, the recipient endpoint may need to add parameters to the COSE headers not included in the EDHOC message, for example, the parameter 'alg' to COSE_Sign1 or COSE_Encrypt0.

5.1. EDHOC Message Processing Outline

For each new/ongoing EDHOC session, the endpoints are assumed to keep an associated protocol state containing identifiers, keying material, etc. used for subsequent processing of protocol-related data. The protocol state is assumed to be associated with an application profile (Section 3.9) that provides the context for how messages are transported, identified, and processed.

EDHOC messages **SHALL** be processed according to the current protocol state. The following steps are expected to be performed at reception of an EDHOC message:

1. Detect that an EDHOC message has been received, for example, by means of a port number, URI, or media type (Section 3.9).
2. Retrieve the protocol state according to the message correlation; see Section 3.4.1. If there is no protocol state, in the case of message_1, a new protocol state is created. The Responder endpoint needs to make use of available denial-of-service mitigation (Section 9.7).
3. If the message received is an error message, then process it according to Section 6, else process it as the expected next message according to the protocol state.

The message processing steps **SHALL** be processed in order, unless otherwise stated. If the processing fails for some reason, then typically an error message is sent, the EDHOC session is aborted, and the protocol state is erased. When the composition and sending of one message is completed and before the next message is received, error messages **SHALL NOT** be sent.

After having successfully processed the last message (message_3 or message_4 depending on application profile), the EDHOC session is completed; after which, no error messages are sent and EDHOC session output **MAY** be maintained even if error messages are received. Further details are provided in the following subsections and in Section 6.

Different instances of the same message **MUST NOT** be processed in one EDHOC session. Note that processing will fail if the same message appears a second time for EDHOC processing in the same EDHOC session because the state of the protocol has moved on and now expects something else. Message deduplication **MUST** be done by the transport protocol (see Section 3.4) or, if not supported by the transport, as described in Section 7.

5.2. EDHOC Message 1

5.2.1. Formatting of Message 1

message_1 **SHALL** be a CBOR Sequence (see Appendix C.1), as defined below.

```
message_1 = (
  METHOD : int,
  SUITES_I : suites,
  G_X : bstr,
  C_I : bstr / -24..23,
  ? EAD_1,
)

suites = [ 2* int ] / int
EAD_1 = 1* ead
```

where:

- METHOD is an authentication method; see Section 3.2,
- SUITES_I is an array of cipher suites that the Initiator supports constructed as specified in Section 5.2.2,
- G_X is the ephemeral public key of the Initiator, and
- C_I is a variable-length connection identifier (note that connection identifiers are byte strings but certain values are represented as integers in the message; see Section 3.3.2), and
- EAD_1 is external authorization data; see Section 3.8.

5.2.2. Initiator Composition of Message 1

The processing steps are detailed below and in Section 6.3.

The Initiator **SHALL** compose message_1 as follows:

- Construct SUITES_I as an array of cipher suites supported by I in order of preference by I with the first cipher suite in the array being the most preferred by I and the last being the one selected by I for this EDHOC session. If the cipher suite most preferred by I is selected, then SUITES_I contains only that cipher suite and is encoded as an int. All cipher suites, if any, preferred by I over the selected one **MUST** be included. (See also Section 6.3.)
  - The selected suite is based on what the Initiator can assume to be supported by the Responder; if the Initiator previously received from the Responder an error message with error code 2 containing SUITES_R (see Section 6.3) indicating cipher suites supported by the Responder, then the Initiator **SHOULD** select its most preferred supported cipher suite among those (bearing in mind that error messages may be forged).
  - The Initiator **MUST NOT** change its order of preference for cipher suites and **MUST NOT** omit a cipher suite preferred to the selected one because of previous error messages received from the Responder.
- Generate an ephemeral ECDH key pair 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 during the EDHOC session.
- 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 in the following order:

1. Decode message_1 (see Appendix C.1).
2. Process message_1. In particular, verify that the selected cipher suite is supported and that no prior cipher suite as ordered in SUITES_I is supported.
3. If all processing completed successfully, and if EAD_1 is present, then make it available to the application for EAD processing.

If any processing step fails, then the Responder **MUST** send an EDHOC error message back as defined in Section 6, and the EDHOC session **MUST** be aborted.

5.3. EDHOC Message 2

5.3.1. Formatting of Message 2

message_2 **SHALL** be a CBOR Sequence (see Appendix C.1), as defined below.

```
message_2 = (
  G_Y_CIPHERTEXT_2 : bstr,
)
```

where:

- G_Y_CIPHERTEXT_2 is the concatenation of G_Y (i.e., the ephemeral public key of the Responder) and CIPHERTEXT_2.

5.3.2. Responder Composition of Message 2

The Responder **SHALL** compose message_2 as follows:

- Generate an ephemeral ECDH key pair 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 EDHOC session.
- Compute the transcript hash TH_2 = H(G_Y, H(message_1)), where H() is the EDHOC hash algorithm of the selected cipher suite. The input to the hash function is a CBOR Sequence. Note that H(message_1) can be computed and cached already in the processing of message_1.
- Compute MAC_2 as in Section 4.1.2 with context_2 = << C_R, ID_CRED_R, TH_2, CRED_R, ? EAD_2 >> (see Appendix C.1 for notation).
    - If the Responder authenticates with a static Diffie-Hellman key (method equals 1 or 3), then mac_length_2 is the EDHOC MAC length of the selected cipher suite. If the Responder authenticates with a signature key (method equals 0 or 2), then mac_length_2 is equal to hash_length.
    - C_R is a variable-length connection identifier. Note that connection identifiers are byte strings but certain values are represented as integers in the message; see Section 3.3.2.
    - ID_CRED_R is an identifier to facilitate the retrieval of CRED_R; see Section 3.5.3.
    - CRED_R is a CBOR item containing the authentication credential of the Responder; see Section 3.5.2.
    - EAD_2 is external authorization data; see Section 3.8.
- 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' field of a COSE_Sign1 object, computed as specified in Section 4.4 of [RFC9052] using the signature algorithm of the selected cipher suite, the private authentication key of the Responder, and the following parameters as input (see Appendix C.3 for an overview of COSE and Appendix C.1 for notation):
    - protected = << ID_CRED_R >>
    - external_aad = << TH_2, CRED_R, ? EAD_2 >>
    - payload = MAC_2
- CIPHERTEXT_2 is calculated with a binary additive stream cipher, using a keystream generated with EDHOC_Expand and the following plaintext:
    - PLAINTEXT_2 = (C_R, ID_CRED_R / bstr / -24..23, Signature_or_MAC_2, ? EAD_2)
        - If ID_CRED_R contains a single 'kid' parameter, i.e., ID_CRED_R = { 4 : kid_R }, then the compact encoding is applied; see Section 3.5.3.2.

  - C_R is the variable-length connection identifier. Note that connection identifiers are byte strings, but certain values are represented as integers in the message; see Section 3.3.2.

  - Compute KEYSTREAM_2 as in Section 4.1.2, where plaintext_length is the length of PLAINTEXT_2. For the case of plaintext_length exceeding the EDHOC_KDF output size, see Appendix G.
  - CIPHERTEXT_2 = PLAINTEXT_2 XOR KEYSTREAM_2

- 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 in the following order:

1. Decode message_2 (see Appendix C.1).
2. Retrieve the protocol state using available message correlation (e.g., the CoAP Token, the 5-tuple, or the prepended C_I; see Section 3.4.1).
3. Decrypt CIPHERTEXT_2; see Section 5.3.2.
4. If all processing is completed successfully, then make ID_CRED_R and (if present) EAD_2 available to the application for authentication and EAD processing. When and how to perform authentication is up to the application.
5. Obtain the authentication credential (CRED_R) and the authentication key of R from the application (or by other means).
6. 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. Make the result of the verification available to the application.

If any processing step fails, then the Initiator **MUST** send an EDHOC error message back as defined in Section 6, and the EDHOC session **MUST** be aborted.

5.4. EDHOC Message 3

5.4.1. Formatting of Message 3

message_3 **SHALL** be a CBOR Sequence (see Appendix C.1), as defined below.

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

5.4.2. Initiator Composition of Message 3

The Initiator **SHALL** compose message_3 as follows:

- Compute the transcript hash TH_3 = H(TH_2, PLAINTEXT_2, CRED_R), where H() is the EDHOC hash algorithm of the selected cipher suite. The input to the hash function is a CBOR Sequence. Note that TH_3 can be computed and cached already in the processing of message_2.

- Compute MAC_3 as in Section 4.1.2, with context_3 = << ID_CRED_I, TH_3, CRED_I, ? EAD_3 >>
  - If the Initiator authenticates with a static Diffie-Hellman key (method equals 2 or 3), then mac_length_3 is the EDHOC MAC length of the selected cipher suite. If the Initiator authenticates with a signature key (method equals 0 or 1), then mac_length_3 is equal to hash_length.
  - ID_CRED_I is an identifier to facilitate the retrieval of CRED_I; see Section 3.5.3.
  - CRED_I is a CBOR item containing the authentication credential of the Initiator; see Section 3.5.2.
  - EAD_3 is external authorization data; see Section 3.8.
- 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' field of a COSE_Sign1 object, computed as specified in Section 4.4 of [RFC9052] using the signature algorithm of the selected cipher suite, the private authentication key of the Initiator, and the following parameters as input (see Appendix C.3):
  - protected = << ID_CRED_I >>
  - external_aad = << TH_3, CRED_I, ? EAD_3 >>
  - payload = MAC_3
- Compute a COSE_Encrypt0 object as defined in Sections 5.2 and 5.3 of [RFC9052], with the EDHOC AEAD algorithm of the selected cipher suite, using the encryption key K_3, the initialization vector IV_3 (if used by the AEAD algorithm), the plaintext PLAINTEXT_3, and the following parameters as input (see Appendix C.3):
  - protected = h''
  - external_aad = TH_3
  - K_3 and IV_3 are defined in Section 4.1.2
  - PLAINTEXT_3 = (ID_CRED_I / bstr / -24..23, Signature_or_MAC_3, ? EAD_3)
    - If ID_CRED_I contains a single 'kid' parameter, i.e., ID_CRED_I = { 4 : kid_I }, then the compact encoding is applied; see Section 3.5.3.2.

  CIPHERTEXT_3 is the 'ciphertext' of COSE_Encrypt0.
- Compute the transcript hash TH_4 = H(TH_3, PLAINTEXT_3, CRED_I), where H() is the EDHOC hash algorithm of the selected cipher suite. The input to the hash function is a CBOR Sequence.
- Calculate PRK_out as defined in Figure 6. The Initiator can now derive application keys using the EDHOC_Exporter interface; see Section 4.2.1.
- Encode message_3 as a CBOR data item as specified in Section 5.4.1.
- Make the connection identifiers (C_I and C_R) and the application algorithms in the selected cipher suite available to the application.

After creating message_3, the Initiator can compute PRK_out (see Section 4.1.3) and derive application keys using the EDHOC_Exporter interface (see Section 4.2.1). The Initiator **SHOULD NOT** persistently store PRK_out or application keys until the Initiator has verified message_4 or a message protected with a derived application key, such as an OSCORE message, from the Responder and the application has authenticated the Responder. This is similar to waiting for an acknowledgment (ACK) in a transport protocol. The Initiator **SHOULD NOT** send protected application data until the application has authenticated the Responder.

5.4.3. Responder Processing of Message 3

The Responder **SHALL** process message_3 in the following order:

1. Decode message_3 (see Appendix C.1).
2. Retrieve the protocol state using available message correlation (e.g., the CoAP Token, the 5-tuple, or the prepended C_R; see Section 3.4.1).
3. Decrypt and verify the COSE_Encrypt0 as defined in Sections 5.2 and 5.3 of [RFC9052], with the EDHOC AEAD algorithm in the selected cipher suite and the parameters defined in Section 5.4.2.
4. If all processing completed successfully, then make ID_CRED_I and (if present) EAD_3 available to the application for authentication and EAD processing. When and how to perform authentication is up to the application.
5. Obtain the authentication credential (CRED_I) and the authentication key of I from the application (or by other means).
6. 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. Make the result of the verification available to the application.
7. Make the connection identifiers (C_I and C_R) and the application algorithms in the selected cipher suite available to the application.

After processing message_3, the Responder can compute PRK_out (see Section 4.1.3) and derive application keys using the EDHOC_Exporter interface (see Section 4.2.1). The Responder **SHOULD NOT** persistently store PRK_out or application keys until the application has authenticated the Initiator. The Responder **SHOULD NOT** send protected application data until the application has authenticated the Initiator.

If any processing step fails, then the Responder **MUST** send an EDHOC error message back as defined in Section 6, and the EDHOC session **MUST** be aborted.

5.5. EDHOC Message 4

This section specifies message_4, which is **OPTIONAL** to support. Key confirmation is normally provided by sending an application message from the Responder to the Initiator protected with a key derived with the EDHOC_Exporter, e.g., using OSCORE (see Appendix A). In deployments where no protected application message is sent from the Responder to the Initiator, message_4 **MUST** be supported and **MUST** be used. Two examples of such deployments are:

1. when EDHOC is only used for authentication and no application data is sent and

2. when application data is only sent from the Initiator to the Responder.

Further considerations about when to use message_4 are provided in Sections 3.9 and 9.1.

5.5.1. Formatting of Message 4

message_4 **SHALL** be a CBOR Sequence (see Appendix C.1), as defined below.

```
message_4 = (
  CIPHERTEXT_4 : bstr,

)
```

5.5.2. Responder Composition of Message 4

The Responder **SHALL** compose message_4 as follows:

- Compute a COSE_Encrypt0 as defined in Sections 5.2 and 5.3 of [RFC9052], with the EDHOC AEAD algorithm of the selected cipher suite, using the encryption key K_4, the initialization vector IV_4 (if used by the AEAD algorithm), the plaintext PLAINTEXT_4, and the following parameters as input (see Appendix C.3):
  - protected = h''
  - external_aad = TH_4
  - K_4 and IV_4 are defined in Section 4.1.2
  - PLAINTEXT_4 = (? EAD_4)
    - EAD_4 is external authorization data; see Section 3.8.

  CIPHERTEXT_4 is the 'ciphertext' of COSE_Encrypt0.
- Encode message_4 as a CBOR data item as specified in Section 5.5.1.

5.5.3. Initiator Processing of Message 4

The Initiator **SHALL** process message_4 as follows:

- Decode message_4 (see Appendix C.1).
- Retrieve the protocol state using available message correlation (e.g., the CoAP Token, the 5-tuple, or the prepended C_I; see Section 3.4.1).
- Decrypt and verify the COSE_Encrypt0 as defined in Sections 5.2 and 5.3 of [RFC9052], with the EDHOC AEAD algorithm in the selected cipher suite and the parameters defined in Section 5.5.2.
- Make (if present) EAD_4 available to the application for EAD processing.

