ACE Working Group
Internet-Draft
Intended status: Standards Track
Expires: October 27, 2017

G. Selander
J. Mattsson
F. Palombini
Ericsson AB

April 25, 2017

Ephemeral Diffie-Hellman Over COSE (EDHOC)
draft-selander-ace-cose-ecdhe-06

Abstract

This document specifies Ephemeral Diffie-Hellman Over COSE (EDHOC), a compact, and lightweight authenticated Diffie-Hellman key exchange with ephemeral keys that can be used over any layer. EDHOC messages are encoded with CBOR and COSE, allowing reuse of existing libraries.

Table of Contents

1. Introduction 2
   1.1. Terminology 3
   1.2. Requirements Language 3
2. Protocol Overview 3
3. EDHOC Overview 5
   3.1. Formatting of the Ephemeral Public Keys 6
   3.2. Key Derivation 6
4. EDHOC Authenticated with Asymmetric Keys 8
   4.1. Overview 8
   4.2. EDHOC Message 1 8
   4.3. EDHOC Message 2 11
   4.4. EDHOC Message 3 13
5. EDHOC Authenticated with Symmetric Keys 15
   5.1. Overview 15
   5.2. EDHOC Message 1 16
   5.3. EDHOC Message 2 18
   5.4. EDHOC Message 3 19
6. Error Handling 21
   6.1. Error Message Format 21
7. IANA Considerations 21
   7.1. Media Types Registry 21
8. Security Considerations 22
9. Acknowledgments 24
10. References 24
   10.1. Normative References 24
   10.2. Informative References 25
Appendix A. Test Vectors 26
Appendix B. PSK Chaining 26
Appendix C. EDHOC with CoAP and OSCOAP 26
   C.1. Transferring EDHOC in CoAP 26
   C.2. Deriving an OSCOAP context from EDHOC 27
Authors' Addresses 28

1. Introduction

Security at the application layer provides an attractive option for protecting Internet of Things (IoT) deployments, for example where transport layer security is not sufficient [I-D.hartke-core-e2e-security-reqs] or where the protocol needs to work on a variety of underlying protocols. IoT devices may be constrained in various ways, including memory, storage, processing capacity, and energy [RFC7228]. A method for protecting individual messages at the application layer suitable for constrained devices, is provided by CBOR Object Signing and Encryption (COSE) [I-D.ietf-cose-msg]), which builds on the Concise Binary Object Representation (CBOR) [RFC7049].

In order for a communication session to provide forward secrecy, the communicating parties can run an Elliptic Curve Diffie-Hellman (ECDH) key exchange protocol with ephemeral keys, from which shared key material can be derived. This document specifies Ephemeral Diffie-Hellman Over COSE (EDHOC), an authenticated ECDH protocol using CBOR and COSE objects. Authentication is based on credentials established out of band, e.g. from a trusted third party, such as an Authorization Server as specified by [I-D.ietf-ace-oauth-authz]. EDHOC supports authentication using pre-shared keys (PSK), raw public keys (RPK), and certificates (Cert). Note that this document focuses on authentication and key establishment: for integration with authorization of resource access, refer to [I-D.seitz-ace-oscoap-profile]. This document also specifies the derivation of shared key material.

The ECDH exchange and the key derivation follow [SIGMA], NIST SP-800-56a [SP-800-56a], and HKDF [RFC5869]. CBOR [RFC7049] and COSE [I-D.ietf-cose-msg] are used to implement these standards.

1.1. Terminology

This document use the same informational CBOR Data Definition Language (CDDL) [I-D.greevenbosch-appsawg-cbor-cddl] grammar as COSE (see Section 1.3 of [I-D.ietf-cose-msg]). A vertical bar | denotes byte string concatenation.

1.2. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119]. These words may also appear in this document in lowercase, absent their normative meanings.

2. Protocol Overview

SIGMA (SIGn-and-MAc) is a family of theoretical protocols with a large number of variants [SIGMA]. Like IKEv2 and TLS 1.3, EDHOC is built on a variant of the SIGMA protocol which provide identity protection, and like TLS 1.3, EDHOC implements the SIGMA-I variant as Sign-then-MAC. The SIGMA-I protocol using an AEAD algorithm is shown in Figure 1.

```
   Party U                                                     Party V
      |                          E_U                              |
      +---------------------------------------------------------->|
      |                                                           |
      |         E_V, Enc(K_2; ID_V, Sig(V; E_U, E_V);)            |
      |<----------------------------------------------------------+
      |                                                           |
      |            Enc(K_3; ID_U, Sig(U; E_V, E_U);)              |
      +---------------------------------------------------------->|
      |                                                           |
```

Figure 1: AEAD variant of the SIGMA-I protocol

The parties exchanging messages are called "U" and "V". They exchange identities and ephemeral public keys, compute the shared secret, and derive the keying material. The messages are signed, MACed, and encrypted.

