Compact TLS 1.3
draft-rescorla-tls-ctls-02

Abstract

This document specifies a "compact" version of TLS 1.3. It is isomorphic to TLS 1.3 but saves space by aggressive use of defaults and tighter encodings. CTLS is not interoperable with TLS 1.3, but it should eventually be possible for the server to distinguish TLS 1.3 and CTLS handshakes.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on January 9, 2020.

Copyright Notice

Copyright (c) 2019 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 Simplified BSD License text as described in Section 4.e of
the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction ........................................... 2
2. Conventions and Definitions .......................... 3
3. Common Primitives ...................................... 3
   3.1. Varints ........................................ 3
   3.2. Record Layer ..................................... 4
   3.3. Handshake Layer .................................. 4
   3.4. Extensions ....................................... 5
4. Handshake Messages .................................... 5
   4.1. ClientHello ...................................... 5
      4.1.1. KeyShare .................................... 6
   4.2. ServerHello ...................................... 7
      4.2.1. KeyShare .................................... 7
      4.2.2. PreSharedKeys ................................ 8
   4.3. EncryptedExtensions ............................... 8
   4.4. CertificateRequest ............................... 8
      4.5. Certificate .................................... 8
         4.5.1. Key IDs .................................... 9
         4.5.2. CertificateVerify ........................... 9
         4.5.3. Finished .................................... 9
         4.5.4. HelloRetryRequest ......................... 10
   5. Handshake Size Calculations ....................... 10
      5.1. ECDHE w/ Signatures ............................ 10
         5.1.1. Flight 1 (ClientHello) *** ............... 10
         5.1.2. Flight 2 (ServerHello..Finished) .......... 10
         5.1.3. Flight 3 (Client Certificate..Finished) .. 11
   6. cTLS as Compression Layer [[OPEN ISSUE]] ........... 12
7. Security Considerations .............................. 13
8. IANA Considerations .................................. 13
9. Normative References ................................. 13
Acknowledgments .......................................... 14
Authors’ Addresses ....................................... 14

1. Introduction

DISCLAIMER: This is a work-in-progress draft of cTLS and has not yet seen significant security analysis, so could contain major errors. It should not be used as a basis for building production systems.

This document specifies a "compact" version of TLS 1.3 [RFC8446]. It is isomorphic to TLS 1.3 but designed to take up minimal bandwidth. The space reduction is achieved by two basic techniques:
Default values for common configurations, thus avoiding the need to take up space on the wire.

More compact encodings, omitting unnecessary values.

For the common (EC)DHE handshake with (EC)DHE and pre-established public keys, CTLS achieves an overhead of [TODO] bytes over the minimum required by the cryptovariables.

Because cTLS is semantically equivalent to TLS, it can be viewed either as a related protocol or as a compression mechanism. Specifically, it can be implemented by a layer between the TLS handshake state machine and the record layer. See Section 6 for more details.

2. Conventions and Definitions

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] when, and only when, they appear in all capitals, as shown here.

Structure definitions listed below override TLS 1.3 definitions; any PDU not internally defined is taken from TLS 1.3.

3. Common Primitives

3.1. Varints

CTLS makes use of variable-length integers in order to allow a wide integer range while still providing for a minimal encoding. The width of the integer is encoded in the first two bits of the field as follows, with xs indicating bits that form part of the integer.

<table>
<thead>
<tr>
<th>Bit pattern</th>
<th>Length (bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xxxxxxx</td>
<td>1</td>
</tr>
<tr>
<td>10xxxxxx xxxxxxxx</td>
<td>2</td>
</tr>
<tr>
<td>11xxxxxx xxxxxxxx xxxxxxxx</td>
<td>3</td>
</tr>
</tbody>
</table>
Thus, one byte can be used to carry values up to 127.

In the TLS syntax variable integers are denoted as "varint" and a vector with a top range of a varint is denoted as:

    opaque foo<1..V>;

[[OPEN ISSUE: Should we just re-encode this directly in CBOR?. That might be easier for people, but I ran out of time.]]

3.2. Record Layer

The CTLS Record Layer assumes that records are externally framed (i.e., that the length is already known because it is carried in a UDP datagram or the like). Depending on how this was carried, you might need another byte or two for that framing. Thus, only the type byte need be carried. Thus, TLSPlaintext becomes:

    struct {
        ContentType type;
        opaque fragment[TLSPlaintext.length];
    } TLSPlaintext;

In addition, because the epoch is known in advance, the dummy content type is not needed for the ciphertext, so TLSCiphertext becomes:

    struct {
        opaque content[TLSPlaintext.length];
        ContentType type;
        uint8 zeros[length_of_padding];
    } TLSInnerPlaintext;

    struct {
        opaque encrypted_record[TLSCiphertext.length];
    } TLSCiphertext;

Note: The user is responsible for ensuring that the sequence numbers/ nonces are handled in the usual fashion.

Overhead: 1 byte per record.

3.3. Handshake Layer

The CTLS handshake layer is the same as the TLS 1.3 handshake layer except that the length is a varint.
struct {
    HandshakeType msg_type;    /* handshake type */
    varint length;             // CHANGED
    select (Handshake.msg_type) {
        case client_hello:          ClientHello;
        case server_hello:          ServerHello;
        case end_of_early_data:     EndOfEarlyData;
        case encrypted_extensions:  EncryptedExtensions;
        case certificate_request:   CertificateRequest;
        case certificate:           Certificate;
        case certificate_verify:    CertificateVerify;
        case finished:              Finished;
        case new_session_ticket:    NewSessionTicket;
        case key_update:            KeyUpdate;
    }
};
} Handshake;

Overhead: 2 bytes per handshake message (min).