If any processing step fails, then the Initiator **MUST** send an EDHOC error message back as defined in Section 6, and the EDHOC session **MUST** be aborted.

After verifying message_4, the Initiator is assured that the Responder has calculated the key PRK_out (key confirmation) and that no other party can derive the key.

6. Error Handling

This section defines the format for error messages and the processing associated with the currently defined error codes. Additional error codes may be registered; see Section 10.4.

Many kinds of errors can occur during EDHOC processing. As in CoAP, an error can be triggered by errors in the received message or internal errors in the receiving endpoint. Except for processing and formatting errors, it is up to the application when to send an error message. Sending error messages is essential for debugging but **MAY** be skipped if, for example, an EDHOC session cannot be found or due to denial-of-service reasons; see Section 9.7. Error messages in EDHOC are always fatal. After sending an error message, the sender **MUST** abort the EDHOC session. The receiver **SHOULD** treat an error message as an indication that the other party likely has aborted the EDHOC session. But since error messages might be forged, the receiver **MAY** try to continue the EDHOC session.

An EDHOC error message can be sent by either endpoint as a reply to any non-error EDHOC message. How errors at the EDHOC layer are transported depends on lower layers, which need to enable error messages to be sent and processed as intended.

error **SHALL** be a CBOR Sequence (see Appendix C.1), as defined below.

```
error = (
  ERR_CODE : int,
  ERR_INFO : any,
)
```

Figure 7: EDHOC Error Message

where:

- ERR_CODE is an error code encoded as an integer. The value 0 is reserved for success and can only be used internally; all other values (negative or positive) indicate errors.
- ERR_INFO is error information. Content and encoding depend on the error code.

The remainder of this section specifies the currently defined error codes; see Table 3. Additional error codes and corresponding error information may be specified.

ERR_CODE	ERR_INFO Type	Description
0		Reserved for success
1	tstr	Unspecified error
2	suites	Wrong selected cipher suite
3	true	Unknown credential referenced

ERR_CODE	ERR_INFO Type	Description
23		Reserved

Table 3: EDHOC Error Codes and Error Information

6.1. Success

Error code 0 **MAY** be used internally in an application to indicate success, i.e., as a standard value in case of no error, e.g., in status reporting or log files. Error code 0 **MUST NOT** be used as part of the EDHOC message exchange. If an endpoint receives an error message with error code 0, then it **MUST** abort the EDHOC session and **MUST NOT** send an error message.

6.2. Unspecified Error

Error code 1 is used for errors that do not have a specific error code defined. ERR_INFO **MUST** be a text string containing a human-readable diagnostic message that **SHOULD** be written in English, for example, "Method not supported". The diagnostic text message is mainly intended for software engineers who during debugging need to interpret it in the context of the EDHOC specification. The diagnostic message **SHOULD** be provided to the calling application where it **SHOULD** be logged.

6.3. Wrong Selected Cipher Suite

Error code 2 **MUST** only be used when replying to message_1 in case the cipher suite selected by the Initiator is not supported by the Responder or if the Responder supports a cipher suite more preferred by the Initiator than the selected cipher suite; see Section 5.2.3. In this case, ERR_INFO = SUITES_R and is of type suites; see Section 5.2.1. If the Responder does not support the selected cipher suite, then SUITES_R **MUST** include one or more supported cipher suites. If the Responder supports a cipher suite in SUITES_I other than the selected cipher suite (independently of if the selected cipher suite is supported or not), then SUITES_R **MUST** include the supported cipher suite in SUITES_I, which is most preferred by the Initiator. SUITES_R **MAY** include a single cipher suite; in which case, it is encoded as an int. If the Responder does not support any cipher suite in SUITES_I, then it **SHOULD** include all its supported cipher suites in SUITES_R.

In contrast to SUITES_I, the order of the cipher suites in SUITES_R has no significance.

6.3.1. Cipher Suite Negotiation

After receiving SUITES_R, the Initiator can determine which cipher suite to select (if any) for the next EDHOC run with the Responder. The Initiator **SHOULD** remember which selected cipher suite to use until the next message_1 has been sent; otherwise, the Initiator and Responder will run into an infinite loop where the Initiator selects its most preferred cipher suite and the Responder sends an error with supported cipher suites.

After a completed EDHOC session, the Initiator **MAY** remember the selected cipher suite to use in future EDHOC sessions with this Responder. Note that if the Initiator or Responder is updated with new cipher suite policies, any cached information may be outdated.

Note that the Initiator's list of supported cipher suites and order of preference is fixed (see Sections 5.2.1 and 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 also 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. For examples, see Section 6.3.2.

If the selected cipher suite is not the first cipher suite that the Responder supports in SUITES_I received in message_1, then the Responder **MUST** abort the EDHOC session; 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 will fail and the Initiator will abort the EDHOC session.

6.3.2. Examples

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

In Example 1 (Figure 8), the Responder supports cipher suite 6 but not the initially selected cipher suite 5. The Responder rejects the first message_1 with an error indicating support for suite 6 in SUITES_R. The Initiator also supports suite 6 and therefore selects suite 6 in the second message_1. The Initiator prepends in SUITES_I the selected suite 6 with the more preferred suites, in this case suite 5, to mitigate a potential attack on the cipher suite negotiation.

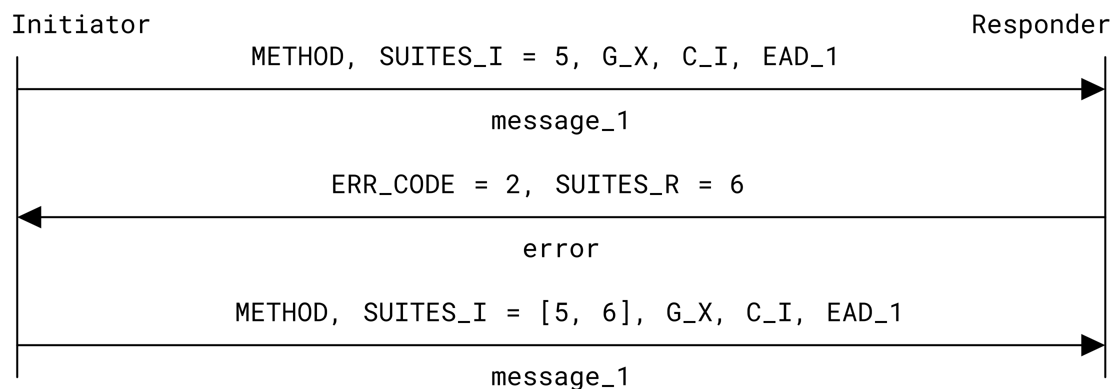

Figure 8: Cipher Suite Negotiation Example 1

In Example 2 (Figure 9), the Responder supports cipher suites 8 and 9 but not the more preferred (by the Initiator) cipher suites 5, 6 or 7. To illustrate the negotiation mechanics, we let the Initiator first make a guess that the Responder supports suite 6 but not suite 5. Since the Responder supports neither 5 nor 6, it rejects the first message_1 with an error indicating support for suites 8 and 9 in SUITES_R (in any order). The Initiator also supports suites 8 and 9,

and prefers suite 8, so it selects suite 8 in the second message_1. The Initiator prepends in SUITES_I the selected suite 8 with the more preferred suites in order of preference, in this case, suites 5, 6 and 7, to mitigate a potential attack on the cipher suite negotiation.

Note 1. If the Responder had supported suite 5, then the first message_1 would not have been accepted either, since the Responder observes that suite 5 is more preferred by the Initiator than the selected suite 6. In that case, the Responder would have included suite 5 in SUITES_R of the response, and it would then have become the selected and only suite in the second message_1.

Note 2. For each message_1, the Initiator **MUST** generate a new ephemeral ECDH key pair matching the selected cipher suite.

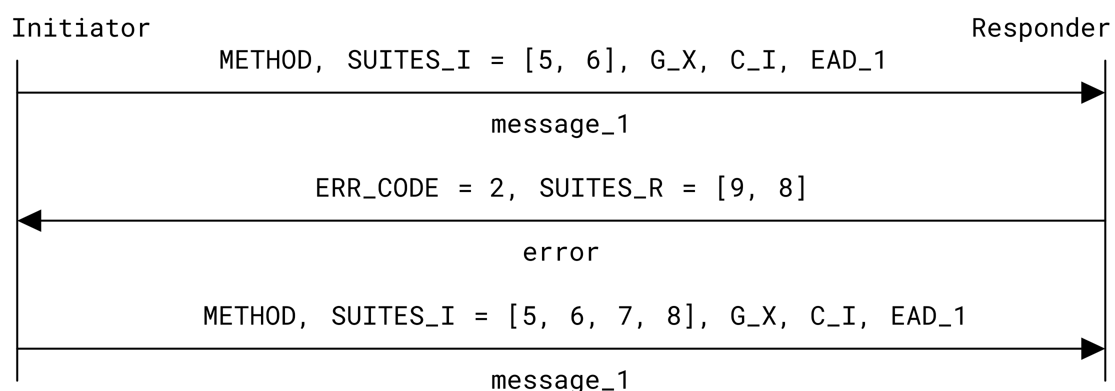

Figure 9: Cipher Suite Negotiation Example 2

6.4. Unknown Credential Referenced

Error code 3 is used for errors due to a received credential identifier (ID_CRED_R in message_2 or ID_CRED_I message_3) containing a reference to a credential that the receiving endpoint does not have access to. The intent with this error code is that the endpoint who sent the credential identifier should, for the next EDHOC session, try another credential identifier supported according to the application profile.

For example, an application profile could list x5t and x5chain as supported credential identifiers and state that x5t should be used if it can be assumed that the X.509 certificate is available at the receiving side. This error code thus enables the certificate chain to be sent only when needed, bearing in mind that error messages are not protected so an adversary can try to cause unnecessary, large credential identifiers.

For the error code 3, the error information **SHALL** be the CBOR simple value `true` (0xf5). Error code 3 **MUST NOT** be used when the received credential identifier type is not supported.

7. EDHOC Message Deduplication

By default, EDHOC assumes that message duplication is handled by the transport (which is exemplified by CoAP in this section); see Appendix A.2.

Deduplication of CoAP messages is described in Section 4.5 of [RFC7252]. This handles the case when the same Confirmable (CON) message is received multiple times due to missing acknowledgment on the CoAP messaging layer. The recommended processing in [RFC7252] is that the duplicate message is acknowledged, but the received message is only processed once by the CoAP stack.

Message deduplication is resource demanding and therefore not supported in all CoAP implementations. Since EDHOC is targeting constrained environments, it is desirable that EDHOC can optionally support transport layers that do not handle message duplication. Special care is needed to avoid issues with duplicate messages; see Section 5.1.

The guiding principle here is similar to the deduplication processing on the CoAP messaging layer, i.e., a received duplicate EDHOC message **SHALL NOT** result in another instance of the next EDHOC message. The result **MAY** be that a duplicate next EDHOC message is sent, provided it is still relevant with respect to the current protocol state. In any case, the received message **MUST NOT** be processed more than once in the same EDHOC session. This is called "EDHOC message deduplication".

An EDHOC implementation **MAY** store the previously sent EDHOC message to be able to resend it.

In principle, if the EDHOC implementation would deterministically regenerate the identical EDHOC message previously sent, it would be possible to instead store the protocol state to be able to recreate and resend the previously sent EDHOC message. However, even if the protocol state is fixed, the message generation may introduce differences that compromise security. For example, in the generation of message_3, if I is performing a (non-deterministic) ECDSA signature (say, method 0 or 1 and cipher suite 2 or 3), then PLAINTEXT_3 is randomized, but K_3 and IV_3 are the same, leading to a key and nonce reuse.

The EDHOC implementation **MUST NOT** store the previous protocol state and regenerate an EDHOC message if there is a risk that the same key and IV are used for two (or more) distinct messages.

The previous message or protocol state **MUST NOT** be kept longer than what is required for retransmission, for example, in the case of CoAP transport, no longer than the EXCHANGE_LIFETIME (see Section 4.8.2 of [RFC7252]).

8. Compliance Requirements

In the absence of an application profile specifying otherwise:

- An implementation **MAY** support only an Initiator or only a Responder.

- An implementation **MAY** support only a single method. None of the methods are mandatory to implement.
- Implementations **MUST** support 'kid' parameters. None of the other COSE header parameters are mandatory to implement.
- An implementation **MAY** support only a single credential type (CCS, CWT, X.509, or C509). None of the credential types are mandatory to implement.
- Implementations **MUST** support the EDHOC_Exporter.
- Implementations **MAY** support message_4. Error codes (ERR_CODE) 1 and 2 **MUST** be supported.
- Implementations **MUST** support EAD.
- Implementations **MUST** support cipher suites 2 and 3. Cipher suites 2 (AES-CCM-16-64-128, SHA-256, 8, P-256, ES256, AES-CCM-16-64-128, SHA-256) and 3 (AES-CCM-16-128-128, SHA-256, 16, P-256, ES256, AES-CCM-16-64-128, SHA-256) only differ in the size of the MAC length, so supporting one or both of these is not significantly different. Implementations only need to implement the algorithms needed for their supported methods.

9. Security Considerations

9.1. Security Properties

EDHOC has similar security properties as can be expected from the theoretical SIGMA-I protocol [SIGMA] and the Noise XX pattern [Noise], which are similar to methods 0 and 3, respectively. Proven security properties are detailed in the security analysis publications referenced at the end of this section.

Using the terminology from [SIGMA], EDHOC provides forward secrecy, mutual authentication with aliveness, consistency, and peer awareness. As described in [SIGMA], message_3 provides peer awareness to the Responder, while message_4 provides peer awareness to the Initiator. By including the authentication credentials in the transcript hash, EDHOC protects against an identity misbinding attack like the Duplicate Signature Key Selection (DSKS) that the MAC-then-Sign variant of SIGMA-I is otherwise vulnerable to.

As described in [SIGMA], different levels of identity protection are provided to the Initiator and Responder. EDHOC provides identity protection of the Initiator against active attacks and identity protection of the Responder against passive attacks. An active attacker can get the credential identifier of the Responder by eavesdropping on the destination address used for transporting message_1 and then sending its own message_1 to the same address. 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.

EDHOC messages might change in transit due to a noisy channel or through modification by an attacker. Changes in message_1 and message_2 (except Signature_or_MAC_2 when the signature scheme is not strongly unforgeable) are detected when verifying Signature_or_MAC_2. Changes to not strongly unforgeable Signature_or_MAC_2 and message_3 are detected when verifying CIPHERTEXT_3. Changes to message_4 are detected when verifying CIPHERTEXT_4.