- o E_U and E_V are the ECDH ephemeral public keys of U and V, respectively.

- o ID_U and ID_V are identifiers for the public keys of U and V, respectively.

- o Sig(U; .) and S(V; .) denote signatures made with the private key of U and V, respectively.

- o Enc(K; P; A) denotes AEAD encryption of plaintext P and additional authenticated data A using the key K derived from the shared secret. The AEAD MUST NOT be replaced by plain encryption, see Section 8.

As described in Appendix B of [SIGMA], in order to create a "full-fledge" protocol some additional protocol elements are needed. EDHOC adds:

- o Explicit session identifiers S_U, S_V chosen by U and V, respectively.

- o Explicit nonces N_U, N_V chosen freshly and anew with each session by U and V, respectively.

- o Computationally independent keys derived from the ECDH shared secret and used for encryption of different messages.

EDHOC also makes the following additions:

o Negotiation of key derivation, encryption, and signature algorithms:

  * U proposes one or more algorithms of the following kinds:

    + HKDF

    + AEAD

    + Signature verification

    + Signature generation

  * V selects one algorithm of each kind

o Verification of common preferred ECDH curve:

  * U lists supported ECDH curves in order of preference

  * V verifies that the ECDH curve of the ephemeral key is the most preferred common curve

o Transport of opaque application defined data.

EDHOC is designed to encrypt and integrity protect as much information as possible, and all symmetric keys are derived using as much previous information as possible. EDHOC is furthermore designed to be as compact and lightweight as possible, in terms of message sizes, processing, and the ability to reuse already existing CBOR and COSE libraries. EDHOC does not put any requirement on the lower layers and can therefore be also be used e.g. in environments without IP.

This paper is organized as follows: Section 3 specifies general properties of EDHOC, including formatting of the ephemeral public keys and key derivation, Section 4 specifies EDHOC with asymmetric key authentication, Section 5 specifies EDHOC with symmetric key authentication, and Appendix A provides a wealth of test vectors to ease implementation and ensure interoperability.

3. EDHOC Overview

EDHOC consists of three messages (message_1, message_2, message_3) that maps directly to the three messages in SIGMA-I, plus an EDHOC error message. All EDHOC messages consists of a CBOR array where the first element is an int specifying the message type (MSG_TYPE). After creating EDHOC message_3, Party U can derive the traffic key (master secret) and protected application data can therefore be sent

in parallel with EDHOC message_3. The application data may e.g. be protected using the negotiated AEAD algorithm. EDHOC may be used with the media type application/edhoc defined in Section 7.

```
Party U                                                 Party V
   |                                                       |
   | ------------------ EDHOC message_1 -----------------> |
   |                                                       |
   | <----------------- EDHOC message_2 ------------------ |
   |                                                       |
   | ----------------- EDHOC message_3 ------------------> |
   |                                                       |
   | <----------- Protected Application Data ------------> |
   |                                                       |
```

Figure 2: EDHOC message flow

The EDHOC message exchange may be authenticated using pre-shared keys (PSK), raw public keys (RPK), or certificates (Cert). EDHOC assumes the existence of mechanisms (certification authority, manual distribution, etc.) for binding identities with authentication keys (public or pre-shared). EDHOC with symmetric key authentication is very similar to EDHOC with asymmetric key authentication, the difference being that information is only MACed, not signed.

EDHOC also allows application data (APP_1, APP_2, APP_3) to be sent in the respective messages. APP_1 is unprotected, APP_2 is protected (encrypted and integrity protected), and APP_3 is protected and mutually authenticated. When EDHOC is used with asymmetric key authentication APP_2 is sent to an unauthenticated party, but with symmetric key authentication APP_2 is mutually authenticated.

3.1. Formatting of the Ephemeral Public Keys

The ECDH ephemeral public key SHALL be formatted as a COSE_Key of type EC2 or OKP according to section 13.1 and 13.2 of [I-D.ietf-cose-msg]. The curve X25519 is mandatory to implement. For Elliptic Curve Keys of type EC2, point compression is mandatory to implement.

3.2. Key Derivation

Key and IV derivation SHALL be done as specified in Section 11.1 of [I-D.ietf-cose-msg] with the following input:

o The PRF SHALL be the HKDF [RFC5869] in the ECDH-SS w/ HKDF negotiated during the message exchange (HKDF_V).

- o The secret SHALL be the ECDH shared secret as defined in Section 12.4.1 of [I-D.ietf-cose-msg].