[OPEN ISSUE: This can be shrunk to 1 byte in some cases if we are
willing to use a custom encoding. There are 11 handshake types, so
we can use the first 4 bits for the type and then the bottom 4 bits
for an encoding of the length, but we would have to offset that by 16
or so to be able to have a meaningful impact.]

3.4. Extensions

CTLS Extensions are the same as TLS 1.3 extensions, except varint
length coded:

struct {
    ExtensionType extension_type;
    opaque extension_data<0..V>;
} Extension;

4. Handshake Messages

In general, we retain the basic structure of each individual TLS
handshake message. However, the following handshake messages are
slightly modified for space reduction.

4.1. ClientHello

The CTLS ClientHello is as follows.
uint8 ProtocolVersion; // 1 byte
opaque Random[16]; // shortened
uint8 CipherSuite; // 1 byte

struct {
    ProtocolVersion versions<0..255>;
    Random random;
    CipherSuite cipher_suites<1..V>;
    Extension extensions[remainder_of_message];
} ClientHello;

[[TODO: Define single-byte mappings of the cipher suites and protocol version.]]

The versions list from "supported_versions" has moved into ClientHello.versions with versions being one byte, but with the modern semantics of the client offering N versions and the server picking one.

In order to conserve space, the following extensions have default values which apply if they are not present:

- SignatureAlgorithms: ed25519
- SupportedGroups: the list of groups present in the KeyShare extension.
- Pre-Shared Key Exchange Modes: psk_dhe_ke
- Certificate Type: A new TBD value indicating a key index.

As a practical matter, the only extension needed is the KeyShare extension, as defined below.

Overhead: 8 bytes (min)

- Versions: 1 + # Versions
- CipherSuites: 1 + # Suites
- Key shares: 2 + 2 * # shares

4.1.1. KeyShare

The KeyShare extension is redefined as:
uint8 NamedGroup;
struct {
    NamedGroup group;
    opaque key_exchange<1..V>;
} KeyShareEntry;

struct {
    KeyShareEntry client_shares[length of extension];
} KeyShareClientHello;

[[TODO: Need a mapping for 8-bit group ids]]

4.2. ServerHello

We redefine ServerHello in a similar way:

struct {
    ProtocolVersion version;
    Random random;
    CipherSuite cipher_suite;
    Extension extensions[remainder_of_message];
} ServerHello;

The extensions have the same default values as in ClientHello, so as a practical matter only KeyShare is needed.

Overhead: 6 bytes

- Version: 1
- Cipher Suite: 1
- KeyShare: 4 bytes

4.2.1. KeyShare

struct {
    KeyShareEntry server_share;
} KeyShareServerHello;

[[OPEN ISSUE: We could save one byte here by removing the length of the key share and another byte by only allowing the client to send one key share (so group wasn’t needed).]]

[[TODO: Need to define a single-byte list of NamedGroups]].
4.2.2. PreSharedKeys

[[TODO]]

4.3. EncryptedExtensions

Unchanged.

[[OPEN ISSUE: We could save 2 bytes in handshake header by omitting this value when it’s unneeded.]]

4.4. CertificateRequest

This message removes the certificate_request_context and re-encodes the extensions.

```c
struct {
    Extension extensions[remainder of message];
} CertificateRequest;
```

4.5. Certificate

We can slim down the Certificate message somewhat.

```c
eenum {
    X509(0),
    RawPublicKey(2),
    (255)
} CertificateType;

struct {
    select (certificate_type) {
    case RawPublicKey:
        /* From RFC 7250 ASN.1_subjectPublicKeyInfo */
        opaque ASN1_subjectPublicKeyInfo<1..V>;
    case X509:
        opaque cert_data<1..V>;
    }
    Extension extensions<0..V>;
} CertificateEntry;

struct {
    CertificateEntry certificate_list[rest of extension];
} Certificate;
```

For a single certificate, this message will have a minimum of 2 bytes of overhead for the two length bytes.
4.5.1. Key IDs

WARNING: This is a new feature which has not seen any analysis and so may have real problems.

It may also be possible to slim down the Certificate message further, by adding a KeyID-based mode, in which they keys were just a table index. This would redefine Certificate as:

```c
struct {
    varint key_id;
} KeyIdCertificate;

struct {
    select (certificate_type):
    case RawPublicKey, x509:
        CertificateEntry certificate_list<0..2^24-1>;
    case key_id:
        KeyIdCertificate;
    }
} Certificate;
```

This allows the use of a short key id. Note that this is orthogonal to the rest of the changes.

IMPORTANT: You really want to include the certificate in the handshake transcript somehow, but this isn’t specified for how.

4.5.2. CertificateVerify

Remove the signature algorithm and assume it’s tied to the key. Note that this does not work for RSA keys, but if we just decide to be EC only, it works fine.

```c
struct {
    opaque signature[rest of message];
} CertificateVerify;
```

4.5.3. Finished

Unchanged.
4.5.4. HelloRetryRequest

[[TODO]]

5. Handshake Size Calculations

This section provides the size of cTLS handshakes with various parameters [[TODO: Fill this out with more options.]]