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

EDHOC also adds the selection 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 that is supported by both the Initiator and Responder and to abort the EDHOC session if not.

As required by [RFC7258], IETF protocols need to mitigate pervasive monitoring when possible. Therefore, EDHOC only supports methods with ephemeral Diffie-Hellman and provides a key update function (see Appendix H) for lightweight application protocol rekeying. Either of these provides forward secrecy, in the sense that compromise of the private authentication keys does not compromise past session keys (PRK_out) and compromise of a session key does not compromise past session keys. Frequently re-running EDHOC with ephemeral Diffie-Hellman forces attackers to perform dynamic key exfiltration where the attacker must have continuous interactions with the collaborator, which is a significant sustained attack.

To limit the effect of breaches, it is important to limit the use of symmetric group keys for bootstrapping. Therefore, EDHOC 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 symmetric group keys for bootstrapping.

Compromise of the long-term keys (private signature or static DH keys) does not compromise the security of completed EDHOC sessions. Compromising the private authentication keys of one party lets an active attacker impersonate that compromised party in EDHOC sessions with other parties but does not let the attacker impersonate other parties in EDHOC sessions with the compromised party. Compromise of the long-term keys does not enable a passive attacker to compromise future session keys (PRK_out). Compromise of the HKDF 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. Compromise of PRK_out leads to compromise of all keying material derived with the EDHOC_Exporter.

Based on the cryptographic algorithm requirements (Section 9.3), EDHOC provides a minimum of 64-bit security against online brute force attacks and a minimum of 128-bit security against offline brute force attacks. To break 64-bit security against online brute force, an attacker would on average have to send 4.3 billion messages per second for 68 years, which is infeasible in constrained IoT radio technologies. A forgery against a 64-bit MAC in EDHOC breaks the security of all future application data, while a forgery against a 64-bit MAC in the subsequent application protocol (e.g., OSCORE [RFC8613]) typically only breaks the security of the data in the forged packet.

As the EDHOC session is aborted when verification fails, the security against online attacks is given by the sum of the strength of the verified signatures and MACs (including MAC in AEAD). As an example, if EDHOC is used with method 3, cipher suite 2, and message_4, the Responder is authenticated with 128-bit security against online attacks (the sum of the 64-bit MACs in message_2 and message_4). The same principle applies for MACs in an application protocol keyed by EDHOC as long as EDHOC is re-run when verification of the first MACs in the application protocol fails. As an example, if EDHOC with method 3 and cipher suite 2 is used as in Figure 2 of [EDHOC-CoAP-OSCORE], 128-bit mutual authentication against online attacks can be achieved after completion of the first OSCORE request and response.

After sending message_3, the Initiator is assured that no other party than the Responder can compute the key PRK_out. While the Initiator can securely send protected application data, the Initiator **SHOULD NOT** persistently store the keying material PRK_out until the Initiator has verified message_4 or a message protected with a derived application key, such as an OSCORE message, from the Responder. After verifying message_3, the Responder is assured that an honest Initiator has computed the key PRK_out. The Responder can securely derive and store the keying material PRK_out and send protected application data.

External authorization data sent in message_1 (EAD_1) or message_2 (EAD_2) should be considered unprotected by EDHOC; see Section 9.5. EAD_2 is encrypted, but the Responder has not yet authenticated the Initiator and the encryption does not provide confidentiality against active attacks.

External authorization data sent in message_3 (EAD_3) or message_4 (EAD_4) is protected between the Initiator and Responder by the protocol, but note that EAD fields may be used by the application before the message verification is completed; see Section 3.8. Designing a secure mechanism that uses EAD is not necessarily straightforward. This document only provides the EAD transport mechanism, but the problem of agreeing on the surrounding context and the meaning of the information passed to and from the application remains. Any new uses of EAD should be subject to careful review.

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 EDHOC session.

Repudiation: If an endpoint authenticates with a signature, the other endpoint can prove that the endpoint performed a run of the protocol by presenting the data being signed as well as the signature itself. With static Diffie-Hellman key authentication, the authenticating endpoint can deny having participated in the protocol.

Earlier versions of EDHOC have been formally analyzed [Bruni18] [Norrman20] [CottierPointcheval22] [Jacomme23] [GuentherIlunga22], and the specification has been updated based on the analysis.

9.2. Cryptographic Considerations

The SIGMA protocol requires that the encryption of message_3 provides confidentiality against active attackers and EDHOC message_4 relies on the use of authenticated encryption. Hence, the message authenticating functionality of the authenticated encryption in EDHOC is critical, i.e., authenticated encryption **MUST NOT** be replaced by plain encryption only, even if authentication is provided at another level or through a different mechanism.

To reduce message overhead, EDHOC does not use explicit nonces and instead relies on the ephemeral public keys to provide randomness to each EDHOC 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 used in more than one EDHOC message, and both parties **SHALL** generate fresh, random ephemeral key pairs. Note that an ephemeral key may be used to calculate several ECDH shared secrets. When static Diffie-Hellman authentication is used, the same ephemeral key is used in both ephemeral-ephemeral and ephemeral-static ECDH.

As discussed in [SIGMA], the encryption of message_2 only needs to protect against a passive attacker since active attackers can always get the Responder's identity by sending their own message_1. EDHOC uses the EDHOC_Expand function (typically HKDF-Expand) as a binary additive stream cipher that is proven secure as long as the expand function is a Pseudorandom Function (PRF). HKDF-Expand is not often used as a stream cipher as it is slow on long messages, and most applications require both confidentiality with indistinguishability under adaptive chosen ciphertext attack (IND-CCA2) as well as integrity protection. For the encryption of message_2, any speed difference is negligible, IND-CCA2 does not increase security, and integrity is provided by the inner MAC (and signature depending on method).

Requirements for how to securely generate, validate, and process the public keys depend on the elliptic curve. For X25519 and X448, the requirements are defined in [RFC7748]. For X25519 and X448, the check for all-zero output as specified in Section 6 of [RFC7748] **MUST** be done. For secp256r1, secp384r1, and secp521r1, the requirements are defined in Section 5 of [SP-800-56A]. For secp256r1, secp384r1, and secp521r1, at least partial public key validation **MUST** be done.

The same authentication credential **MAY** be used for both the Initiator and Responder roles. As noted in Section 12 of [RFC9052], the use of a single key for multiple algorithms is strongly discouraged unless proven secure by a dedicated cryptographic analysis. In particular, this recommendation applies to using the same private key for static Diffie-Hellman authentication and digital signature authentication. A preliminary conjecture is that a minor change to EDHOC may be sufficient to fit the analysis of a secure shared signature and ECDH key usage in [Degabriele11] and [Thormarker21]. Note that Section 5.6.3.2 of [SP-800-56A] allows a key agreement key pair to be used with a signature algorithm in certificate requests.

The property that a completed EDHOC session implies that another identity has been active is upheld as long as the Initiator does not have its own identity in the set of Responder identities it is allowed to communicate with. In trust-on-first-use (TOFU) use cases (see Appendix D.5), the Initiator should verify that the Responder's identity is not equal to its own. Any future EDHOC methods using, e.g., PSKs might need to mitigate this in other ways. However, an active attacker can gain information about the set of identities an Initiator is willing to communicate with. If the Initiator is willing to communicate with all identities except its own, an attacker can determine that a guessed Initiator identity is correct. To not leak any long-term identifiers, using a freshly generated authentication key as an identity in each initial TOFU session is **RECOMMENDED**.

NIST SP 800-56A [SP-800-56A] forbids deriving secret and non-secret randomness from the same Key Derivation Function (KDF) instance, but this decision has been criticized by Krawczyk in [HKDFpaper] and doing so is common practice. In addition to IVs, other examples are the challenge in Extensible Authentication Protocol Tunneled Transport Layer Security (EAP-TTLS), the RAND in 3GPP Authentication and Key Agreement (AKA), and the Session-Id in EAP-TLS 1.3. Note that part of KEYSTREAM_2 is also non-secret randomness, as it is known or predictable to an attacker. The more recent NIST SP 800-108 [SP-800-108] aligns with [HKDFpaper] and states that, for a secure KDF, the revelation of one portion of the derived keying material must not degrade the security of any other portion of that keying material.

9.3. Cipher Suites and Cryptographic Algorithms

When using a private cipher suite or registering new cipher suites, the choice of the key length used in the different algorithms needs to be harmonized so that a sufficient security level is maintained for authentication credentials, the EDHOC session, and the protection of application data. The Initiator and Responder should enforce a minimum security level.

The output size of the EDHOC hash algorithm **MUST** be at least 256 bits. In particular, the hash algorithms SHA-1 and SHA-256/64 (SHA-256 truncated to 64 bits) **SHALL NOT** be supported for use in EDHOC except for certificate identification with x5t and c5t. For security considerations of SHA-1, see [RFC6194]. As EDHOC integrity protects all the authentication credentials, the choice of hash algorithm in x5t and c5t does not affect security and using the same hash algorithm as in the cipher suite, but with as much truncation as possible, is **RECOMMENDED**. That is, when the EDHOC hash algorithm is SHA-256, using SHA-256/64 in x5t and c5t is **RECOMMENDED**. The EDHOC MAC length **MUST** be at least 8 bytes and the tag length of the EDHOC AEAD algorithm **MUST** be at least 64 bits. Note that secp256k1 is only defined for use with ECDSA and not for ECDH. Note that some COSE algorithms are marked as not recommended in the COSE IANA registry.

9.4. Post-Quantum Considerations

As of the publication of this specification, it is unclear when or even if a quantum computer of sufficient size and power to exploit public key cryptography will exist. Deployments that need to consider risks decades into the future should transition to Post-Quantum Cryptography (PQC) in the not-too-distant future. Many other systems should take a slower wait-and-see approach

where PQC is phased in when the quantum threat is more imminent. Current PQC algorithms have limitations compared to Elliptic Curve Cryptography (ECC), and the data sizes would be problematic in many constrained IoT systems.

Symmetric algorithms used in EDHOC, such as SHA-256 and AES-CCM-16-64-128, are practically secure against even large quantum computers. Two of NIST's security levels for quantum-resistant public key cryptography are based on AES-128 and SHA-256. A quantum computer will likely be expensive and slow due to heavy error correction. Grover's algorithm, which is proven to be optimal, cannot effectively be parallelized. It will provide little or no advantage in attacking AES, and AES-128 will remain secure for decades to come [NISTPQC].

EDHOC supports all signature algorithms defined by COSE, including PQC signature algorithms such as HSS-LMS. EDHOC is currently only specified for use with key exchange algorithms of type ECDH curves, but any Key Encapsulation Method (KEM), including PQC KEMs, can be used in method 0. While the key exchange in method 0 is specified with the terms of the Diffie-Hellman protocol, the key exchange adheres to a KEM interface: G_X is then the public key of the Initiator, G_Y is the encapsulation, and G_XY is the shared secret. Use of PQC KEMs to replace static DH authentication would likely require a specification updating EDHOC with new methods.

9.5. Unprotected Data and Privacy

The Initiator and 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 EAD_1, ID_CRED_R, EAD_2, and error messages. Using the same EAD_1 in several EDHOC sessions allows passive eavesdroppers to correlate the different sessions. Note that even if ead_value is encrypted outside of EDHOC, the ead_labels in EAD_1 are revealed to passive attackers and the ead_labels in EAD_2 are revealed to active attackers. Another consideration is that the list of supported cipher suites may potentially be used to identify the application. The Initiator and Responder must also make sure that unauthenticated data does not trigger any harmful actions. In particular, this applies to EAD_1 and error messages.

An attacker observing network traffic may use connection identifiers sent in clear in EDHOC or the subsequent application protocol to correlate packets sent on different paths or at different times. The attacker may use this information for traffic flow analysis or to track an endpoint. Application protocols using connection identifiers from EDHOC **SHOULD** provide mechanisms to update the connection identifiers and **MAY** provide mechanisms to issue several simultaneously active connection identifiers. See [RFC9000] for a non-constrained example of such mechanisms. Connection identifiers can, e.g., be chosen randomly among the set of unused 1-byte connection identifiers. Connection identity privacy mechanisms are only useful when there are not fixed identifiers, such as IP address or MAC address in the lower layers.

9.6. Updated Internet Threat Model Considerations

Since the publication of [RFC3552], there has been an increased awareness of the need to protect against endpoints that are compromised or malicious or whose interests simply do not align with the interests of users [THREAT-MODEL-GUIDANCE]. [RFC7624] describes an updated threat model for Internet confidentiality; see Section 9.1. [THREAT-MODEL-GUIDANCE] further expands

the threat model. Implementations and users should take these threat models into account and consider actions to reduce the risk of tracking by other endpoints. In particular, even data sent protected to the other endpoint, such as ID_CRED fields and EAD fields, can be used for tracking; see Section 2.7 of [THREAT-MODEL-GUIDANCE].

The fields ID_CRED_I, ID_CRED_R, EAD_2, EAD_3, and EAD_4 have variable length, and information regarding the length may leak to an attacker. A passive attacker may, e.g., be able to differentiate endpoints using identifiers of different length. To mitigate this information leakage, an implementation may ensure that the fields have a fixed length or use padding. An implementation may, e.g., only use fixed length identifiers like 'kid' of length 1. Alternatively, padding may be used (see Section 3.8.1) to hide the true length of, e.g., certificates by value in 'x5chain' or 'c5c'.

9.7. Denial of Service

EDHOC itself does not provide countermeasures against denial-of-service attacks. In particular, by sending a number of new or replayed message_1, an attacker may cause the Responder to allocate the state, perform cryptographic operations, and amplify messages. To mitigate such attacks, an implementation **SHOULD** make use of available lower layer mechanisms. For instance, when EDHOC is transferred as an exchange of CoAP messages, the CoAP server can use the Echo option defined in [RFC9175], which forces the CoAP client to demonstrate reachability at its apparent network address. To avoid an additional round trip, the Initiator can reduce the amplification factor by padding message_1, i.e., using EAD_1; see Section 3.8.1. Note that while the Echo option mitigates some resource exhaustion aspects of spoofing, it does not protect against a distributed denial-of-service attack made by real, potentially compromised, clients. Similarly, limiting amplification only reduces the impact, which still may be significant because of a large number of clients engaged in the attack.

An attacker can also send a faked message_2, message_3, message_4, or error in an attempt to trick the receiving party to send an error message and abort the EDHOC session. EDHOC implementations **MAY** evaluate if a received message is likely to have been forged by an attacker and ignore it without sending an error message or aborting the EDHOC session.