- o salt = PSK / nil

- o The context information SHALL be the serialized COSE_KDF_Context with the following values:

  - * AlgorithmID = tstr / int

  - * PartyInfo = (nil, nil, nil)

  - * SuppPubInfo SHALL contain:

    - + keyDataLength int

    - + protected SHALL be a zero length bstr

    - + other = aad_2 / aad_3 / exchange_hash

exchange_hash = bstr

where exchange_hash, in diagnostic non-normative notation, is:

exchange_hash = H(H(message_1 | message_2) | message_3)

where H() is the hash function in HKDF_V, and | denotes byte string concatenation.

The salt SHALL only be present in the symmetric case.

Symmetric keys and IVs SHALL be derived with the negotiated PRF (HKDF_V) and with the secret set to the ECDH shared secret.

For message_i the key and IV, called K_i and IV_i, SHALL be derived using other = aad_i, where i = 2 or 3. The key SHALL be derived using AlgorithmID set to the negotiated AEAD (AEAD_V), and keyDataLength equal to the key length of AEAD_V. The IV SHALL be derived using AlgorithmID = "IV-GENERATION" as specified in section 12.1.2. of [I-D.ietf-cose-msg], and keyDataLength equal to the IV length of AEAD_V.

Application specific traffic keys and other data SHALL be derived using other = exchange_hash. AlgorithmID is defined by the application and SHALL be different for different data being derived (an example is given in Appendix C.2). keyDataLength is set to the length of the data being derived.

4. EDHOC Authenticated with Asymmetric Keys

4.1. Overview

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

- o Party U's SHALL be able to identify Party V's public key using ID_V.

- o Party V's SHALL be able to identify Party U's public key using ID_U.

ID_U and ID_V either enable the other party to retrieve the public key (kid, x5t, x5u) or they contain the public key (x5c), see [I-D.schaad-cose-x509]. Party U and party V MAY use different type of credentials, e.g. one uses RPK and the other Cert. Party U and party V MAY use different signature algorithms.

EDHOC with asymmetric key authentication is illustrated in Figure 3.

```
Party U                                                          Party V
|                      S_U, N_U, E_U, ALG_1, APP_1                     |
+--------------------------------------------------------------------->|
|                               message_1                              |
|                                                                      |
|S_U, S_V, N_V, E_V, ALG_2, Enc(K_2; APP_2, ID_V, Sig(V; aad_2); aad_2)|
|<---------------------------------------------------------------------+
|                               message_2                              |
|                                                                      |
|           S_V, Enc(K_3; APP_3, ID_U, Sig(U; aad_3); aad_3)           |
+--------------------------------------------------------------------->|
|                               message_3                              |
```

Figure 3: EDHOC with asymmetric key authentication.

4.1.1. Mandatory to Implement Algorithms

For EDHOC authenticated with asymmetric keys, the COSE algorithms ECDH-SS + HKDF-256, AES-CCM-64-64-128, and EdDSA are mandatory to implement.

4.2. EDHOC Message 1

4.2.1. Formatting of Message 1

message_1 SHALL be a CBOR array as defined below

```
message_1 = [
  MSG_TYPE : int,
  S_U : bstr,
  N_U : bstr,
  E_U : serialized_COSE_Key,
  ECDH-Curves_U : alg_array,
  HKDFs_U : alg_array,
  AEADs_U : alg_array,
  SIGs_V : alg_array,
  SIGs_U : alg_array,
  ? APP_1 : bstr
]

serialized_COSE_Key = bstr .cbor COSE_Key

alg_array = [ + alg : int / tstr ]
```

where:

- MSG_TYPE = 1
- S_U - variable length session identifier
- N_U - 64-bit random nonce
- E_U - the ephemeral public key of Party U
- ECDH-Curves_U - EC curves for ECDH which Party U supports, in the order of decreasing preference
- HKDFs_U - supported ECDH-SS w/ HKDF algorithms
- AEADs_U - supported AEAD algorithms
- SIGs_V - signature algorithms, with which Party U supports verification
- SIGs_U - signature algorithms, with which Party U supports signing
- APP_1 - bstr containing application data

4.2.2. Party U Processing of Message 1

Party U SHALL compose message_1 as follows:

- o Determine which ECDH curve to use with Party V. If U previously received from Party V an error message to message_1 with diagnostic payload identifying an ECDH curve in ECDH-Curves_U, then U SHALL retrieve an ephemeral from that curve. Otherwise the first curve in ECDH-Curves_U MUST be used.
- o Retrieve an ephemeral ECDH key pair generated as specified in Section 5 of [SP-800-56a] and format the ephemeral public key E_U as a COSE_key as specified in Section 3.1.
- o Generate the pseudo-random nonce N_U
- o Choose a session identifier S_U and store it for the length of the protocol.
- o Format message_1 as specified in Section 4.2.1.