5.1. ECDHE w/ Signatures

We compute the total flight size with X25519 and P-256 signatures, thus the keys are 32-bytes long and the signatures 64 bytes, with a cipher with an 8 byte auth tag, as in AEAD_AES_128_CCM_8. [Note: GCM should not be used with a shortened tag.] Overhead estimates marked with *** have been verified with Mint. Others are hand calculations and so may prove to be approximate.

5.1.1. Flight 1 (ClientHello) ***
  o Random: 16
  o KeyShare: 32
  o Message Overhead: 8
  o Handshake Overhead: 2
  o Record Overhead: 1
  o Total: 59

5.1.2. Flight 2 (ServerHello..Finished)

ServerHello ***
  o Random: 16
  o KeyShare: 32
  o Message Overhead: 6
  o Handshake Overhead: 2
  o Total: 56

EncryptedExtensions ***
CertificateRequest

Certificate

CertificateVerify

Finished

Record Overhead: 2 bytes (2 records) + 8 bytes (auth tag).

[[OPEN ISSUE: We’ll actually need a length field for the ServerHello, to separate it from the ciphertext.]]

Total Size: 175 + X bytes.

5.1.3. Flight 3 (Client Certificate..Finished)

Certificate

Certificate: X
The above text treats cTLS as a new protocol; however it is also possible to view it as a form of compression for TLS, which sits in between the handshake layer and the record layer, like so:

```
+---------------+---------------+---------------+
|   Handshake   |  Application  |     Alert     |
+---------------+---------------+---------------+
|               cTLS Compression Layer          |
+---------------+---------------+---------------+
|               cTLS Record Layer               |
```

This structure does involve one technical difference: because the handshake message transformation happens below the handshake layer, the cTLS handshake transcript would be the same as the TLS 1.3 handshake transcript. This has both advantages and disadvantages.

The major advantage is that it makes it possible to reuse all the TLS security proofs even with very aggressive compression (with suitable proofs about the bijectiveness of the compression). [Thanks to
Karthik Bhargavan for this point.) This probably also makes it easier to implement more aggressive compression. For instance, the above text shrinks the handshake headers but does not elide them entirely. If the handshake shape (i.e., which messages are sent) is known in advance, then these headers can be removed, thus trimming about 20 bytes from the handshake. This is easier to reason about as a form of compression. With somewhat aggressive parameters, including predetermined cipher suites, this technique can bring the handshake (without record overhead) to:

- Client's first flight: 48
- Server's first flight: 164
- Client's second flight: 116

The major potential disadvantage of a compression approach is that it makes cTLS and TLS handshakes confusable. For instance, an attacker who obtained the handshake keys might be able to undetectably transform a cTLS <- TLS connection into a TLS <-> TLS connection. This is easily dealt with by modifying the transcript, e.g., by injecting a cTLS extension in the transcript (though not into cTLS wire format).

7. Security Considerations

WARNING: This document is effectively brand new and has seen no analysis. The idea here is that cTLS is isomorphic to TLS 1.3, and therefore should provide equivalent security guarantees, modulo use of new features such as KeyID certificate messages.

One piece that is a new TLS 1.3 feature is the addition of the key_id, which definitely requires some analysis, especially as it looks like a potential source of identity misbinding. This is entirely separable from the rest of the specification. The compression version would also need further analysis.

8. IANA Considerations

This document has no IANA actions.

9. Normative References

Acknowledgments

TODO acknowledge.

Authors’ Addresses

Eric Rescorla
Mozilla
Email: ekr@rtfm.com

Richard Barnes
Cisco
Email: rlb@ipv.sx
Abstract

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

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 12, 2019.

Copyright Notice

Copyright (c) 2019 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 Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction ........................................... 3
   1.1. Rationale for EDHOC .............................. 4
   1.2. Terminology and Requirements Language ............... 5
2. Background ............................................. 5
3. EDHOC Overview ......................................... 7
   3.1. Cipher Suites ..................................... 8
   3.2. Ephemeral Public Keys ............................. 9
   3.3. Key Derivation ................................... 9
4. EDHOC Authenticated with Asymmetric Keys .................. 11
   4.1. Overview ......................................... 11
   4.2. EDHOC Message 1 ................................ 13
   4.3. EDHOC Message 2 ................................ 15
   4.4. EDHOC Message 3 ................................ 18
5. EDHOC Authenticated with Symmetric Keys ................... 20
   5.1. Overview ......................................... 20
   5.2. EDHOC Message 1 ................................ 21
   5.3. EDHOC Message 2 ................................ 21
   5.4. EDHOC Message 3 ................................ 22
6. Error Handling ........................................... 22
   6.1. EDHOC Error Message ............................... 22
7. Transferring EDHOC and Deriving Application Keys .......... 24
   7.1. Transferring EDHOC in CoAP ........................ 24
   7.2. Transferring EDHOC over Other Protocols ............. 27
8. IANA Considerations ...................................... 27
   8.1. EDHOC Cipher Suites Registry ..................... 27
   8.2. EDHOC Method Type Registry ....................... 27
   8.3. The Well-Known URI Registry ...................... 27
   8.4. Media Types Registry .............................. 27
   8.5. CoAP Content-Formats Registry .................... 28
9. Security Considerations .................................. 29
   9.1. Security Properties ............................... 29
   9.2. Cryptographic Considerations ..................... 30
   9.3. Mandatory to Implement Cipher Suite ............... 30
   9.4. Unprotected Data .................................. 31
   9.5. Denial-of-Service ................................ 31
   9.6. Implementation Considerations .................... 31
   9.7. Other Documents Referencing EDHOC ................. 32
10. References ............................................ 32
    10.1. Normative References ............................. 32
    10.2. Informative References ........................... 34
Appendix A. Use of CBOR, CDDL and COSE in EDHOC ............ 36
1. Introduction