9.8. Implementation Considerations

The availability of a secure random number generator is essential for the security of EDHOC. If no true random number generator is available, a random seed **MUST** be provided from an external source and used with a cryptographically secure pseudorandom number generator. As each pseudorandom number must only be used once, an implementation needs to get a unique input to the pseudorandom number generator after reboot or continuously store state in nonvolatile memory. Appendix B.1.1 of [RFC8613] describes issues and solution approaches for writing to nonvolatile memory. Intentionally or unintentionally weak or predictable pseudorandom number generators can be abused or exploited for malicious purposes. [RFC8937] describes a way for security protocol implementations to augment their (pseudo)random number generators using a long-term private key and a deterministic signature function. This improves randomness from broken or otherwise subverted random number generators. The

same idea can be used with other secrets and functions, such as a Diffie-Hellman function or a symmetric secret, and a PRF like HMAC or KMAC. It is **RECOMMENDED** to not trust a single source of randomness and to not put unaugmented random numbers on the wire.

For many constrained IoT devices, it is problematic to support several crypto primitives. Existing devices can be expected to support either ECDSA or Edwards-curve Digital Signature Algorithm (EdDSA). If ECDSA is supported, "deterministic ECDSA", as specified in [RFC6979], **MAY** be used. Pure deterministic elliptic-curve signatures, such as deterministic ECDSA and EdDSA, have gained popularity over randomized ECDSA as their security does not depend on a source of high-quality randomness. Recent research has however found that implementations of these signature algorithms may be vulnerable to certain side-channel and fault injection attacks due to their determinism. For example, see Section 1 of [HEDGED-ECC-SIGS] for a list of attack papers. As suggested in Section 2.1.1 of [RFC9053], this can be addressed by combining randomness and determinism.

Appendix D of [CURVE-REPR] describes how Montgomery curves, such as X25519 and X448, and (twisted) Edwards curves, such as Ed25519 and Ed448, can be mapped to and from short-Weierstrass form for implementations on platforms that accelerate elliptic curve group operations in short-Weierstrass form.

All private keys, symmetric keys, and IVs **MUST** be secret. Only the Responder **SHALL** have access to the Responder's private authentication key, and only the Initiator **SHALL** have access to the Initiator's private authentication key. Implementations should provide countermeasures to side-channel attacks, such as timing attacks. Intermediate computed values, such as ephemeral ECDH keys and ECDH shared secrets, **MUST** be deleted after key derivation is completed.

The Initiator and Responder are responsible for verifying the integrity and validity of certificates. Verification of validity may require the use of a Real-Time Clock (RTC). The selection of trusted certification authorities (CAs) should be done very carefully and certificate revocation should be supported. The choice of revocation mechanism is left to the application. For example, in case of X.509 certificates, Certificate Revocation Lists [RFC5280] or the Online Certificate Status Protocol (OCSP) [RFC6960] may be used.

Similar considerations as for certificates are needed for CWT/CCS. The endpoints are responsible for verifying the integrity and validity of CWT/CCS and to handle revocation. The application needs to determine what trust anchors are relevant and have a well-defined trust-establishment process. A self-signed certificate / CWT or CCS appearing in the protocol cannot be a trigger to modify the set of trust anchors. One common way for a new trust anchor to be added to (or removed from) a device is by means firmware upgrade. See [RFC9360] for a longer discussion on trust and validation in constrained devices.

Just like for certificates, the contents of the COSE header parameters 'kcwt' and 'kccs' defined in Section 10.6 must be processed as untrusted inputs. Endpoints that intend to rely on the assertions made by a CWT/CCS obtained from any of these methods need to validate the contents. For 'kccs', which enables transport of raw public keys, the data structure used does not include any protection or verification data. 'kccs' may be used for unauthenticated operations, e.g., trust on first use, with the limitations and caveats entailed; see Appendix D.5.

The Initiator and Responder are allowed to select connection identifiers C_I and C_R, respectively, for the other party to use in the ongoing EDHOC session as well as in a subsequent application protocol (e.g., OSCORE [RFC8613]). The choice of the 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 abort the EDHOC session) or retrieve the wrong security context (which also aborts the EDHOC session as the message cannot be verified).

If two nodes unintentionally initiate two simultaneous EDHOC sessions with each other, even if they only want to complete a single EDHOC session, they **MAY** abort the EDHOC session with the lexicographically smallest G_X. Note that in cases where several EDHOC sessions with different parameter sets (method, COSE headers, etc.) are used, an attacker can affect which parameter set will be used by blocking some of the parameter sets.

If supported by the device, it is **RECOMMENDED** that at least the long-term private keys are stored in a Trusted Execution Environment (TEE) (for example, see [RFC9397]) and that sensitive operations using these keys are performed inside the TEE. To achieve even higher security, it is **RECOMMENDED** that additional operations such as ephemeral key generation, all computations of shared secrets, and storage of the PRK keys can be done inside the TEE. The use of a TEE aims at preventing code within that environment to be tampered with and preventing data used by such code to be read or tampered with by code outside that environment.

Note that HKDF-Expand has a relatively small maximum output length of 255 · hash_length, where hash_length is the output size in bytes of the EDHOC hash algorithm of the selected cipher suite. This means that when SHA-256 is used as a hash algorithm, PLAINTEXT_2 cannot be longer than 8160 bytes. This is probably not a limitation for most intended applications, but to be able to support, for example, long certificate chains or large external authorization data, there is a backwards compatible method specified in Appendix G.

The sequence of transcript hashes in EDHOC (TH_2, TH_3, and TH_4) does not make use of a so-called running hash. This is a design choice, as running hashes are often not supported on constrained platforms.

When parsing a received EDHOC message, implementations **MUST** abort the EDHOC session if the message does not comply with the CDDL for that message. Implementations are not required to support non-deterministic encodings and **MAY** abort the EDHOC session if the received EDHOC message is not encoded using deterministic CBOR. Implementations **MUST** abort the EDHOC session if validation of a received public key fails or if any cryptographic field has the wrong length. It is **RECOMMENDED** to abort the EDHOC session if the received EDHOC message is not encoded using deterministic CBOR.

10. IANA Considerations

This section gives IANA considerations and, unless otherwise noted, conforms with [RFC8126].

10.1. EDHOC Exporter Label Registry

IANA has created a new registry under the new registry group "Ephemeral Diffie-Hellman Over COSE (EDHOC)" as follows:

Registry Name: EDHOC Exporter Labels

Reference: RFC 9528

Label	Description	Reference
0	Derived OSCORE Master Secret	RFC 9528
1	Derived OSCORE Master Salt	RFC 9528
2-22	Unassigned	
23	Reserved	RFC 9528
24-32767	Unassigned	
32768-65535	Reserved for Private Use	

Table 4: EDHOC Exporter Labels

This registry also has a "Change Controller" field. For registrations made by IETF documents, the IETF is listed.

Range	Registration Procedures
0-23	Standards Action
24-32767	Expert Review
32768-65535	Private Use

Table 5: Registration Procedures for EDHOC Exporter Labels

10.2. EDHOC Cipher Suites Registry

IANA has created a new registry under the new registry group "Ephemeral Diffie-Hellman Over COSE (EDHOC)" as follows:

Registry Name: EDHOC Cipher Suites

Reference: RFC 9528

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	Array	Description	Reference
-24	N/A	Private Use	RFC 9528
-23	N/A	Private Use	RFC 9528
-22	N/A	Private Use	RFC 9528
-21	N/A	Private Use	RFC 9528
0	10, -16, 8, 4, -8, 10, -16	AES-CCM-16-64-128, SHA-256, 8, X25519, EdDSA, AES-CCM-16-64-128, SHA-256	RFC 9528
1	30, -16, 16, 4, -8, 10, -16	AES-CCM-16-128-128, SHA-256, 16, X25519, EdDSA, AES-CCM-16-64-128, SHA-256	RFC 9528
2	10, -16, 8, 1, -7, 10, -16	AES-CCM-16-64-128, SHA-256, 8, P-256, ES256, AES-CCM-16-64-128, SHA-256	RFC 9528
3	30, -16, 16, 1, -7, 10, -16	AES-CCM-16-128-128, SHA-256, 16, P-256, ES256, AES-CCM-16-64-128, SHA-256	RFC 9528
4	24, -16, 16, 4, -8, 24, -16	ChaCha20/Poly1305, SHA-256, 16, X25519, EdDSA, ChaCha20/Poly1305, SHA-256	RFC 9528
5	24, -16, 16, 1, -7, 24, -16	ChaCha20/Poly1305, SHA-256, 16, P-256, ES256, ChaCha20/Poly1305, SHA-256	RFC 9528
6	1, -16, 16, 4, -7, 1, -16	A128GCM, SHA-256, 16, X25519, ES256, A128GCM, SHA-256	RFC 9528
23		Reserved	RFC 9528
24	3, -43, 16, 2, -35, 3, -43	A256GCM, SHA-384, 16, P-384, ES384, A256GCM, SHA-384	RFC 9528
25	24, -45, 16, 5, -8, 24, -45	ChaCha20/Poly1305, SHAKE256, 16, X448, EdDSA, ChaCha20/Poly1305, SHAKE256	RFC 9528

Table 6: EDHOC Cipher Suites

Range	Registration Procedures
-65536 to -25	Specification Required
-24 to -21	Private Use

Range	Registration Procedures
-20 to 23	Standards Action with Expert Review
24 to 65535	Specification Required

Table 7: Registration Procedures for EDHOC Cipher Suites

10.3. EDHOC Method Type Registry

IANA has created a new registry under the new registry group "Ephemeral Diffie-Hellman Over COSE (EDHOC)" as follows:

Registry Name: EDHOC Method Types

Reference: RFC 9528

The columns of the registry are Value, Initiator Authentication Key, Responder Authentication Key, and Reference, where Value is an integer and the key columns are text strings describing the authentication keys.

The initial contents of the registry are shown in Table 2. Method 23 is Reserved.

Range	Registration Procedures
-65536 to -25	Specification Required
-24 to 23	Standards Action with Expert Review
24 to 65535	Specification Required

Table 8: Registration Procedures for EDHOC Method Types

10.4. EDHOC Error Codes Registry

IANA has created a new registry under the new registry group "Ephemeral Diffie-Hellman Over COSE (EDHOC)" as follows:

Registry Name: EDHOC Error Codes

Reference: RFC 9528

The columns of the registry are ERR_CODE, ERR_INFO Type, Description, Change Controller, and Reference, where ERR_CODE is an integer, ERR_INFO is a CDDL defined type, and Description is a text string. The initial contents of the registry are shown in Table 3. Error code 23 is Reserved. This registry also has a "Change Controller" field. For registrations made by IETF documents, the IETF is listed.

Range	Registration Procedures
-65536 to -25	Expert Review
-24 to 23	Standards Action
24 to 65535	Expert Review

Table 9: Registration Procedures for EDHOC Error Codes

10.5. EDHOC External Authorization Data Registry

IANA has created a new registry under the new registry group "Ephemeral Diffie-Hellman Over COSE (EDHOC)" as follows:

Registry Name: EDHOC External Authorization Data

Reference: RFC 9528

The columns of the registry are Name, Label, Description, and Reference, where Label is a nonnegative integer and the other columns are text strings. The initial contents of the registry are shown in Table 10. EAD label 23 is Reserved.

Name	Label	Description	Reference
Padding	0	Randomly generated CBOR byte string	RFC 9528, Section 3.8.1
	23	Reserved	RFC 9528

Table 10: EDHOC EAD Labels

Range	Registration Procedures
0 to 23	Standards Action with Expert Review
24 to 65535	Specification Required

Table 11: Registration Procedures for EDHOC EAD Labels

10.6. COSE Header Parameters Registry

IANA has registered the following entries in the "COSE Header Parameters" registry under the registry group "CBOR Object Signing and Encryption (COSE)" (see Table 12). The value of the 'kcwt' header parameter is a COSE Web Token (CWT) [RFC8392], and the value of the 'kccs' header parameter is a CWT Claims Set (CCS); see Section 1.4. The CWT/CCS must contain a COSE_Key in a 'cnf' claim [RFC8747]. The Value Registry column for this item is empty and omitted from the table below.

Name	Label	Value Type	Description
kcwt	13	COSE_Messages	A CBOR Web Token (CWT) containing a COSE_Key in a 'cnf' claim and possibly other claims. CWT is defined in RFC 8392. COSE_Messages is defined in RFC 9052.
kccs	14	map	A CWT Claims Set (CCS) containing a COSE_Key in a 'cnf' claim and possibly other claims. CCS is defined in RFC 8392.

Table 12: COSE Header Parameter Labels

10.7. Well-Known URI Registry

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

URI Suffix: edhoc

Change Controller: IETF

Reference: RFC 9528

Related Information: None

10.8. Media Types Registry

IANA has added the media types "application/edhoc+cbor-seq" and "application/cid-edhoc+cbor-seq" to the "Media Types" registry.

10.8.1. application/edhoc+cbor-seq Media Type Registration

Type name: application

Subtype name: edhoc+cbor-seq

Required parameters: N/A

Optional parameters: N/A

Encoding considerations: binary

Security considerations: See Section 7 of RFC 9528.

Interoperability considerations: N/A

Published specification: RFC 9528

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 in RFC 9528.

Intended usage: COMMON

Restrictions on usage: N/A

Author: See "Authors' Addresses" section.

Change Controller: IETF

10.8.2. application/cid-edhoc+cbor-seq Media Type Registration

Type name: application

Subtype name: cid-edhoc+cbor-seq

Required parameters: N/A

Optional parameters: N/A

Encoding considerations: binary

Security considerations: See Section 7 of RFC 9528.

Interoperability considerations: N/A

Published specification: RFC 9528

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 in RFC 9528.

Intended usage: COMMON

Restrictions on usage: N/A

Author: See "Authors' Addresses" section.

Change Controller: IETF

10.9. CoAP Content-Formats Registry

IANA has added the media types "application/edhoc+cbor-seq" and "application/cid-edhoc+cbor-seq" to the "CoAP Content-Formats" registry under the registry group "Constrained RESTful Environments (CoRE) Parameters".

Content Type	Content Coding	ID	Reference
application/edhoc+cbor-seq	-	64	RFC 9528
application/cid-edhoc+cbor-seq	-	65	RFC 9528

Table 13: CoAP Content-Format IDs

10.10. Resource Type (rt=) Link Target Attribute Values Registry

IANA has added the resource type "core.edhoc" to the "Resource Type (rt=) Link Target Attribute Values" registry under the registry group "Constrained RESTful Environments (CoRE) Parameters".