4.2.3. Party V Processing of Message 1

Party V SHALL process message_1 as follows:

- o Verify (OPTIONAL) that N_U has not been received before.
- o Verify that at least one of each kind of the proposed algorithms are supported.
- o Verify that the ECDH curve used in E_U is supported, and that no prior curve in ECDH-Curves_U is supported

If any verification step fails, Party V MUST send an EDHOC error message back, formatted as defined in Section 6.1, and the protocol MUST be discontinued. If V does not support the ECDH curve used in E_U, but supports another ECDH curves in ECDH-Curves_U, then the error message MUST include the following diagnostic payload describing the first supported ECDH curve in ECDH-Curves_U:

ERR_MSG = "Curve not supported; X"

where X is the first curve in ECDH-Curves_U that V supports, encoded as in Table 22 of {{I-D.ietf-cose-msg}}.

4.3. EDHOC Message 2

4.3.1. Formatting of Message 2

message_2 SHALL be a CBOR array as defined below

```
message_2 = [
  data_2,
  COSE_ENC_2 : COSE_Encrypt0
]

data_2 = (
  MSG_TYPE : int,
  S_U : bstr,
  S_V : bstr,
  N_V : bstr,
  E_V : serialized_COSE_Key,
  HKDF_V : int / tstr,
  AEAD_V : int / tstr,
  SIG_V : int / tstr,
  SIG_U : int / tstr
)

aad_2 = bstr
```

where aad_2, in diagnostic non-normative notation, is:

```
aad_2 = message_1 | [ data_2 ] | ? Cert_V
```

where:

- o MSG_TYPE = 2
- o S_V - variable length session identifier
- o N_V - 64-bit random nonce
- o E_V - the ephemeral public key of Party V
- o HKDF_V - a single chosen algorithm from HKDFs_U
- o AEAD_V - a single chosen algorithm from AEADs_U
- o SIG_V - a single chosen algorithm from SIGs_V with which Party V signs
- o SIG_U - a single chosen algorithm from SIGs_U with which Party U signs

- o COSE_ENC_2 has the following fields and values:

  * external_aad = aad_2

  * plaintext = [COSE_SIG_V, ? APP_2]

- o COSE_SIG_V is a COSE_Sign1 object with the following fields and values:

  * unprotected = { xyz: ID_V }

  * detached payload = aad_2

- o xyz - any COSE map label that can identify a public key, see Section 4.1

- o ID_V - identifier for the public key of Party V

- o APP_2 - bstr containing application data

- o Cert_V - The end-entity certificate of Party V

- o H() - the hash function in HKDF_V

4.3.2. Party V Processing of Message 2

Party V SHALL compose message_2 as follows:

- o Retrieve an ephemeral ECDH key pair generated as specified in Section 5 of [SP-800-56a] using same curve as used in E_U. Format the ephemeral public key E_V as a COSE_key as specified in Section 3.1.

- o Generate the pseudo-random nonce N_V

- o Choose a session identifier S_V and store it for the length of the protocol.

- o Select HKDF_V, AEAD_V, SIG_V, and SIG_U from the algorithms proposed in HKDFs_U, AEADs_U, SIGs_V, and SIGs_U.

- o Format message_2 as specified in Section 4.3.1:

  * COSE_Sign1 is computed as defined in section 4.4 of [I-D.ietf-cose-msg], using algorithm SIG_V and the private key of Party V.

   * COSE_Encrypt0 is computed as defined in section 5.3 of [I-D.ietf-cose-msg], with AEAD_V, K_2, and IV_2. The AEAD algorithm MUST NOT be replaced by plain encryption, see Section 8.

     + If certificates are used then aad_2 MUST include Cert_V

4.3.3. Party U Processing of Message 2

Party U SHALL process message_2 as follows:

- Use the session identifier S_U to retrieve the protocol state.

- Verify that HKDF_V, AEAD_V, SIG_V, and SIG_U were proposed in HKDFs_U, AEADs_U, SIGs_V, and SIGs_U.

- Verify (OPTIONAL) that N_V has not been received before.

- Verify message_2 as specified in Section 4.3.1:

   * COSE_Encrypt0 is decrypted defined in section 5.3 of [I-D.ietf-cose-msg], with AEAD_V, K_2, and IV_2.

   * COSE_Sign1 is verified as defined in section 4.4 of [I-D.ietf-cose-msg], using algorithm SIG_V and the public key of Party V.

If any verification step fails, Party V MUST send an EDHOC error message back, formatted as defined in Section 6.1, and the protocol MUST be discontinued.