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

In order for a communication session to provide forward secrecy, the communicating parties can run an Elliptic Curve Diffie-Hellman (ECDH) key exchange protocol with ephemeral keys, from which shared key material can be derived. This document specifies Ephemeral Diffie-Hellman Over COSE (EDHOC), a lightweight key exchange protocol providing perfect forward secrecy and identity protection. Authentication is based on credentials established out of band, e.g. from a trusted third party, such as an Authorization Server as specified by [I-D.ietf-ace-oauth-authz]. EDHOC supports authentication using pre-shared keys (PSK), raw public keys (RPK), and public key certificates. After successful completion of the EDHOC protocol, application keys and other application specific data can be derived using the EDHOC-Exporter interface. A main use case for EDHOC is to establish an OSCORE security context. EDHOC uses COSE for cryptography, CBOR for encoding, and CoAP for transport. By reusing existing libraries, the additional code footprint can be kept very low. Note that this document focuses on authentication and key establishment: for integration with authorization of resource access, refer to [I-D.ietf-ace-oscore-profile].
EDHOC is designed to work in highly constrained scenarios making it especially suitable for network technologies such as Cellular IoT, 6TiSCH [I-D.ietf-6tisch-dtsecurity-zerotouch-join], LoRaWAN [LoRa1][LoRa2]. These network technologies are characterized by their low throughput, low power consumption, and small frame sizes. Compared to the DTLS 1.3 handshake [I-D.ietf-tls-dtls13] with ECDH and connection ID, the number of bytes in EDHOC is less than 1/4 when PSK authentication is used and less than 1/3 when RPK authentication is used, see Appendix B.

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

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

1.1. Rationale for EDHOC

Many constrained IoT systems today do not use any security at all, and when they do, they often do not follow best practices. One reason is that many current security protocols are not designed with constrained IoT in mind. Constrained IoT systems often deal with personal information, valuable business data, and actuators interacting with the physical world. Not only do such systems need security and privacy, they often need end-to-end protection with source authentication and perfect forward secrecy. EDHOC and OSCORE [I-D.ietf-core-object-security] enables security following current best practices to devices and systems where current security protocols are impractical.

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

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

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

1.2. Terminology and Requirements Language

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

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

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

2. Background

SIGMA (SIGn-and-MAC) is a family of theoretical protocols with a large number of variants [SIGMA]. Like IKEv2 and (D)TLS 1.3 [RFC8446], EDHOC is built on a variant of the SIGMA protocol which provide identity protection of the initiator (SIGMA-I), and like (D)TLS 1.3, EDHOC implements the SIGMA-I variant as Sign-then-MAC.
The SIGMA-I protocol using an authenticated encryption algorithm is shown in Figure 1.

![Diagram of authenticated encryption variant of the SIGMA-I protocol.](image)

Figure 1: Authenticated encryption variant of the SIGMA-I protocol.

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

- $X_U$ and $X_V$ are the ECDH ephemeral public keys of U and V, respectively.
- CRED_U and CRED_V are the credentials containing the public authentication keys of U and V, respectively.
- ID_CRED_U and ID_CRED_V are data enabling the recipient party to retrieve the credential of U and V, respectively.
- Sig(U; . ) and S(V; . ) denote signatures made with the private authentication key of U and V, respectively.
- AEAD(K; . ) denotes authenticated encryption with additional data using the key K derived from the shared secret. The authenticated encryption MUST NOT be replaced by plain encryption, see Section 9.

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

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

o Method types and error handling.

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, COSE, and CoAP libraries.

To simplify for implementors, the use of CBOR and COSE in EDHOC is summarized in Appendix A and example messages in CBOR diagnostic notation are given in Appendix B.

3. EDHOC Overview

EDHOC consists of three flights (message_1, message_2, message_3) that maps directly to the three messages in SIGMA-I, plus an EDHOC error message. All EDHOC messages consist of a sequence of CBOR encoded data items, where the first data item of message_1 is an int (TYPE) specifying the method (asymmetric, symmetric, error) and the correlation properties of the transport used. The messages may be viewed as a CBOR encoding of an indefinite-length array without the first and last byte, see Appendix A.1.

While EDHOC uses the COSE_Key, COSE_Sign1, and COSE_Encrypt0 structures, only a subset of the parameters is included in the EDHOC messages. After creating EDHOC message_3, Party U can derive symmetric application keys, and application protected data can therefore be sent in parallel with EDHOC message_3. The application may protect data using the algorithms (AEAD, HKDF, etc.) in the selected cipher suite and the connection identifiers (C_U, C_V). EDHOC may be used with the media type application/edhoc defined in Section 8.
Figure 2: EDHOC message flow

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

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

Cryptographically, EDHOC does not put requirements on the lower layers. EDHOC is not bound to a particular transport layer, and can be used in environments without IP. It is recommended to transport the EDHOC message in CoAP payloads, see Section 7. An implementation may support only Party U or only Party V.

3.1. Cipher Suites

EDHOC cipher suites consist of a set of COSE algorithms: an AEAD algorithm, an ECDH algorithm (including HKDF algorithm), an ECDH curve, a signature algorithm, and signature algorithm parameters. The signature algorithm is not used when EDHOC is authenticated with symmetric keys. Each cipher suite is either identified with a pre-
This document specifies two pre-defined cipher suites.