Value: core.edhoc

Description: EDHOC resource

Reference: RFC 9528

10.11. Expert Review Instructions

The IANA registries established in this document are defined as "Expert Review", "Specification Required", or "Standards Action with 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:

- The 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 is required. Experts should consider requesting an opinion on the correctness of registered parameters from relevant IETF working groups. Encodings that do not meet these objectives of clarity and completeness should not be registered.
- The expected usage of fields when approving code 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.
- It is recommended to have a specification even if the registration procedure is "Expert Review". When specifications are not provided for a request where Expert Review is the

assignment policy, the description provided needs to have sufficient information to verify the code points as above.

11. References

11.1. Normative References

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

[RFC3279] Bassham, L., Polk, W., and R. Housley, "Algorithms and Identifiers for the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 3279, DOI 10.17487/RFC3279, April 2002, <https://www.rfc-editor.org/info/rfc3279>.

[RFC3552] Rescorla, E. and B. Korver, "Guidelines for Writing RFC Text on Security Considerations", BCP 72, RFC 3552, DOI 10.17487/RFC3552, July 2003, <https://www.rfc-editor.org/info/rfc3552>.

[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>.

[RFC6960] Santesson, S., Myers, M., Ankney, R., Malpani, A., Galperin, S., and C. Adams, "X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP", RFC 6960, DOI 10.17487/RFC6960, June 2013, <https://www.rfc-editor.org/info/rfc6960>.

[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>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/info/rfc8126>.

[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>.

[RFC8392] Jones, M., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "CBOR Web Token (CWT)", RFC 8392, DOI 10.17487/RFC8392, May 2018, <https://www.rfc-editor.org/info/rfc8392>.

[RFC8410] Josefsson, S. and J. Schaad, "Algorithm Identifiers for Ed25519, Ed448, X25519, and X448 for Use in the Internet X.509 Public Key Infrastructure", RFC 8410, DOI 10.17487/RFC8410, August 2018, <https://www.rfc-editor.org/info/rfc8410>.

[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>.

[RFC8724] Minaburo, A., Toutain, L., Gomez, C., Barthel, D., and JC. Zuniga, "SCHC: Generic Framework for Static Context Header Compression and Fragmentation", RFC 8724, DOI 10.17487/RFC8724, April 2020, <https://www.rfc-editor.org/info/rfc8724>.

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

[RFC8747] Jones, M., Seitz, L., Selander, G., Erdtman, S., and H. Tschofenig, "Proof-of-Possession Key Semantics for CBOR Web Tokens (CWTs)", RFC 8747, DOI 10.17487/RFC8747, March 2020, <https://www.rfc-editor.org/info/rfc8747>.

[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>.

[RFC9052] Schaad, J., "CBOR Object Signing and Encryption (COSE): Structures and Process", STD 96, RFC 9052, DOI 10.17487/RFC9052, August 2022, <https://www.rfc-editor.org/info/rfc9052>.

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

[RFC9175] Amsüss, C., Preuß Mattsson, J., and G. Selander, "Constrained Application Protocol (CoAP): Echo, Request-Tag, and Token Processing", RFC 9175, DOI 10.17487/RFC9175, February 2022, <https://www.rfc-editor.org/info/rfc9175>.

[RFC9360] Schaad, J., "CBOR Object Signing and Encryption (COSE): Header Parameters for Carrying and Referencing X.509 Certificates", RFC 9360, DOI 10.17487/RFC9360, February 2023, <https://www.rfc-editor.org/info/rfc9360>.

11.2. Informative References

[Bruni18] Bruni, A., Sahl Jørgensen, T., Grønbech Petersen, T., and C. Schürmann, "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>.

[C509-CERTS] Preuß Mattsson, J., Selander, G., Raza, S., Höglund, J., and M. Furuhed, "CBOR Encoded X.509 Certificates (C509 Certificates)", Work in Progress, Internet-Draft, draft-ietf-cose-cbor-encoded-cert-09, 4 March 2024, <https://datatracker.ietf.org/doc/html/draft-ietf-cose-cbor-encoded-cert-09>.

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

[CNSA] Wikipedia, "Commercial National Security Algorithm Suite", October 2023, <https://en.wikipedia.org/w/index.php?title=Commercial_National_Security_Algorithm_Suite&oldid=1181333611>.

[CoAP-SEC-PROT] Mattsson, J. P., Palombini, F., and M. Vučinić, "Comparison of CoAP Security Protocols", Work in Progress, Internet-Draft, draft-ietf-iotops-security-protocol-comparison-04, 4 March 2024, <https://datatracker.ietf.org/doc/html/draft-ietf-iotops-security-protocol-comparison-04>.

[CottierPointcheval22] Cottier, B. and D. Pointcheval, "Security Analysis of the EDHOC protocol", September 2022, <https://arxiv.org/abs/2209.03599>.

[CURVE-REPR] Struik, R., "Alternative Elliptic Curve Representations", Work in Progress, Internet-Draft, draft-ietf-lwig-curve-representations-23, 21 January 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-lwig-curve-representations-23>.

[Degabriele11] Degabriele, J., Lehmann, A., Paterson, K., Smart, N., and M. Strefler, "On the Joint Security of Encryption and Signature in EMV", December 2011, <https://eprint.iacr.org/2011/615>.

[EAT] Lundblade, L., Mandyam, G., O'Donoghue, J., and C. Wallace, "The Entity Attestation Token (EAT)", Work in Progress, Internet-Draft, draft-ietf-rats-eat-25, 15 January 2024, <https://datatracker.ietf.org/doc/html/draft-ietf-rats-eat-25>.

[EDHOC-CoAP-OSCORE] Palombini, F., Tiloca, M., Höglund, R., Hristozov, S., and G. Selander, "Using Ephemeral Diffie-Hellman Over COSE (EDHOC) with the Constrained Application Protocol (CoAP) and Object Security for Constrained RESTful Environments (OSCORE)", Work in Progress, Internet-Draft, draft-ietf-core-oscore-edhoc-10, 29 November 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-core-oscore-edhoc-10>.

[GuentherIlunga22] Günther, F. and M. Mukendi, "Careful with MAc-then-SIGn: A Computational Analysis of the EDHOC Lightweight Authenticated Key Exchange Protocol", December 2022, <https://eprint.iacr.org/2022/1705>.

[HEDGED-ECC-SIGS] Preuß Mattsson, J., Thormarker, E., and S. Ruohomaa, "Hedged ECDSA and EdDSA Signatures", Work in Progress, Internet-Draft, draft-irtf-cfrg-det-sigs-with-noise-02, 1 March 2024, <https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-det-sigs-with-noise-02>.

[HKDFpaper] Krawczyk, H., "Cryptographic Extraction and Key Derivation: The HKDF Scheme", May 2010, <https://eprint.iacr.org/2010/264.pdf>.

[IEEE.802.15.4-2015] IEEE, "IEEE Standard for Low-Rate Wireless Networks", DOI 10.1109/IEEESTD.2016.7460875, April 2016, <https://ieeexplore.ieee.org/document/7460875>.

[Jacomme23] Jacomme, C., Klein, E., Kremer, S., and M. Racouchot, "A comprehensive, formal and automated analysis of the EDHOC protocol", October 2022, <https://hal.inria.fr/hal-03810102/>.

[KUDOS] Höglund, R. and M. Tiloca, "Key Update for OSCORE (KUDOS)", Work in Progress, Internet-Draft, draft-ietf-core-oscore-key-update-07, 4 March 2024, <https://datatracker.ietf.org/doc/html/draft-ietf-core-oscore-key-update-07>.

[LAKE-AUTHZ] Selander, G., Mattsson, J. P., Vučinić, M., Fedrecheski, G., and M. Richardson, "Lightweight Authorization using Ephemeral Diffie-Hellman Over COSE", Work in Progress, Internet-Draft, draft-ietf-lake-authz-01, 4 March 2024, <https://datatracker.ietf.org/doc/html/draft-ietf-lake-authz-01>.

[LAKE-REQS] Vučinić, M., Selander, G., Preuß Mattsson, J., and D. Garcia-Carillo, "Requirements for a Lightweight AKE for OSCORE", Work in Progress, Internet-Draft, draft-ietf-lake-reqs-04, 8 June 2020, <https://datatracker.ietf.org/doc/html/draft-ietf-lake-reqs-04>.

[NISTPQC] National Institute Standards and Technology (NIST), "Post-Quantum Cryptography FAQs", <https://csrc.nist.gov/Projects/post-quantum-cryptography/faqs>.

[Noise] Perrin, T., "The Noise Protocol Framework", Revision 34, July 2018, <https://noiseprotocol.org/noise.html>.

[Norrman20] Norrman, K., Sundararajan, V., and A. Bruni, "Formal Analysis of EDHOC Key Establishment for Constrained IoT Devices", September 2020, <https://arxiv.org/abs/2007.11427>.

[RFC2986] Nystrom, M. and B. Kaliski, "PKCS #10: Certification Request Syntax Specification Version 1.7", RFC 2986, DOI 10.17487/RFC2986, November 2000, <https://www.rfc-editor.org/info/rfc2986>.

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <https://www.rfc-editor.org/info/rfc5280>.

[RFC6194] Polk, T., Chen, L., Turner, S., and P. Hoffman, "Security Considerations for the SHA-0 and SHA-1 Message-Digest Algorithms", RFC 6194, DOI 10.17487/RFC6194, March 2011, <https://www.rfc-editor.org/info/rfc6194>.

[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>.

[RFC7624] Barnes, R., Schneier, B., Jennings, C., Hardie, T., Trammell, B., Huitema, C., and D. Borkmann, "Confidentiality in the Face of Pervasive Surveillance: A Threat Model and Problem Statement", RFC 7624, DOI 10.17487/RFC7624, August 2015, <https://www.rfc-editor.org/info/rfc7624>.

[RFC8366] Watsen, K., Richardson, M., Pritikin, M., and T. Eckert, "A Voucher Artifact for Bootstrapping Protocols", RFC 8366, DOI 10.17487/RFC8366, May 2018, <https://www.rfc-editor.org/info/rfc8366>.

[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>.

[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>.

[RFC8937] Cremers, C., Garratt, L., Smyshlyaev, S., Sullivan, N., and C. Wood, "Randomness Improvements for Security Protocols", RFC 8937, DOI 10.17487/RFC8937, October 2020, <https://www.rfc-editor.org/info/rfc8937>.

[RFC9000] Iyengar, J., Ed. and M. Thomson, Ed., "QUIC: A UDP-Based Multiplexed and Secure Transport", RFC 9000, DOI 10.17487/RFC9000, May 2021, <https://www.rfc-editor.org/info/rfc9000>.

[RFC9147] Rescorla, E., Tschofenig, H., and N. Modadugu, "The Datagram Transport Layer Security (DTLS) Protocol Version 1.3", RFC 9147, DOI 10.17487/RFC9147, April 2022, <https://www.rfc-editor.org/info/rfc9147>.

[RFC9176] Amsüss, C., Ed., Shelby, Z., Koster, M., Bormann, C., and P. van der Stok, "Constrained RESTful Environments (CoRE) Resource Directory", RFC 9176, DOI 10.17487/RFC9176, April 2022, <https://www.rfc-editor.org/info/rfc9176>.

[RFC9397] Pei, M., Tschofenig, H., Thaler, D., and D. Wheeler, "Trusted Execution Environment Provisioning (TEEP) Architecture", RFC 9397, DOI 10.17487/RFC9397, July 2023, <https://www.rfc-editor.org/info/rfc9397>.

[RFC9529] Selander, G., Preuß Mattsson, J., Serafin, M., Tiloca, M., and M. Vučinić, "Traces of Ephemeral Diffie-Hellman Over COSE (EDHOC)", RFC 9529, DOI 10.17487/RFC9529, March 2024, <https://www.rfc-editor.org/info/rfc9529>.

[SECG] Certicom Research, "SEC 1: Elliptic Curve Cryptography", Standards for Efficient Cryptography, May 2009, <https://www.secg.org/sec1-v2.pdf>.

[SIGMA] Krawczyk, H., "SIGMA: the 'SIGn-and-MAc' Approach to Authenticated Diffie-Hellman and Its Use in the IKE-Protocols", June 2003, <https://www.iacr.org/cryptodb/archive/2003/CRYPTO/1495/1495.pdf>.

[SP-800-108] Chen, L., "Recommendation for Key Derivation Using Pseudorandom Functions", NIST Special Publication 800-108 Revision 1, DOI 10.6028/NIST.SP.800-108r1-upd1, August 2022, <https://doi.org/10.6028/NIST.SP.800-108r1-upd1>.

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

[SP800-185] Kelsey, J., Chang, S., and R. Perlner, "SHA-3 Derived Functions cSHAKE, KMAC, TupleHash and ParallelHash", NIST Special Publication 800-185, DOI 10.6028/NIST.SP.800-185, December 2016, <https://doi.org/10.6028/NIST.SP.800-185>.

[Thormarker21] Thormarker, E., "On using the same key pair for Ed25519 and an X25519 based KEM", April 2021, <https://eprint.iacr.org/2021/509.pdf>.

[THREAT-MODEL-GUIDANCE] Arkko, J. and S. Farrell, "Internet Threat Model Guidance", Work in Progress, Internet-Draft, draft-arkko-arch-internet-threat-model-guidance-00, 12 July 2021, <https://datatracker.ietf.org/doc/html/draft-arkko-arch-internet-threat-model-guidance-00>.

Appendix A. Use with OSCORE and Transfer over CoAP

This appendix describes how to derive an OSCORE security context when EDHOC is used to key OSCORE and how to transfer EDHOC messages over CoAP. The use of CoAP or OSCORE with EDHOC is optional, but if you are using CoAP or OSCORE, then certain normative requirements apply as detailed in the subsections.

A.1. Deriving the OSCORE Security Context

This section specifies how to use EDHOC output to derive the OSCORE security context.

After successful processing of EDHOC message_3, the Client and Server derive Security Context parameters for OSCORE as follows (see Section 3.2 of [RFC8613]):

- The Master Secret and Master Salt **SHALL** be derived by using the EDHOC_Exporter interface (see Section 4.2.1):
  - The EDHOC Exporter Labels for deriving the OSCORE Master Secret and OSCORE Master Salt are the uints 0 and 1, respectively.
  - The context parameter is h'' (0x40), the empty CBOR byte string.
  - By default, oscore_key_length is the key length (in bytes) of the application AEAD Algorithm of the selected cipher suite for the EDHOC session. Also by default, oscore_salt_length has value 8. The Initiator and Responder **MAY** agree out-of-band on a longer oscore_key_length than the default and on shorter or longer than the default oscore_salt_length.

```
Master Secret = EDHOC_Exporter( 0, h'', oscore_key_length )
Master Salt   = EDHOC_Exporter( 1, h'', oscore_salt_length )
```

- The AEAD Algorithm **SHALL** be the application AEAD algorithm of the selected cipher suite for the EDHOC session.
- The HKDF Algorithm **SHALL** be the one based on the application hash algorithm of the selected cipher suite for the EDHOC session. For example, if SHA-256 is the application hash algorithm of the selected cipher suite, HKDF SHA-256 is used as the HKDF Algorithm in the OSCORE Security Context.
- The relationship between identifiers in OSCORE and EDHOC is specified in Section 3.3.3. The OSCORE Sender ID and Recipient ID **SHALL** be determined by EDHOC connection identifiers C_R and C_I for the EDHOC session as shown in Table 14.