4.4. EDHOC Message 3

4.4.1. Formatting of Message 3

message_3 SHALL be a CBOR array as defined below

```
message_3 = [
  data_3,
  COSE_ENC_3 : COSE_Encrypt0
]

data_3 = (
  MSG_TYPE : int,
  S_V : bstr
)

aad_3 = bstr
```

where aad_3, in diagnostic non-normative notation, is:

```
aad_3 = H( message_1 | message_2 ) | [ data_3 ] | ? Cert_U
```

where:

- o MSG_TYPE = 3
- o COSE_ENC_3 has the following fields and values:
  - * external_aad = aad_3
  - * plaintext = [COSE_SIG_U, ? APP_3]
- o COSE_SIG_U is a COSE_Sign1 object with the following fields and values:
  - * unprotected = { xyz: ID_U }
  - * detached payload = aad_3
- o xyz - any COSE map label that can identify a public key, see Section 4.1
- o ID_U - identifier for the public key of Party U
- o APP_3 - bstr containing application data
- o Cert_U - The end-entity certificate of Party U

4.4.2. Party U Processing of Message 3

Party U SHALL compose message_3 as follows:

- o Format message_3 as specified in Section 4.4.1:
  - * COSE_Sign1 is computed as defined in section 4.4 of [I-D.ietf-cose-msg], using algorithm SIG_U and the private key of Party U.
  - * COSE_Encrypt0 is computed as defined in section 5.3 of [I-D.ietf-cose-msg], with AEAD_V, K_3, and IV_3. The AEAD algorithm MUST NOT be replaced by plain encryption, see Section 8.
    - + If certificates are used then aad_3 MUST include Cert_U

4.4.3. Party V Processing of Message 3

Party V SHALL process message_3 as follows:

o Use the session identifier S_V to retrieve the protocol state.

o Verify message_3 as specified in Section 4.4.1.

   * COSE_Encrypt0 is decrypted as defined in section 5.3 of [I-D.ietf-cose-msg], with AEAD_V, K_3, and IV_3.

   * COSE_Sign1 is verified as defined in section 4.4 of [I-D.ietf-cose-msg], using algorithm SIG_U and the public key of Party U;

If any verification step fails, Party V MUST send an EDHOC error message back, formatted as defined in Section 6.1, and the protocol MUST be discontinued.

5. EDHOC Authenticated with Symmetric Keys

5.1. Overview

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

o Party V SHALL be able to identify the PSK using KID.

KID either enable the other party to retrieve the PSK or contain the PSK (e.g. CBOR Web Token).

EDHOC with symmetric key authentication is illustrated in Figure 4.

```
Party U                                                       Party V
|                S_U, N_U, E_U, ALG_1, KID, APP_1                   |
+------------------------------------------------------------------>|
|                             message_1                             |
|                                                                   |
|         S_U, S_V, N_V, E_V, ALG_2, Enc(K_2; APP_2; aad_2)         |
|<------------------------------------------------------------------+
|                             message_2                             |
|                                                                   |
|                    S_V, Enc(K_3; APP_3; aad_3)                    |
+------------------------------------------------------------------>|
|                             message_3                             |
```

Figure 4: EDHOC with symmetric key authentication.

5.1.1. Mandatory to Implement Algorithms

For EDHOC authenticated with symmetric keys, the COSE algorithms ECDH-SS + HKDF-256 and AES-CCM-64-64-128 are mandatory to implement.

5.2. EDHOC Message 1

5.2.1. Formatting of Message 1

message_1 SHALL be a CBOR array as defined below

```
message_1 = [
  data_1
]

data_1 = (
  MSG_TYPE : int,
  S_U : bstr,
  N_U : bstr,
  E_U : serialized_COSE_Key,
  ECDH-Curves_U : alg_array,
  HKDFs_U : alg_array,
  AEADs_U : alg_array,
  KID : bstr,
  ? APP_1 : bstr
)

serialized_COSE_Key = bstr .cbor COSE_Key

alg_array = [ + alg : int / tstr ]
```

where:

- MSG_TYPE = 4

- S_U - variable length session identifier

- N_U - 64-bit random nonce

- E_U - the ephemeral public key of Party U

- ECDH-Curves_U - EC curves for ECDH which Party U supports, in the order of decreasing preference

- HKDFs_U - supported ECDH-SS w/ HKDF algorithms

- AEADs_U - supported AEAD algorithms

o KID - identifier of the pre-shared key

o APP_1 - bstr containing application data

5.2.2. Party U Processing of Message 1

Party U SHALL compose message_1 as follows:

o Determine which ECDH curve to use with Party V. If U previously received from Party V an error message to message_1 with diagnostic payload identifying an ECDH curve in ECDH-Curves_U, then U SHALL retrieve an ephemeral from that curve. Otherwise the first curve in ECDH-Curves_U MUST be used.

o Retrieve an ephemeral ECDH key pair generated as specified in Section 5 of [SP-800-56a] and format the ephemeral public key E_U as a COSE_key as specified in Section 3.1.

o Generate the pseudo-random nonce N_U

o Choose a session identifier S_U and store it for the length of the protocol.

o Format message_1 as specified in Section 5.2.1.