0. [ 12, -27, 4, -8, 6 ]
   (AES-CCM-64-64-128, ECDH-SS + HKDF-256, X25519, EdDSA, Ed25519)

1. [ 12, -27, 1, -7 ]
   (AES-CCM-64-64-128, ECDH-SS + HKDF-256, P-256, ES256)

Two additional numbers are registered for application defined cipher suites. Application defined cipher suites MUST only use algorithms specified for COSE, are not interoperable with other deployments and can therefore only be used in local networks.

-24. First application defined cipher suite.
-23. Second application defined cipher suite.

3.2. Ephemeral Public Keys

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

3.3. Key Derivation

Key and IV derivation SHALL be performed as specified in Section 11 of [RFC8152] with the following input:

- The KDF SHALL be the HKDF [RFC5869] in the selected cipher suite.
- The secret (Section 11.1 of [RFC8152]) SHALL be the ECDH shared secret as defined in Section 12.4.1 of [RFC8152].
- The salt (Section 11.1 of [RFC8152]) SHALL be the PSK when EDHOC is authenticated with symmetric keys, and the empty byte string when EDHOC is authenticated with asymmetric keys. The secret and the salt are both inputs to the HKDF extract stage and the PSK is used as ‘salt’ to simplify implementation. Note that [RFC5869]
specifies that if the salt is not provided, it is set to a string of zeros (see Section 2.2 of [RFC5869]). For implementation purposes, not providing the salt is the same as setting the salt to the empty byte string.

- The fields in the context information COSE_KDF_Context (Section 11.2 of [RFC8152]) SHALL have the following values:
  * AlgorithmID is an int or tstr, see below
  * PartyUInfo = PartyVInfo = (null, null, null)
  * keyDataLength is a uint, see below
  * protected SHALL be a zero length bstr
  * other is a bstr and SHALL be one of the transcript hashes TH_2, TH_3, or TH_4 as defined in Sections 4.3.1, 4.4.1, and 3.3.1.
  * SuppPrivInfo is omitted

We define EDHOC-Key-Derivation to be the function which produces the output as described in [RFC5869] and [RFC8152] depending on the variable input AlgorithmID, keyDataLength, and other:

\[
\text{output} = \text{EDHOC-Key-Derivation(AlgorithmID, keyDataLength, other)}
\]

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

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

3.3.1. EDHOC-Exporter Interface

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

\[
\text{EDHOC-Exporter(label, length)} = \\
\text{EDHOC-Key-Derivation(label, 8 * length, TH_4)}
\]
The output of the EDHOC-Exporter function SHALL be derived using other = TR_4, AlgorithmID = label, and keyDataLength = 8 * length, where label is a tstr defined by the application and length is a uint defined by the application. The label SHALL be different for each different exporter value. The transcript hash TR_4, in non-CDDL notation, is:

\[
TH_4 = H(bstr .cborseq [ TH_3, CIPHERTEXT_3 ])
\]

where \( H() \) is the hash function in the HKDF, which takes a CBOR byte string (bstr) as input and produces a CBOR byte string as output. The use of \'.cborseq’ is exemplified in Appendix A.1. Example use of the EDHOC-Exporter is given in Sections 3.3.2 and 7.1.1.

3.3.2. EDHOC PSK Chaining

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

\[
\begin{align*}
PSK &= \text{EDHOC-Exporter("EDHOC Chaining PSK", length)} \\
ID_PSK &= \text{EDHOC-Exporter("EDHOC Chaining ID_PSK", 4)}
\end{align*}
\]

4. EDHOC Authenticated with Asymmetric Keys

4.1. Overview

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

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

where the identifiers ID_CRED_U and ID_CRED_V are COSE header maps containing COSE header parameter that can identify a public authentication key, see Appendix A.2. In the following we give some examples of possible COSE header parameters.
Raw public keys are most optimally stored as COSE_Key objects and identified with a ‘kid’ parameter (see [RFC8152]):

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

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

- by a hash value with the ‘x5t’ parameter;
  * ID_CRED_x = { TBD1 : COSE_CertHash }, for x = U or V,
- by a URL with the ‘x5u’ parameter;
  * ID_CRED_x = { TBD2 : uri }, for x = U or V,
- or by a bag of certificates with the ‘x5bag’ parameter;
  * ID_CRED_x = { TBD3 : COSE_X509 }, for x = U or V.
- by a certificate chain with the ‘x5chain’ parameter;
  * ID_CRED_x = { TBD4 : COSE_X509 }, for x = U or V,

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

The actual credentials CRED_U and CRED_V (e.g. a COSE_Key or a single X.509 certificate) are signed by party U and V, respectively to prevent duplicate-signature key selection (DSKS) attacks, see Section 4.4.1 and Section 4.3.1. Party U and Party V MAY use different types of credentials, e.g. one uses RPK and the other uses certificate.

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

The first data item of message_1 is an int TYPE = 4 * method + corr specifying the method and the correlation properties of the transport used. corr = 0 is used when there is no external correlation mechanism. corr = 1 is used when there is an external correlation mechanism (e.g. the Token in CoAP) that enables Party U to correlate message_1 and message_2. corr = 2 is used when there is an external correlation mechanism that enables Party V to correlate message_2 and message_3. corr = 3 is used when there is an external correlation mechanism that enables the parties to correlate all the messages. The use of the correlation parameter is exemplified in Section 7.1.