	OSCORE Sender ID	OSCORE Recipient ID
EDHOC Initiator	C_R	C_I

	OSCORE Sender ID	OSCORE Recipient ID
EDHOC Responder	C_I	C_R

Table 14: Usage of Connection Identifiers in OSCORE

The Client and Server **SHALL** use the parameters above to establish an OSCORE Security Context, as per Section 3.2.1 of [RFC8613].

From then on, the Client and Server retrieve the OSCORE protocol state using the Recipient ID and optionally other transport information such as the 5-tuple.

A.2. Transferring EDHOC over CoAP

This section specifies how EDHOC can be transferred as an exchange of CoAP [RFC7252] messages. CoAP provides a reliable transport that can preserve packet ordering, provides flow and congestion control, and handles message duplication. CoAP can also perform fragmentation and mitigate certain denial-of-service attacks. The underlying CoAP transport should be used in reliable mode, in particular, when fragmentation is used, to avoid, e.g., situations with hanging endpoints waiting for each other.

EDHOC may run with the Initiator either being a CoAP client or CoAP server. We denote the former by the "forward message flow" (see Appendix A.2.1) and the latter by the "reverse message flow" (see Appendix A.2.2). By default, we assume the forward message flow, but the roles **SHOULD** be chosen to protect the most sensitive identity; see Section 9.

According to this specification, EDHOC is transferred in POST requests to the Uri-Path: "/.well-known/edhoc" (see Section 10.7) and 2.04 (Changed) responses. An application may define its own path that can be discovered, e.g., using a resource directory [RFC9176]. Client applications can use the resource type "core.edhoc" to discover a server's EDHOC resource, i.e., where to send a request for executing the EDHOC protocol; see Section 10.10. An alternative transfer of the forward message flow is specified in [EDHOC-CoAP-OSCORE].

In order for the server to correlate a message received from a client to a message previously sent in the same EDHOC session over CoAP, messages sent by the client **SHALL** be prepended with the CBOR serialization of the connection identifier that the server has selected; see Section 3.4.1. This applies both to the forward and the reverse message flows. To indicate a new EDHOC session in the forward message flow, message_1 **SHALL** be prepended with the CBOR simple value `true` (0xf5). Even if CoAP is carried over a reliable transport protocol, such as TCP, the prepending of identifiers specified here **SHALL** be practiced to enable interoperability independent of how CoAP is transported.

The prepended identifiers are encoded in CBOR and thus self-delimiting. The representation of identifiers described in Section 3.3.2 **SHALL** be used. They are sent in front of the actual EDHOC message to keep track of messages in an EDHOC session, and only the part of the body following the identifier is used for EDHOC processing. In particular, the connection identifiers within the EDHOC messages are not impacted by the prepended identifiers.

An EDHOC message has media type "application/edhoc+cbor-seq", whereas an EDHOC message prepended by a connection identifier has media type "application/cid-edhoc+cbor-seq"; see Section 10.9.

To mitigate certain denial-of-service attacks, the CoAP server **MAY** respond to the first POST request with a 4.01 (Unauthorized) containing an Echo option [RFC9175]. This forces the Initiator to demonstrate 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].

EDHOC error messages need to be transported in response to a message that failed (see Section 6). EDHOC error messages transported with CoAP are carried in the payload.

Note that the transport over CoAP can serve as a blueprint for other client-server protocols:

- The client prepends the connection identifier selected by the server (or, for message_1, the CBOR simple value `true`) to any request message it sends.
- The server does not send any such indicator, as responses are matched to request by the client-server protocol design.

A.2.1. The Forward Message Flow

In the forward message flow, the CoAP client is the Initiator and the CoAP server is the Responder. This flow protects the client identity against active attackers and the server identity against passive attackers.

In the forward message flow, the CoAP Token enables correlation on the Initiator (client) side, and the prepended C_R enables correlation on the Responder (server) side.

- EDHOC message_1 is sent in the payload of a POST request from the client to the server's resource for EDHOC, prepended with the identifier `true` (0xf5), indicating a new EDHOC session.
- EDHOC message_2 or the EDHOC error message is sent from the server to the client in the payload of the response, in the former case with response code 2.04 (Changed) and in the latter with response code as specified in Appendix A.2.3.
- 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, prepended with connection identifier C_R.
- If EDHOC message_4 is used, or in case of an error message, it is sent from the server to the client in the payload of the response, with response codes analogously to message_2. In case of an error message sent in response to message_4, it is sent analogously to the error message sent in response to message_2.

An example of a completed EDHOC session over CoAP in the forward message flow is shown in Figure 10.

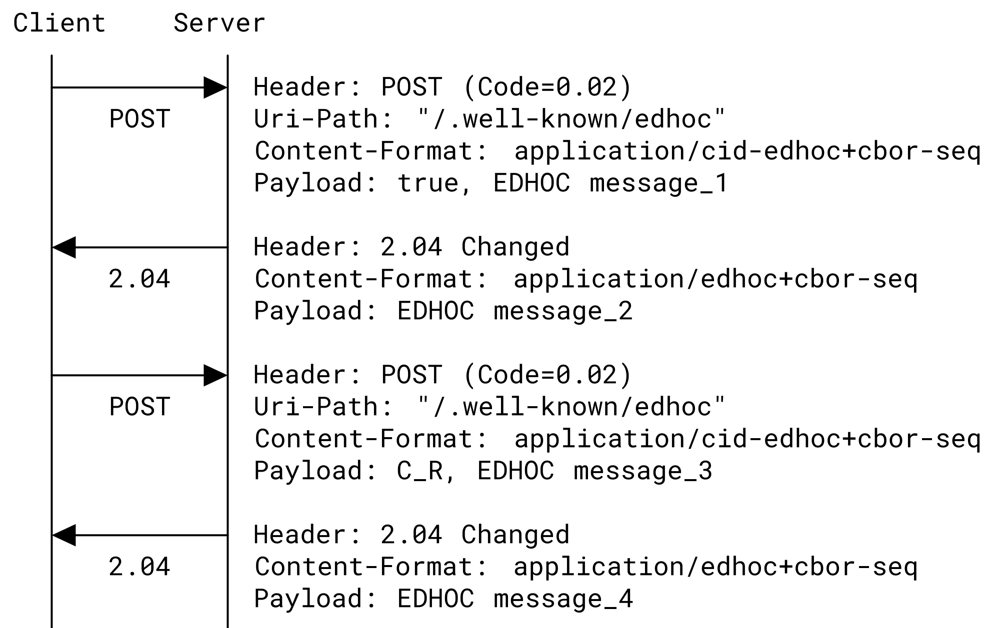

Figure 10: Example of the Forward Message Flow

The forward message flow of EDHOC can be combined with an OSCORE exchange in a total of two round trips; see [EDHOC-CoAP-OSCORE].

A.2.2. The Reverse Message Flow

In the reverse message flow, the CoAP client is the Responder and the CoAP server is the Initiator. This flow protects the server identity against active attackers and the client identity against passive attackers.

In the reverse message flow, the CoAP Token enables correlation on the Responder (client) side, and the prepended C_I enables correlation on the Initiator (server) side.

- To trigger a new EDHOC session, the client makes an empty POST request to the server's resource for EDHOC.
- EDHOC message_1 is sent from the server to the client in the payload of the response with response code 2.04 (Changed).
- EDHOC message_2 or the EDHOC error message is sent from the client to the server's resource in the payload of a POST request, prepended with connection identifier C_I.
- EDHOC message_3 or the EDHOC error message is sent from the server to the client in the payload of the response, in the former case with response code 2.04 (Changed) and in the latter with response code as specified in Appendix A.2.3.
- If EDHOC message_4 is used, or in case of an error message, it is sent from the client to the server's resource in the payload of a POST request, prepended with connection identifier C_I.

In case of an error message sent in response to message_4, it is sent analogously to an error message sent in response to message_2.

An example of a completed EDHOC session over CoAP in the reverse message flow is shown in Figure 11.

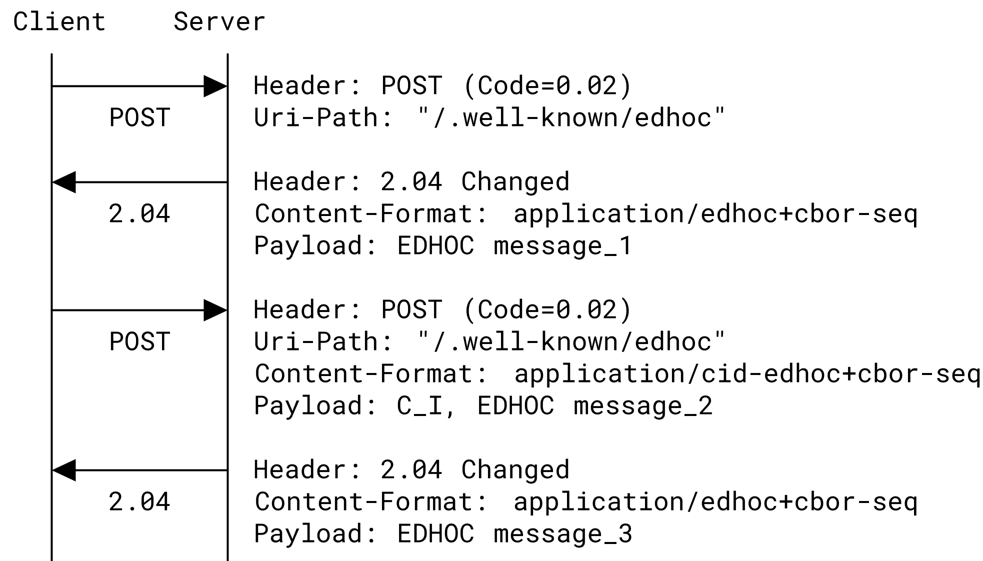

Figure 11: Example of the Reverse Message Flow

A.2.3. Errors in EDHOC over CoAP

When using EDHOC over CoAP, EDHOC error messages sent as CoAP responses **MUST** be sent in the payload of error responses, i.e., they **MUST** specify a CoAP error response code. In particular, it is **RECOMMENDED** that such error responses have response code either 4.00 (Bad Request) in case of client error (e.g., due to a malformed EDHOC message) or 5.00 (Internal Server Error) in case of server error (e.g., due to failure in deriving EDHOC keying material). The Content-Format of the error response **MUST** be set to "application/edhoc+cbor-seq"; see Section 10.9.

Appendix B. Compact Representation

This section defines a format for compact representation based on the Elliptic-Curve-Point-to-Octet-String Conversion defined in Section 2.3.3 of [SECG].

As described in Section 4.2 of [RFC6090], the x-coordinate of an elliptic curve public key is a suitable representative for the entire point whenever scalar multiplication is used as a one-way function. One example is ECDH with compact output, where only the x-coordinate of the computed value is used as the shared secret.

In EDHOC, compact representation is used for the ephemeral public keys (G_X and G_Y); see Section 3.7. Using the notation from [SECG], the output is an octet string of length ceil((log2 q) / 8), where ceil(x) is the smallest integer not less than x. See [SECG] for a definition of q, M, X, xp, and ~yp. The steps in Section 2.3.3 of [SECG] are replaced with the following steps:

1. Convert the field element xp to an octet string X of length ceil((log2 q) / 8) octets using the conversion routine specified in Section 2.3.5 of [SECG].
2. Output M = X.

The encoding of the point at infinity is not supported.

Compact representation does not change any requirements on validation; see Section 9.2. Using compact representation has some security benefits. An implementation does not need to check that the point is not the point at infinity (the identity element). Similarly, as not even the sign of the y-coordinate is encoded, compact representation trivially avoids so-called "benign malleability" attacks where an attacker changes the sign; see [SECG].

The following may be needed for validation or compatibility with APIs that do not support compact representation or do not support the full [SECG] format:

- If a compressed y-coordinate is required, then the value ~yp set to zero can be used. In such a case, the compact representation described above can be transformed into the Standards for Efficient Cryptography Group (SECG) point-compressed format by prepending it with the single byte 0x02 (i.e., M = 0x02 || X).
- If an uncompressed y-coordinate is required, then a y-coordinate has to be calculated following Section 2.3.4 of [SECG] or Appendix C of [RFC6090]. Any of the square roots (see [SECG] or [RFC6090]) can be used. The uncompressed SECG format is M = 0x04 || X || Y.

For example: The curve P-256 has the parameters (using the notation in [RFC6090]):

- $p = 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$
- a = -3
- b = 41058363725152142129326129780047268409114441015993725554835256314039467401291

Given an example x:

- x = 115792089183396302095546807154740558443406795108653336398970697772788799766525

We can calculate y as the square root $w = (x^3 + a \cdot x + b)^{((p + 1)/4)} \pmod{p}$.

- y = 83438718007019280682007586491862600528145125996401575416632522940595860276856

Note that this does not guarantee that (x, y) is on the correct elliptic curve. A full validation according to Section 5.6.2.3.3 of [SP-800-56A] is done by also checking that $0 \le x < p$ and that $y^2 \equiv x^3 + a \cdot x + b \pmod{p}$.

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

This appendix is intended to help implementors not familiar with CBOR [RFC8949], CDDL [RFC8610], COSE [RFC9052], and HKDF [RFC5869].

C.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 types of data items, integers (int, uint), simple values, 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.

The EDHOC specification sometimes use CDDL names in CBOR diagnostic notation as in, e.g., << ID_CRED_R, ? EAD_2 >>. This means that EAD_2 is optional and that ID_CRED_R and EAD_2 should be substituted with their values before evaluation. That is, if ID_CRED_R = { 4 : h'' } and EAD_2 is omitted, then << ID_CRED_R, ? EAD_2 >> = << { 4 : h'' } >>, which encodes to 0x43a10440. We also make use of the occurrence symbol "*", like in, e.g., 2* int, meaning two or more CBOR integers.