5.2.3. Party V Processing of Message 1

Party V SHALL process message_1 as follows:

o Verify (OPTIONAL) that N_U has not been received before.

o Verify that at least one of each kind of the proposed algorithms are supported.

o Verify that the ECDH curve used in E_U is supported, and that no prior curve in ECDH-Curves_U is supported.

If any verification step fails, Party V MUST send an EDHOC error message back, formatted as defined in Section 6.1, and the protocol MUST be discontinued. If V does not support the ECDH curve used in E_U, but supports another ECDH curves in ECDH-Curves_U, then the error message SHOULD include a diagnostic payload describing the first supported ECDH curve in ECDH-Curves_U.

5.3. EDHOC Message 2

5.3.1. Formatting of Message 2

message_2 SHALL be a CBOR array as defined below

```
message_2 = [
  data_2,
  COSE_ENC_2 : COSE_Encrypt0
]

data_2 = (
  MSG_TYPE : int,
  S_U : bstr,
  S_V : bstr,
  N_V : bstr,
  E_V : serialized_COSE_Key,
  HKDF_V : int / tstr,
  AEAD_V : int / tstr
)
```

aad_2, in diagnostic non-normative notation, is:

```
aad_2 = message_1 | [ data_2 ]
```

where:

o MSG_TYPE = 5

o S_V - variable length session identifier

o N_V - 64-bit random nonce

o E_V - the ephemeral public key of Party V

o HKDF_V - an single chosen algorithm from HKDFs_U

o AEAD_V - an single chosen algorithm from AEADs_U

o COSE_ENC_2 has the following fields and values:

   * external_aad = aad_2

   * plaintext = ? APP_2

o APP_2 - bstr containing application data

o H() - the hash function in HKDF_V

5.3.2. Party V Processing of Message 2

Party V SHALL compose message_2 as follows:

- Retrieve an ephemeral ECDH key pair generated as specified in Section 5 of [SP-800-56a] using same curve as used in E_U. Format the ephemeral public key E_V as a COSE_key as specified in Section 3.1.

- Generate the pseudo-random nonce N_V

- Choose a session identifier S_V and store it for the length of the protocol.

- Select HKDF_V and AEAD_V from the algorithms proposed in HKDFs_U and AEADs_U.

- Format message_2 as specified in Section 5.3.1 where COSE_Encrypt0 is computed as defined in section 5.3 of [I-D.ietf-cose-msg], with AEAD_V, K_2, and IV_2.

5.3.3. Party U Processing of Message 2

Party U SHALL process message_2 as follows:

- Use the session identifier S_U to retrieve the protocol state.

- Verify message_2 as specified in Section 5.3.1 where COSE_Encrypt0 is decrypted defined in section 5.3 of [I-D.ietf-cose-msg], with AEAD_V, K_2, and IV_2.

If any verification step fails, Party U MUST send an EDHOC error message back, formatted as defined in Section 6.1, and the protocol MUST be discontinued.

5.4. EDHOC Message 3

5.4.1. Formatting of Message 3

message_3 SHALL be a CBOR array as defined below

```
message_3 = [
  data_3,
  COSE_ENC_3 : COSE_Encrypt0
]

data_3 = (
  MSG_TYPE : int,
  S_V : bstr
)
```

aad_3, in diagnostic non-normative notation, is:

```
aad_3 = H( message_1 | message_2 ) | [ data_3 ]
```

where:

- MSG_TYPE = 6
- COSE_ENC_3 has the following fields and values:
  - external_aad = aad_3
  - plaintext = ? APP_3
- APP_3 - bstr containing application data

5.4.2. Party U Processing of Message 3

Party U SHALL compose message_3 as follows:

- Format message_3 as specified in Section 5.4.1 where COSE_Encrypt0 is computed as defined in section 5.3 of [I-D.ietf-cose-msg], with AEAD_V, K_3, and IV_3.

5.4.3. Party V Processing of Message 3

Party V SHALL process message_3 as follows:

- Use the session identifier S_V to retrieve the protocol state.
- Verify message_3 as specified in Section 5.4.1 where COSE_Encrypt0 is decrypted and verified as defined in section 5.3 of [I-D.ietf-cose-msg], with AEAD_V, K_3, and IV_3.

If any verification step fails, Party V MUST send an EDHOC error message back, formatted as defined in Section 6.1, and the protocol MUST be discontinued.

6. Error Handling

6.1. Error Message Format

This section defines a message format for an EDHOC error message, used during the protocol. This is an error on EDHOC level and is independent of the transport layer used. An advantage of using such a construction is to avoid issues created by usage of cross protocol proxies (e.g. UDP to TCP).

error SHALL be a CBOR array as defined below

```
error = [
  MSG_TYPE : int,
  ? ERR_MSG : tstr
]
```

where:

- MSG_TYPE = 0

- ERR_MSG is an optional text string containing the diagnostic payload, defined in the same way as in Section 5.5.2 of [RFC7252].