EDHOC with asymmetric key authentication is illustrated in Figure 3.

Party U

| TYPE, SUITES_U, X_U, C_U, UAD_1 |
+-----------------------------------
| message_1                         |
| C_U, X_V, C_V, AEAD(K_2; ID_CRED_V, Sig(V; CRED_V, TH_2), UAD_2) |
| message_2                          |
| C_V, AEAD(K_3; ID_CRED_U, Sig(U; CRED_U, TH_3), PAD_3) |
| message_3                          |

Figure 3: Overview of EDHOC with asymmetric key authentication.

4.2. EDHOC Message 1

4.2.1. Formatting of Message 1

message_1 SHALL be a sequence of CBOR data items (see Appendix A.1) as defined below

message_1 = {
  TYPE : int,
  SUITES_U : suite / [ index: uint, 2* suite ],
  X_U : bstr,
  C_U : bstr,
  ? UAD_1 : bstr,
}

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

SUITES_U - cipher suites which Party U supports, in order of decreasing preference. If a single cipher suite is conveyed, a single suite is used, if multiple cipher suites are conveyed, an array of suites and an index is used. The zero-based index (i.e. 0 for the first, 1 for the second, etc.) identifies a single selected cipher suite from the array.

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

C_U - variable length connection identifier

UAD_1 - bstr containing unprotected opaque application data

4.2.2. Party U Processing of Message 1

Party U SHALL compose message_1 as follows:

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

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

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

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

- Format message_1 as the sequence of CBOR data items specified in Section 4.2.1 and encode it to a byte string (see Appendix A.1).
4.2.3. Party V Processing of Message 1

Party V SHALL process message_1 as follows:

- Decode message_1 (see Appendix A.1).
- Verify that the selected cipher suite is supported and that no prior cipher suites in SUITES_U are supported.
- Validate that there is a solution to the curve definition for the given x-coordinate X_U.
- Pass UAD_1 to the application.

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

4.3. EDHOC Message 2

4.3.1. Formatting of Message 2

message_2 SHALL be a sequence of CBOR data items (see Appendix A.1) as defined below

message_2 = (
  data_2,
  CIPHERTEXT_2 : bstr,
)

data_2 = (
  ? C_U : bstr,
  X_V : bstr,
  C_V : bstr,
)

TH_2 : bstr

where the transcript hash TH_2, in non-CDDL notation, is:

TH_2 = H( bstr .cborseq [ message_1, data_2 ] )

where:
4.3.2. Party V Processing of Message 2

Party V SHALL compose message_2 as follows:

- If TYPE mod 4 equals 1 or 3, C_U is omitted, otherwise C_U is not omitted.
- Generate an ephemeral ECDH key pair as specified in Section 5 of [SP-800-56A] using the curve in the selected cipher suite. Let X_V be the x-coordinate of the ephemeral public key.
- Choose a connection identifier C_V and store it for the length of the protocol.
- Compute COSE_Sign1 as defined in Section 4.4 of [RFC8152], using the signature algorithm in the selected cipher suite, the private authentication key of Party V, and the following parameters (further clarifications in Appendix A.2.2). The unprotected header (not included in the EDHOC message) MAY contain parameters (e.g. ‘alg’).
  * protected = bstr .cbor ID_CRED_V
  * payload = CRED_V
  * external_aad = TH_2
  * ID_CRED_V - identifier to facilitate retrieval of a public authentication key of Party V, see Section 4.1
  * CRED_V - bstr credential containing the public authentication key of Party V, see Section 4.1

Note that only ‘protected’ and ‘signature’ of the COSE_Sign1 object are used in message_2, see next bullet.

- Compute COSE_Encrypt0 as defined in Section 5.3 of [RFC8152], with the AEAD algorithm in the selected cipher suite, K_2, IV_2, and the following parameters (further clarifications in Appendix A.2.2). The protected header SHALL be empty. The
unprotected header (not included in the EDHOC message) MAY contain parameters (e.g. ‘alg’).

* plaintext = bstr .cborseq [ ID_CRED_V, signature, ? UAD_2 ]
* external_aad = TH_2
* UAD_2 = bstr containing opaque unprotected application data

Note that ‘protected’ and ‘signature’ in the plaintext are taken from the COSE_Sign1 object, and that only ‘ciphertext’ of the COSE_Encrypt0 object are used in message_2, see next bullet. If ID_CRED_V contains a single ‘kid’ parameter, i.e., ID_CRED_V = { 4 : bstr }, only the bstr is conveyed in the plaintext, in CDDL notation

plaintext = bstr .cborseq [ bstr / header_map, bstr, ? bstr ]

Format message_2 as the sequence of CBOR data items specified in Section 4.3.1 and encode it to a byte string (see Appendix A.1). CIPHERTEXT_2 is the COSE_Encrypt0 ciphertext.

4.3.3. Party U Processing of Message 2

Party U SHALL process message_2 as follows:

o Decode message_2 (see Appendix A.1).