For a complete specification and more examples, see [RFC8949] and [RFC8610]. We recommend implementors 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
true	0xf5	simple value

Diagnostic	Encoded	Type
h''	0x40	byte string
h'12cd'	0x4212cd	byte string
'12cd'	0x4431326364	byte string
"12cd"	0x6431326364	text string
{ 4 : h'cd' }	0xa10441cd	map
<< 1, 2, true >>	0x430102f5	byte string
[1, 2, true]	0x830102f5	array
(1, 2, true)	0x0102f5	sequence
1, 2, true	0x0102f5	sequence

Table 15: Examples of Use of CBOR and CDDL

C.2. CDDL Definitions

This section compiles the CDDL definitions for ease of reference.

```
suites = [ 2* int ] / int

ead = (
  ead_label : int,
  ? ead_value : bstr,
)

EAD_1 = 1* ead
EAD_2 = 1* ead
EAD_3 = 1* ead
EAD_4 = 1* ead

message_1 = (
  METHOD : int,
  SUITES_I : suites,
  G_X : bstr,
  C_I : bstr / -24..23,
  ? EAD_1,
)

message_2 = (
  G_Y_CIPHERTEXT_2 : bstr,
)

PLAINTEXT_2 = (
  C_R,
  ID_CRED_R : map / bstr / -24..23,
```

```
  Signature_or_MAC_2 : bstr,
  ? EAD_2,
)

message_3 = (
  CIPHERTEXT_3 : bstr,
)

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

message_4 = (
  CIPHERTEXT_4 : bstr,
)

PLAINTEXT_4 = (
  ? EAD_4,
)

error = (
  ERR_CODE : int,
  ERR_INFO : any,
)

info = (
  info_label : int,
  context : bstr,
  length : uint,
)
```

C.3. COSE

CBOR Object Signing and Encryption (COSE) [RFC9052] describes how to create and process signatures, MACs, and encryptions using CBOR. COSE builds on JSON Object Signing and Encryption (JOSE) but is adapted to allow more efficient processing in constrained devices. EDHOC makes use of COSE_Key, COSE_Encrypt0, and COSE_Sign1 objects in the message processing:

- ECDH ephemeral public keys of type EC2 or OKP in message_1 and message_2 consist of the COSE_Key parameter named 'x'; see Sections 7.1 and 7.2 of [RFC9053].
- The ciphertexts in message_3 and message_4 consist of a subset of the single recipient encrypted data object COSE_Encrypt0, which is described in Sections 5.2 and 5.3 of [RFC9052]. The ciphertext is computed over the plaintext and associated data, using an encryption key and an initialization vector. The associated data is an Enc_structure consisting of protected headers and externally supplied data (external_aad). COSE constructs the input to the AEAD [RFC5116] for message_i (i = 3 or 4; see Sections 5.4 and 5.5, respectively) as follows:
    - Secret key K = K_i
    - Nonce N = IV_i

  - Plaintext P for message_i
  - Associated Data A = ["Encrypt0", h'', TH_i]

- Signatures in message_2 of method 0 and 2, and in message_3 of method 0 and 1, consist of a subset of the single signer data object COSE_Sign1, which is described in Sections 4.2 and 4.4 of [RFC9052]. The signature is computed over a Sig_structure containing payload, protected headers and externally supplied data (external_aad) using a private signature key, and verified using the corresponding public signature key. For COSE_Sign1, the message to be signed is:

```
[ "Signature1", protected, external_aad, payload ]
```

  where protected, external_aad, and payload are specified in Sections 5.3 and 5.4.

Different header parameters to identify X.509 or C509 certificates by reference are defined in [RFC9360] and [C509-CERTS]:

- by a hash value with the 'x5t' or 'c5t' parameters, respectively:
  - ID_CRED_x = { 34 : COSE_CertHash }, for x = I or R and
  - ID_CRED_x = { 22 : COSE_CertHash }, for x = I or R,
- or by a URI with the 'x5u' or 'c5u' parameters, respectively:
  - ID_CRED_x = { 35 : uri }, for x = I or R, and
  - ID_CRED_x = { 23 : uri }, for x = I or R.

When ID_CRED_x does not contain the actual credential, it may be very short, e.g., if the endpoints have agreed to use a key identifier parameter 'kid':

- ID_CRED_x = { 4 : kid_x }, where kid_x : kid, for x = I or R. For further optimization, see Section 3.5.3.

Note that ID_CRED_x can contain several header parameters, for example, { x5u, x5t } or { kid, kid_context }.

ID_CRED_x **MAY** also identify the credential by value. For example, a certificate chain can be transported in an ID_CRED field with COSE header parameter c5c or x5chain, as defined in [C509-CERTS] and [RFC9360]. Credentials of type CWT and CCS can be transported with the COSE header parameters registered in Section 10.6.

Appendix D. Authentication-Related Verifications

EDHOC performs certain authentication-related operations (see Section 3.5), but in general, it is necessary to make additional verifications beyond EDHOC message processing. Which verifications that are needed depend on the deployment, in particular, the trust model and the security policies, but most commonly, it can be expressed in terms of verifications of credential content.

EDHOC assumes the existence of mechanisms (certification authority or other trusted third party, pre-provisioning, etc.) for generating and distributing authentication credentials and other credentials, as well as the existence of trust anchors (CA certificates, trusted public keys, etc.). For example, a public key certificate or CWT may rely on a trusted third party whose public key is pre-provisioned, whereas a CCS or a self-signed certificate / CWT may be used when trust in the public key can be achieved by other means, or in the case of trust on first use, see Appendix D.5.

In this section, we provide some examples of such verifications. These verifications are the responsibility of the application but may be implemented as part of an EDHOC library.

D.1. Validating the Authentication Credential

In addition to the authentication key, the authentication credential may contain other parameters that need to be verified. For example:

- In X.509 and C509 certificates, signature keys typically have key usage "digitalSignature", and Diffie-Hellman public keys typically have key usage "keyAgreement" [RFC3279] [RFC8410].
- In X.509 and C509 certificates, validity is expressed using Not After and Not Before. In CWT and CCS, the "exp" and "nbf" claims have similar meanings.

D.2. Identities

The application must decide on allowing a connection or not, depending on the intended endpoint, and in particular whether it is a specific identity or in a set of identities. To prevent misbinding attacks, the identity of the endpoint is included in a MAC verified through the protocol. More details and examples are provided in this section.

Policies for what connections to allow are typically set based on the identity of the other endpoint, and endpoints 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 that authenticate with certificates having a particular range of serial numbers and signed by a particular CA. Conversely, a device may only be allowed to connect to a network that authenticates with a particular public key.

- When a Public Key Infrastructure (PKI) is used with certificates, the identity is the subject whose unique name, e.g., a domain name, a Network Access Identifier (NAI), or an Extended Unique Identifier (EUI), is included in the endpoint's certificate.

- Similarly, when a PKI is used with CWTs, the identity is the subject identified by the relevant claim(s), such as 'sub' (subject).
- When PKI is not used (e.g., CCS, self-signed certificate / CWT), the identity is typically directly associated with the authentication key of the other party. For example, if identities can be expressed in the form of unique subject names assigned to public keys, then a binding to identity is achieved by including both the public key and associated subject name in the authentication credential. CRED_I or CRED_R may be a self-signed certificate / CWT or CCS containing the authentication key and the subject name; see Section 3.5.2. Thus, each endpoint needs to know the specific authentication key / unique associated subject name or set of public authentication keys / unique associated subject names, which it is allowed to communicate with.

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". EDHOC follows SIGMA by calculating a MAC over the whole authentication credential, which in case of an X.509 or C509 certificate, includes the "subject" and "subjectAltName" fields and, in the case of CWT or CCS, includes the "sub" claim.

(While the SIGMA paper only focuses on the identity, the same principle is true for other information such as policies associated with the public key.)

D.3. Certification Path and Trust Anchors

When a Public Key Infrastructure (PKI) is used with certificates, the trust anchor is a certification authority (CA) certificate. Each party needs at least one CA public key certificate or just the CA public key. The certification path contains proof that the subject of the certificate owns the public key in the certificate. Only validated public key certificates are to be accepted.

Similarly, when a PKI is used with CWTs, each party needs to have at least one trusted third-party public key as a trust anchor to verify the end entity CWTs. The trusted third-party public key can, e.g., be stored in a self-signed CWT or in a CCS.

The signature of the authentication credential needs to be verified with the public key of the issuer. X.509 and C509 certificates includes the "Issuer" field. In CWT and CCS, the "iss" claim has a similar meaning. The public key is either a trust anchor or the public key in another valid and trusted credential in a certification path from the trust anchor to the authentication credential.

Similar verifications as made with the authentication credential (see Appendix D.1) are also needed for the other credentials in the certification path.

When PKI is not used (CCS and self-signed certificate / CWT), the trust anchor is the authentication key of the other party; in which case, there is no certification path.

D.4. Revocation Status

The application may need to verify that the credentials are not revoked; see Section 9.8. Some use cases may be served by short-lived credentials, for example, where the validity of the credential is on par with the interval between revocation checks. But, in general, credential lifetime and revocation checking are complementary measures to control credential status. Revocation information may be transported as External Authorization Data (EAD); see Appendix E.

D.5. Unauthenticated Operation

EDHOC might be used without authentication by allowing the Initiator or Responder to communicate with any identity except its own. Note that EDHOC without mutual authentication is vulnerable to active on-path attacks and therefore unsafe for general use. However, it is possible to later establish a trust relationship with an unknown or not-yet-trusted endpoint. Some examples are listed below:

- The EDHOC authentication credential can be verified out-of-band at a later stage.
- The EDHOC session key can be bound to an identity out-of-band at a later stage.
- Trust on first use (TOFU) can be used to verify that several EDHOC connections are made to the same identity. TOFU combined with proximity is a common IoT deployment model that provides good security if done correctly. Note that secure proximity based on short range wireless technology requires very low signal strength or very low latency.

Appendix E. Use of External Authorization Data

In order to reduce the number of messages and round trips, or to simplify processing, external security applications may be integrated into EDHOC by transporting related external authorization data (EAD) in the messages.

The EAD format is specified in Section 3.8. This section contains examples and further details of how EAD may be used with an appropriate accompanying specification.

- One example is third-party-assisted authorization, requested with EAD_1, and an authorization artifact ("voucher", cf. [RFC8366]) returned in EAD_2; see [LAKE-AUTHZ].
- Another example is remote attestation, requested in EAD_2, and an Entity Attestation Token (EAT) [EAT] returned in EAD_3.
- A third example is certificate enrollment, where a Certificate Signing Request (CSR) [RFC2986] is included in EAD_3, and the issued public key certificate (X.509 [RFC5280] and C509 [C509-CERTS]) or a reference thereof is returned in EAD_4.

External authorization data should be considered unprotected by EDHOC, and the protection of EAD is the responsibility of the security application (third-party authorization, remote attestation, certificate enrollment, etc.). The security properties of the EAD fields (after EDHOC processing) are discussed in Section 9.1.

The content of the EAD field may be used in the EDHOC processing of the message in which they are contained. For example, authentication-related information, like assertions and revocation information, transported in EAD fields may provide input about trust anchors or validity of credentials relevant to the authentication processing. The EAD fields (like ID_CRED fields) are therefore made available to the application before the message is verified; see details of message processing in Section 5. In the first example above, a voucher in EAD_2 made available to the application can enable the Initiator to verify the identity or the public key of the Responder before verifying the signature. An application allowing EAD fields containing authentication information thus may need to handle authentication-related verifications associated with EAD processing.

Conversely, the security application may need to wait for EDHOC message verification to complete. In the third example above, the validation of a CSR carried in EAD_3 is not started by the Responder before EDHOC has successfully verified message_3 and proven the possession of the private key of the Initiator.

The security application may reuse EDHOC protocol fields that therefore need to be available to the application. For example, the security application may use the same crypto algorithms as in the EDHOC session and therefore needs access to the selected cipher suite (or the whole SUITES_I). The application may use the ephemeral public keys G_X and G_Y as ephemeral keys or as nonces; see [LAKE-AUTHZ].

The processing of the EAD item (ead_label, ? ead_value) by the security application needs to be described in the specification where the ead_label is registered (see Section 10.5), including the optional ead_value for each message and actions in case of errors. An application may support multiple security applications that make use of EAD, which may result in multiple EAD items in one EAD field; see Section 3.8. Any dependencies on security applications with previously registered EAD items need to be documented, and the processing needs to consider their simultaneous use.

Since data carried in EAD may not be protected, or processed by the application before the EDHOC message is verified, special considerations need to be made such that it does not violate security and privacy requirements of the service that uses this data; see Section 9.5. The content in an EAD item may impact the security properties provided by EDHOC. Security applications making use of the EAD items must perform the necessary security analysis.

Appendix F. Application Profile Example

This appendix contains a rudimentary example of an application profile; see Section 3.9.

For use of EDHOC with application X, the following assumptions are made:

1. Transfer in CoAP as specified in Appendix A.2 with requests expected by the CoAP server (= Responder) at /app1-edh, no Content-Format needed.
2. METHOD = 1 (I uses signature key; R uses static DH key.)

3. CRED_I is an IEEE 802.1AR Initial Device Identifier (IDevID) encoded as a C509 certificate of type 0 [C509-CERTS].
   - R acquires CRED_I out-of-band, indicated in EAD_1.
   - ID_CRED_I = {4: h''} is a 'kid' with the value of the empty CBOR byte string.
4. CRED_R is a CCS of type OKP as specified in Section 3.5.2.
   - The CBOR map has parameters 1 (kty), -1 (crv), and -2 (x-coordinate).
   - ID_CRED_R is {14 : CCS}.
5. External authorization data is defined and processed as specified in [LAKE-AUTHZ].
6. EUI-64 is used as the identity of the endpoint (see an example in Section 3.5.2).
7. No use of message_4. The application sends protected messages from R to I.

Appendix G. Long PLAINTEXT_2

By the definition of encryption of PLAINTEXT_2 with KEYSTREAM_2, it is limited to lengths of PLAINTEXT_2 not exceeding the output of EDHOC_KDF; see Section 4.1.2. If the EDHOC hash algorithm is SHA-2, then HKDF-Expand is used, which limits the length of the EDHOC_KDF output to 255 · hash_length, where hash_length is the length of the output of the EDHOC hash algorithm given by the cipher suite. For example, with SHA-256 as the EDHOC hash algorithm, the length of the hash output is 32 bytes and the maximum length of PLAINTEXT_2 is 255 · 32 = 8160 bytes.

While PLAINTEXT_2 is expected to be much shorter than 8 kB for the intended use cases, it seems nevertheless prudent to specify a solution for the event that this should turn out to be a limitation.

A potential work-around is to use a cipher suite with a different hash function. In particular, the use of KMAC removes all practical limitations in this respect.