7. IANA Considerations

7.1. Media Types Registry

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

Type name: application

Subtype name: edhoc

Required parameters: N/A

Optional parameters: N/A

Encoding considerations: binary

Security considerations: See Section 7 of this document.

Interoperability considerations: N/A

Published specification: [[this document]] (this document)

Applications that use this media type: To be identified

Fragment identifier considerations: N/A

Additional information:

* Magic number(s): N/A

* File extension(s): N/A

* Macintosh file type code(s): N/A

Person & email address to contact for further information:
Goeran Selander <goran.selander@ericsson.com>

Intended usage: COMMON

Restrictions on usage: N/A

Author: Goeran Selander <goran.selander@ericsson.com>

Change Controller: IESG

8. Security Considerations

EDHOC builds on the SIGMA-I family of theoretical protocols that provides perfect forward secrecy and identity protection with a minimal number of messages. The encryption algorithm of the SIGMA-I protocol provides identity protection, but the security of the protocol requires the MAC to cover the identity of the signer. Hence the message authenticating functionality of the authenticated encryption in EDHOC is critical: authenticated encryption MUST NOT be

replaced by plain encryption only, even if authentication is provided at another level or through a different mechanism.

EDHOC adds an explicit message type and expands the message authentication coverage to additional elements such as algorithms, application data, and previous messages. EDHOC uses the same Sign-then-MAC approach as TLS 1.3.

EDHOC does not include negotiation of parameters related to the ephemeral key, but it enables Party V to verify that the ECDH curve used in the protocol is the most preferred curve by U which is supported by both U and V.

Party U and V must make sure that unprotected data and metadata do not reveal any sensitive information. This also applies for encrypted data sent to an unauthenticated party. In particular, it applies to APP_1 and APP_2 in the asymmetric case, and APP_1 and KID in the symmetric case. The communicating parties may therefore anonymize KID.

Using the same KID or unprotected application data in several EDHOC sessions allows passive eavesdroppers to correlate the different sessions. Another consideration is that the list of supported algorithms may be used to identify the application.

Party U and V must make sure that unprotected data does not trigger any harmful actions. In particular, this applies to APP_1 in the asymmetric case, and APP_1 and KID in the symmetric case. Party V should be aware that replays of EDHOC message_1 cannot be detected unless previous nonces are stored.

The availability of a secure pseudorandom number generator and truly random seeds are essential for the security of EDHOC. If no true random number generator is available, a truly random seed must be provided from an external source. If ECDSA is supported, "deterministic ECDSA" as specified in RFC6979 is RECOMMENDED.

Nonces MUST NOT be reused, both parties MUST generate fresh random nonces.

Ephemeral keys SHOULD NOT be reused, both parties SHOULD generate fresh random ephemeral key pairs. Party V MAY reuse the ephemeral key to limit the effect of certain DoS attacks. For example, to reduce processing costs in the case of repeated uncompleted protocol runs, party V MAY pre-compute its ephemeral key E_V and reuse it for a small number of concurrent EDHOC executions, for example until a number of EDHOC protocol instances has been successfully completed,

which triggers party V to pre-compute a new ephemeral key E_V to use with subsequent protocol runs.

The referenced processing instructions in [SP-800-56a] must be complied with, including deleting the intermediate computed values along with any ephemeral ECDH secrets after the key derivation is completed.

Party U and V are responsible for verifying the integrity of certificates. The selection of trusted CAs should be done very carefully and certificate revocation should be supported.

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

Note that, depending on the application, the keys established through the EDHOC protocol will need to be renewed, in which case the communicating parties need to run the protocol again.

Implementations should provide countermeasures to side-channel attacks such as timing attacks.

9. Acknowledgments

The authors want to thank Jim Schaad for reviewing intermediate versions and for contributing many concrete proposals incorporated in this version. We are also greatful to Ilari Liusvaara and Ludwig Seitz for reviewing previous versions of the draft.

TODO: This section should be after Appendixes and before Author's address according to RFC7322.

10. References

10.1. Normative References

[I-D.ietf-cose-msg]
Schaad, J., "CBOR Object Signing and Encryption (COSE)", draft-ietf-cose-msg-24 (work in progress), November 2016.

[I-D.schaad-cose-x509]
Schaad, J., "CBOR Encoded Message Syntax (COSE): Headers for carrying and referencing X.509 certificates", draft-schaad-cose-x509-00 (work in progress), November 2016.

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

[RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <http://www.rfc-editor.org/info/rfc7049>.

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

[SP-800-56a] Barker, E., Chen, L., Roginsky, A., and M. Smid, "Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography", NIST Special Publication 800-56A Revision 2, May 2013, <http://dx.doi.org/10.6028/NIST.SP.800-56Ar2>.