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

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

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

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

If any verification step fails, Party U MUST send an EDHOC error message back, formatted as defined in Section 6, and the protocol MUST be discontinued.
4.4.  EDHOC Message 3

4.4.1.  Formatting of Message 3

message_3 SHALL be a sequence of CBOR data items (see Appendix A.1) as defined below

message_3 = (data_3,
            CIPHERTEXT_3 : bstr,
            )

data_3 = (? C_V : bstr,
            )

TH_3 : bstr

where the transcript hash TH_3, in non-CDDL notation, is:

TH_3 = H( bstr .cborseq [ TH_2, CIPHERTEXT_2, data_3 ] )

4.4.2.  Party U Processing of Message 3

Party U SHALL compose message_3 as follows:

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

o  Compute COSE_Sign1 as defined in Section 4.4 of [RFC8152], using the signature algorithm in the selected cipher suite, the private authentication key of Party U, and the following parameters. The unprotected header (not included in the EDHOC message) MAY contain parameters (e.g. ‘alg’).

*  protected = bstr .cbor ID_CRED_U
*  payload = CRED_U
*  external_aad = TH_3
*  ID_CRED_U - identifier to facilitate retrieval of a public authentication key of Party U, see Section 4.1
*  CRED_U - bstr credential containing the public authentication key of Party U, see Section 4.1
Note that only ‘protected’ and ‘signature’ of the COSE_Sign1 object are used in message_3, see next bullet.

- Compute COSE_Encrypt0 as defined in Section 5.3 of [RFC8152], with the AEAD algorithm in the selected cipher suite, K_3, and IV_3 and the following parameters. The protected header SHALL be empty. The unprotected header (not included in the EDHOC message) MAY contain parameters (e.g. ‘alg’).
  * plaintext = bstr .cborseq [ ID_CRED_U, signature, ? PAD_3 ]
  * external_aad = TH_3
  * PAD_3 = bstr containing opaque protected application data

Note that ‘protected’ and ‘signature’ in the plaintext are taken from the COSE_Sign1 object, and that only ‘ciphertext’ of the COSE_Encrypt0 object are used in message_3, see next bullet. If ID_CRED_U contains a single ‘kid’ parameter, i.e., ID_CRED_U = { 4 : bstr }, only the bstr is conveyed in the plaintext.

- Format message_3 as the sequence of CBOR data items specified in Section 4.4.1 and encode it to a byte string (see Appendix A.1). CIPHERTEXT_3 is the COSE_Encrypt0 ciphertext.

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

4.4.3. Party V Processing of Message 3

Party V SHALL process message_3 as follows:

- Decode message_3 (see Appendix A.1).

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

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

- Verify COSE_Sign1 as defined in Section 4.4 of [RFC8152], using the signature algorithm in the selected cipher suite and the public authentication key of Party U.
If any verification step fails, Party V MUST send an EDHOC error message back, formatted as defined in Section 6, and the protocol MUST be discontinued.

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

5. EDHOC Authenticated with Symmetric Keys

5.1. Overview

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

- Only Party U and Party V SHALL have access to the PSK,
- Party V is able to retrieve the PSK using ID_PSK.

where the identifier ID_PSK is a COSE header maps containing COSE header parameter that can identify a pre-shared key. Pre-shared keys are typically stored as COSE_Key objects and identified with a ‘kid’ parameter (see [RFC8152]):

- ID_PSK = { 4 : bstr }

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

EDHOC with symmetric key authentication is illustrated in Figure 4.

<table>
<thead>
<tr>
<th>Party U</th>
<th>Party V</th>
</tr>
</thead>
<tbody>
<tr>
<td>TYPE, SUITES_U, X_U, C_U, ID_PSK, UAD_1</td>
<td></td>
</tr>
<tr>
<td>+---------------------------------------&gt; message_1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>C_U, X_V, C_V, AEAD(K_2; TH_2, UAD_2)</td>
</tr>
<tr>
<td></td>
<td>message_2</td>
</tr>
<tr>
<td></td>
<td>C_V, AEAD(K_3; TH_3, PAD_3)</td>
</tr>
<tr>
<td>+---------------------------------------&gt; message_3</td>
<td></td>
</tr>
</tbody>
</table>

Figure 4: Overview of EDHOC with symmetric key authentication.
EDHOC with symmetric key authentication is very similar to EDHOC with asymmetric key authentication. In the following subsections the differences compared to EDHOC with asymmetric key authentication are described.

5.2. EDHOC Message 1

5.2.1. Formatting of Message 1

message_1 SHALL be a sequence of CBOR data items (see Appendix A.1) as defined below

\[
\text{message}_1 = (\text{TYPE : int},
\text{SUITE} : \text{suite} / \{ \text{index: uint, 2* suite } \},
\text{X} : \text{bstr},
\text{C} : \text{bstr},
\text{ID} : \text{bstr} / \text{header_map},
? \text{UAD} : \text{bstr},
) \]

where:

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

- \text{ID} - identifier to facilitate retrieval of the pre-shared key. If \text{ID} contains a single ‘kid’ parameter, i.e., \text{ID} = \{ 4 : \text{bstr} \}, only the \text{bstr} used.

5.3. EDHOC Message 2

5.3.1. Processing of Message 2

- COSE_Sign1 is not used.