This section specifies a solution that works with any hash function by making use of multiple invocations of HKDF-Expand and negative values of info_label.

Consider the PLAINTEXT_2 partitioned in parts P(i) of length equal to M = 255 · hash_length, except possibly the last part P(last), which has 0 < length ≤ M.

```
PLAINTEXT_2 = P(0) | P(1) | ... | P(last)
```

where "|" indicates concatenation.

The object is to define a matching KEYSTREAM_2 of the same length and perform the encryption in the same way as defined in Section 5.3.2:

```
CIPHERTEXT_2 = PLAINTEXT_2 XOR KEYSTREAM_2
```

Define the keystream as:

```
KEYSTREAM_2 = OKM(0) | OKM(1)  | ... | OKM(last)
```

where:

```
OKM(i) = EDHOC_KDF( PRK_2e, -i, TH_2, length(P(i)) )
```

Note that if length(PLAINTEXT_2) ≤ M, then P(0) = PLAINTEXT_2 and the definition of KEYSTREAM_2 = OKM(0) coincides with Figure 6.

This describes the processing of the Responder when sending message_2. The Initiator makes the same calculations when receiving message_2 but interchanging PLAINTEXT_2 and CIPHERTEXT_2.

An application profile may specify if it supports or does not support the method described in this appendix.

Appendix H. EDHOC_KeyUpdate

To provide forward secrecy in an even more efficient way than re-running EDHOC, this section specifies the optional function EDHOC_KeyUpdate in terms of EDHOC_KDF and PRK_out.

When EDHOC_KeyUpdate is called, a new PRK_out is calculated as the output of the EDHOC_Expand function with the old PRK_out as input. The change of PRK_out causes a change to PRK_exporter, which enables the derivation of new application keys superseding the old ones, using EDHOC_Exporter; see Section 4.2.1. The process is illustrated by the following pseudocode.

```
EDHOC_KeyUpdate( context ):
   new PRK_out = EDHOC_KDF( old PRK_out, 11, context, hash_length )
   new PRK_exporter = EDHOC_KDF( new PRK_out, 10, h'', hash_length )
```

where hash_length denotes the output size in bytes of the EDHOC hash algorithm of the selected cipher suite.

The EDHOC_KeyUpdate takes a context as input to enable binding of the updated PRK_out to some event that triggered the key update. The Initiator and Responder need to agree on the context, which can, e.g., be a counter, a pseudorandom number, or a hash. To provide forward secrecy, the old PRK_out and keys derived from it (old PRK_exporter and old application keys) must be deleted as soon as they are not needed. When to delete the old keys and how to verify that they are not needed is up to the application. Note that the security properties depend on the type of context and the number of KeyUpdate iterations.

An application using EDHOC_KeyUpdate needs to store PRK_out. Compromise of PRK_out leads to compromise of all keying material derived with the EDHOC_Exporter since the last invocation of the EDHOC_KeyUpdate function.

While this key update method provides forward secrecy, it does not give as strong security properties as re-running EDHOC. EDHOC_KeyUpdate can be used to meet cryptographic limits and provide partial protection against key leakage, but it provides significantly weaker security properties than re-running EDHOC with ephemeral Diffie-Hellman. Even with frequent use of EDHOC_KeyUpdate, compromise of one session key compromises all future session keys, and an attacker therefore only needs to perform static key exfiltration [RFC7624], which is less complicated and has a lower risk profile than the dynamic case; see Section 9.1.

A similar method to do a key update for OSCORE is KUDOS; see [KUDOS].

Appendix I. Example Protocol State Machine

This appendix describes an example protocol state machine for the Initiator and Responder. States are denoted in all capitals, and parentheses denote actions taken only in some circumstances.

Note that this state machine is just an example, and that details of processing are omitted. For example:

- when error messages are being sent (with one exception);
- how credentials and EAD are processed by EDHOC and the application in the RCVD state; and
- what verifications are made, which includes not only MACs and signatures.

I.1. Initiator State Machine

The Initiator sends message_1, triggering the state machine to transition from START to WAIT_M2, and waits for message_2.

If the incoming message is an error message, then the Initiator transitions from WAIT_M2 to ABORTED. In case of error code 2 (Wrong Selected Cipher Suite), the Initiator remembers the supported cipher suites for this particular Responder and transitions from ABORTED to START. The message_1 that the Initiator subsequently sends takes into account the cipher suites supported by the Responder.

Upon receiving a non-error message, the Initiator transitions from WAIT_M2 to RCVD_M2 and processes the message. If a processing error occurs on message_2, then the Initiator transitions from RCVD_M2 to ABORTED. In case of successful processing of message_2, the Initiator transitions from RCVD_M2 to VRFD_M2.

The Initiator prepares and processes message_3 for sending. If any processing error is encountered, the Initiator transitions from VRFD_M2 to ABORTED. If message_3 is successfully sent, the Initiator transitions from VRFD_M2 to COMPLETED.

If the application profile includes message_4, then the Initiator waits for message_4. If the incoming message is an error message, then the Initiator transitions from COMPLETED to ABORTED. Upon receiving a non-error message, the Initiator transitions from COMPLETED (="WAIT_M4") to RCVD_M4 and processes the message. If a processing error occurs on message_4, then the Initiator transitions from RCVD_M4 to ABORTED. In case of successful processing of message_4, the Initiator transitions from RCVD_M4 to PERSISTED (="VRFD_M4").

If the application profile does not include message_4, then the Initiator waits for an incoming application message. If the decryption and verification of the application message is successful, then the Initiator transitions from COMPLETED to PERSISTED.

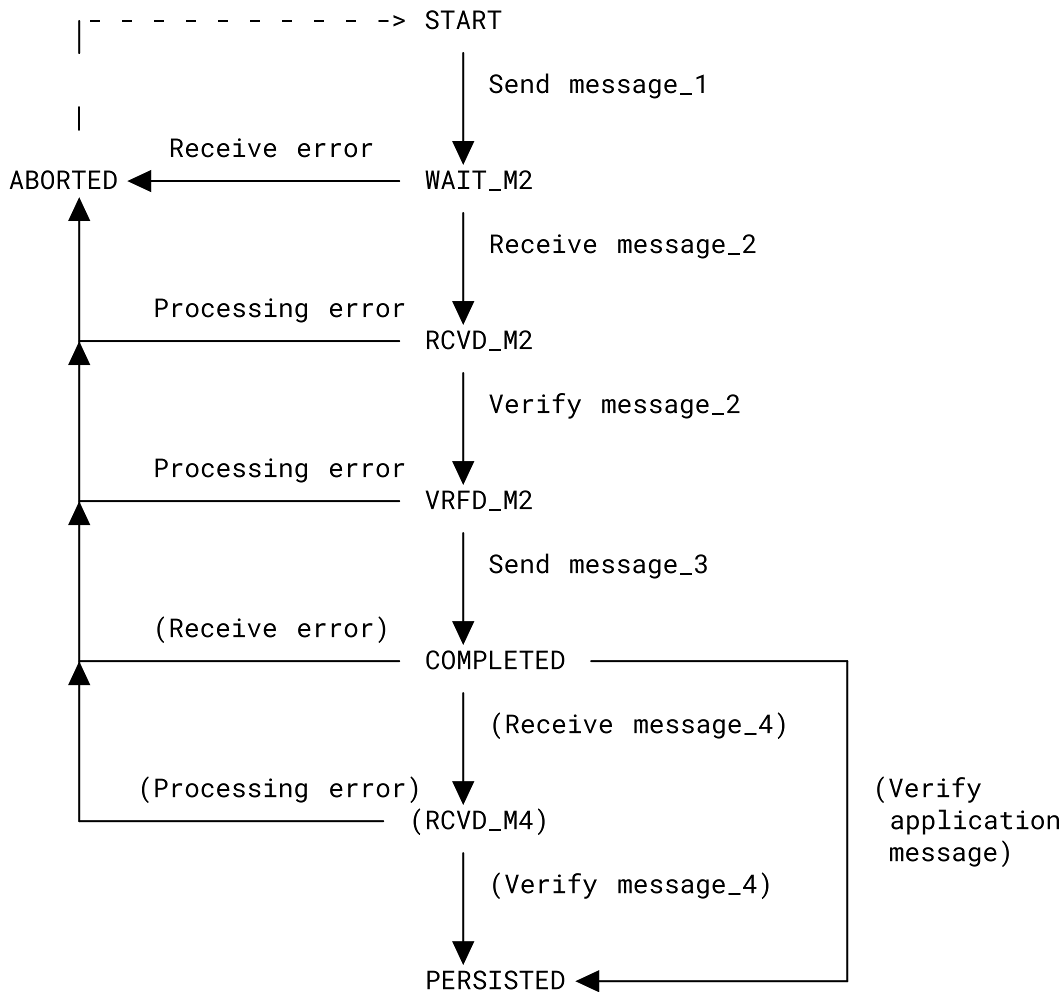

Figure 12: Initiator State Machine

I.2. Responder State Machine

Upon receiving message_1, the Responder transitions from START to RCVD_M1.

If a processing error occurs on message_1, the Responder transitions from RCVD_M1 to ABORTED. This includes sending an error message with error code 2 (Wrong Selected Cipher Suite) if the selected cipher suite in message_1 is not supported. In case of successful processing of message_1, the Responder transitions from RCVD_M1 to VRFD_M1.

The Responder prepares and processes message_2 for sending. If any processing error is encountered, the Responder transitions from VRFD_M1 to ABORTED. If message_2 is successfully sent, the Initiator transitions from VRFD_M2 to WAIT_M3 and waits for message_3.

If the incoming message is an error message, then the Responder transitions from WAIT_M3 to ABORTED.

Upon receiving message_3, the Responder transitions from WAIT_M3 to RCVD_M3. If a processing error occurs on message_3, the Responder transitions from RCVD_M3 to ABORTED. In case of successful processing of message_3, the Responder transitions from RCVD_M3 to COMPLETED (="VRFD_M3").

If the application profile includes message_4, the Responder prepares and processes message_4 for sending. If any processing error is encountered, the Responder transitions from COMPLETED to ABORTED.

If message_4 is successfully sent, or if the application profile does not include message_4, the Responder transitions from COMPLETED to PERSISTED.

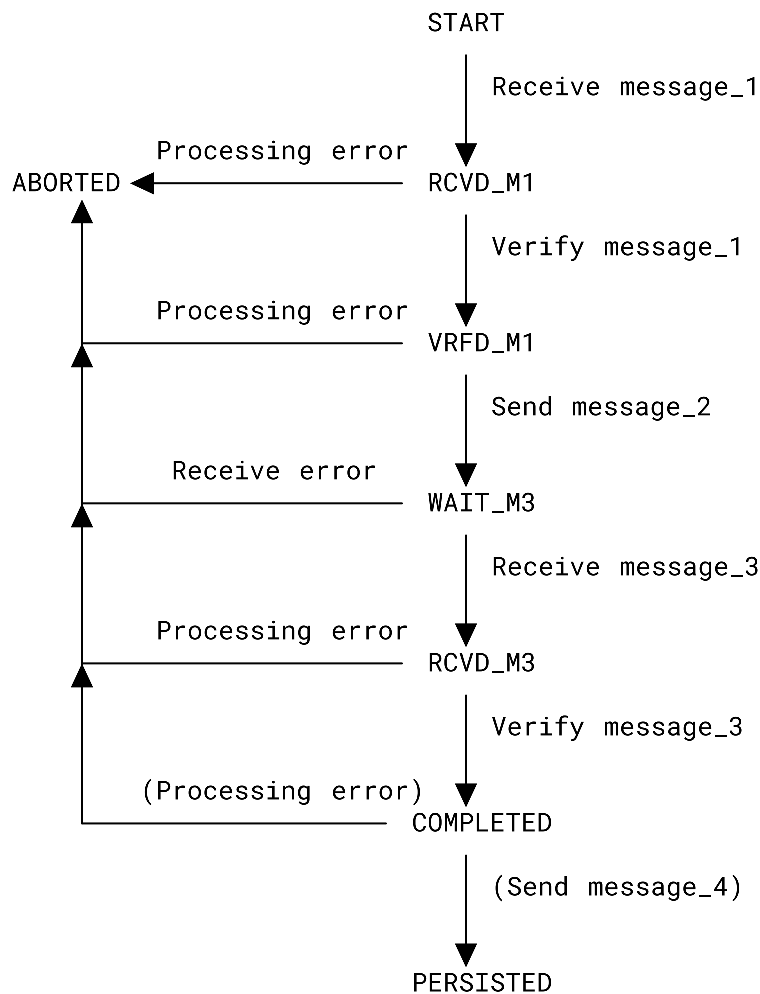

Figure 13: Responder State Machine

Acknowledgments

The authors want to thank Christian Amsüss, Karthikeyan Bhargavan, Carsten Bormann, Alessandro Bruni, Timothy Claeys, Baptiste Cottier, Roman Danyliw, Martin Disch, Martin Duke, Donald Eastlake 3rd, Lars Eggert, Stephen Farrell, Loïc Ferreira, Theis Grønbech Petersen, Felix Günther, Dan Harkins, Klaus Hartke, Russ Housley, Stefan Hristozov, Marc Ilunga, Charlie Jacomme, Elise Klein, Erik Kline, Steve Kremer, Alexandros Krontiris, Ilari Liusvaara, Rafa Marín-López, Kathleen Moriarty, David Navarro, Karl Norrman, Salvador Pérez, Radia Perlman, David Pointcheval, Maïwenn Racouchot, Eric Rescorla, Michael Richardson, Thorvald Sahl Jørgensen, Zaheduzzaman Sarker, Jim Schaad, Michael Scharf, Carsten Schürmann, John Scudder, Ludwig Seitz, Brian Sipos, Stanislav Smyshlyaev, Valery Smyslov, Peter van der Stok,

Rene Struik, Vaishnavi Sundararajan, Erik Thormarker, Marco Tiloca, Sean Turner, Michel Veillette, Mališa Vučinić, Paul Wouters, and Lei Yan for reviewing and commenting on intermediate draft versions of this document.

We are especially indebted to the late Jim Schaad for his continuous review and implementation of draft versions of this document, as well as his work on other technologies such as COSE and OSCORE without which EDHOC would not have been.

Work on this document has in part been supported by the H2020 project SIFIS-Home (grant agreement 952652).

Authors' Addresses

Göran Selander
Ericsson AB
SE-164 80 Stockholm
Sweden
Email: goran.selander@ericsson.com

John Preuß Mattsson
Ericsson AB
SE-164 80 Stockholm
Sweden
Email: john.mattsson@ericsson.com

Francesca Palombini
Ericsson AB
SE-164 80 Stockholm
Sweden
Email: francesca.palombini@ericsson.com