10.2. Informative References

[I-D.greevenbosch-appsawg-cbor-cddl] Birkholz, H., Vigano, C., and C. Bormann, "CBOR data definition language (CDDL): a notational convention to express CBOR data structures", draft-greevenbosch-appsawg-cbor-cddl-10 (work in progress), March 2017.

[I-D.hartke-core-e2e-security-reqs] Selander, G., Palombini, F., and K. Hartke, "Requirements for CoAP End-To-End Security", draft-hartke-core-e2e-security-reqs-02 (work in progress), January 2017.

[I-D.ietf-ace-oauth-authz] Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "Authentication and Authorization for Constrained Environments (ACE)", draft-ietf-ace-oauth-authz-06 (work in progress), March 2017.

[I-D.ietf-core-object-security] Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security of CoAP (OSCOAP)", draft-ietf-core-object-security-02 (work in progress), March 2017.

[I-D.ietf-core-resource-directory]
Shelby, Z., Koster, M., Bormann, C., and P. Stok, "CoRE Resource Directory", draft-ietf-core-resource-directory-10 (work in progress), March 2017.

[I-D.seitz-ace-oscoap-profile]
Seitz, L. and F. Palombini, "OSCOAP profile of ACE", draft-seitz-ace-oscoap-profile-01 (work in progress), October 2016.

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

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

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

Appendix A. Test Vectors

TODO: This section needs to be updated.

Appendix B. PSK Chaining

An application using EDHOC with symmetric keys may have a security policy to change the PSK as a result of successfully completing the EDHOC protocol. In this case, the old PSK SHALL be replaced with a new PSK derived using other = exchange_hash, AlgorithmID = "EDHOC PSK Chaining" and keyDataLength equal to the key length of AEAD_V, see Section 3.2.

Appendix C. EDHOC with CoAP and OSCOAP

C.1. Transferring EDHOC in CoAP

EDHOC can be transferred as an exchange of CoAP [RFC7252] messages, with the CoAP client as party U and the CoAP server as party V. By default EDHOC is sent to the Uri-Path: "/.well-known/edhoc", but an application may define its own path that can be discorvered e.g. using resource directory [I-D.ietf-core-resource-directory].

In practice, EDHOC message_1 is sent in the payload of a POST request from the client to the server's resource for EDHOC. EDHOC message_2 or the EDHOC error message is sent from the server to the client in the payload of a 2.04 Changed response. EDHOC message_3 or the EDHOC error message is sent from the client to the server's resource in the payload of a POST request. If needed, an EDHOC error message is sent from the server to the client in the payload of a 2.04 Changed response

An example of successful EDHOC exchange using CoAP is shown in Figure 5.

```
           Client    Server
             |          |
             +--------->| Header: POST (Code=0.02)
             |   POST   | Uri-Path: "/.well-known/edhoc"
             |          | Content-Type: application/edhoc
             |          | Payload: EDHOC message_1
             |          |
             |<---------+ Header: 2.04 Changed
             |   2.04   | Content-Type: application/edhoc
             |          | Payload: EDHOC message_2
             |          |
             +--------->| Header: POST (Code=0.02)
             |   POST   | Uri-Path: "/.well-known/edhoc"
             |          | Content-Type: application/edhoc
             |          | Payload: EDHOC message_3
             |          |
             |<---------+ Header: 2.04 Changed
             |   2.04   |
             |          |
```

Figure 5: Transferring EDHOC in CoAP

C.2. Deriving an OSCOAP context from EDHOC

When EDHOC is use to derive parameters for OSCOAP [I-D.ietf-core-object-security], the parties must make sure that the EDHOC session identifiers are unique Recipient IDs in OSCOAP. In case that the CoAP client is party U and the CoAP server is party V:

o The AEAD Algorithm is AEAD_V, as defined in this document

o The KDF algorithm is HKDF_V, as defined in this document

o The Client's Sender ID is S_V, as defined in this document

o The Server's Sender ID is S_U, as defined in this document

- o The Master Secret is derived as specified in Section 3.2 of this document, with other = exchange_hash, AlgorithmID = "EDHOC OSCOAP Master Secret" and keyDataLength equal to the key length of AEAD_V.

- o The Master Salt is derived as specified in Section 3.2 of this document, with other = exchange_hash, AlgorithmID = "EDHOC OSCOAP Master Salt" and keyDataLength equal to 64 bits.

Authors' Addresses

Goeran Selander
Ericsson AB
Faeroegatan 6
Kista SE-164 80 Stockholm
Sweden

Email: goran.selander@ericsson.com

John Mattsson
Ericsson AB
Faeroegatan 6
Kista SE-164 80 Stockholm
Sweden

Email: john.mattsson@ericsson.com

Francesca Palombini
Ericsson AB
Faeroegatan 6
Kista SE-164 80 Stockholm
Sweden

Email: francesca.palombini@ericsson.com