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

\[
\begin{align*}
* & \text{external_aad} = \text{TH}_2 \\
* & \text{plaintext} = \text{h’’} / \text{UAD}_2 \\
* & \text{UAD}_2 = \text{bstr} containing opaque unprotected application data
\end{align*}
\]
5.4. EDHOC Message 3

5.4.1. Processing of Message 3

- COSE_Sign1 is not used.
- COSE_Encrypt0 is computed as defined in Section 5.3 of [RFC8152], with the AEAD algorithm in the selected cipher suite, K_3, IV_3, and the following parameters. The protected header SHALL be empty. The unprotected header MAY contain parameters (e.g. 'alg').
  * external_aad = TH_3
  * plaintext = h'' / PAD_3
  * PAD_3 = bstr containing opaque protected application data

6. Error Handling

6.1. EDHOC Error Message

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

error SHALL be a sequence of CBOR data items (see Appendix A.1) as defined below:

```plaintext
```

where:

- TYPE = -1
- ERR_MSG - text string containing the diagnostic payload, defined in the same way as in Section 5.5.2 of [RFC7252]
6.1.1. Example Use of EDHOC Error Message with SUITES_V

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

```
Party U                                                                 Party V
<table>
<thead>
<tr>
<th>TYPE, SUITES_U {0, 5, 6, 7}, X_U, C_U, UAD_1</th>
<th>message_1</th>
</tr>
</thead>
<tbody>
<tr>
<td>TYPE, ERR_MSG, SUITES_V {6}</td>
<td>error</td>
</tr>
<tr>
<td>------------------------------------------------------------------------</td>
<td>---------------------------------------------</td>
</tr>
<tr>
<td>TYPE, SUITES_U {1, 5, 6}, X_U, C_U, UAD_1</td>
<td>message_1</td>
</tr>
</tbody>
</table>
```

Figure 5: Example use of error message with SUITES_V.

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

```
Party U                                                                 Party V
<table>
<thead>
<tr>
<th>TYPE, SUITES_U {0, 5, 6}, X_U, C_U, UAD_1</th>
<th>message_1</th>
</tr>
</thead>
<tbody>
<tr>
<td>TYPE, ERR_MSG, SUITES_V {7, 9}</td>
<td>error</td>
</tr>
<tr>
<td>------------------------------------------------------------------------</td>
<td>---------------------------------------------</td>
</tr>
<tr>
<td>TYPE, SUITES_U {2, 5, 6, 7}, X_U, C_U, UAD_1</td>
<td>message_1</td>
</tr>
</tbody>
</table>
```

Figure 6: Example use of error message with SUITES_V.

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

7. Transferring EDHOC and Deriving Application Keys

7.1. Transferring EDHOC in CoAP

It is recommended to transport EDHOC as an exchange of CoAP [RFC7252] messages. CoAP is a reliable transport that can preserve packet ordering and handle message duplication. CoAP can also perform fragmentation and protect against denial of service attacks. It is recommended to carry the EDHOC flights in Confirmable messages, especially if fragmentation is used.

By default, the CoAP client is Party U and the CoAP server is Party V, but the roles SHOULD be chosen to protect the most sensitive identity, see Section 9. By default, EDHOC is transferred in POST requests and 2.04 (Changed) responses to the Uri-Path: "/.well-known/edhoc", but an application may define its own path that can be discovered e.g. using resource directory [I-D.ietf-core-resource-directory].

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

An example of a successful EDHOC exchange using CoAP is shown in Figure 7. In this case the CoAP Token enables Party U to correlate message_1 and message_2 so the correlation parameter corr = 1.
The exchange in Figure 7 protects the client identity against active attackers and the server identity against passive attackers. An alternative exchange that protects the server identity against active attackers and the client identity against passive attackers is shown in Figure 8. In this case the CoAP Token enables Party V to correlate message_2 and message_3 so the correlation parameter corr = 2.

<table>
<thead>
<tr>
<th>Client</th>
<th>Server</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image.png" alt="Diagram" /></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 7: Transferring EDHOC in CoAP**
To protect against denial-of-service attacks, the CoAP server MAY respond to the first POST request with a 4.01 (Unauthorized) containing an Echo option [I-D.ietf-core-echo-request-tag]. This forces the initiator to demonstrate its reachability at its apparent network address. If message fragmentation is needed, the EDHOC messages may be fragmented using the CoAP Block-Wise Transfer mechanism [RFC7959].

7.1.1. Deriving an OSCORE Context from EDHOC

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

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

- The AEAD Algorithm and the HMAC-based Key Derivation Function (HKDF) are the AEAD and HKDF algorithms in the selected cipher suite.

- The Master Secret and Master Salt are derived as follows where length is the key length (in bytes) of the AEAD Algorithm.
Master Secret = EDHOC-Exporter("OSCORE Master Secret", length)
Master Salt   = EDHOC-Exporter("OSCORE Master Salt", 8)

7.2. Transferring EDHOC over Other Protocols

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

8. IANA Considerations

8.1. EDHOC Cipher Suites Registry

IANA has created a new registry titled "EDHOC Cipher Suites".

TODO

8.2. EDHOC Method Type Registry

IANA has created a new registry titled "EDHOC Method Type".

TODO

8.3. The Well-Known URI Registry

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

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

8.4. Media Types Registry

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

  o Type name: application
  o Subtype name: edhoc
  o Required parameters: N/A
  o Optional parameters: N/A
Encoding considerations: binary

Security considerations: See Section 7 of this document.

Interoperability considerations: N/A

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

Applications that use this media type: To be identified

Fragment identifier considerations: N/A

Additional information:
  * Magic number(s): N/A
  * File extension(s): N/A
  * Macintosh file type code(s): N/A

Person & email address to contact for further information: See "Authors’ Addresses" section.

Intended usage: COMMON

Restrictions on usage: N/A

Author: See "Authors’ Addresses" section.

Change Controller: IESG

8.5. CoAP Content-Formats Registry

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

Media Type: application/edhoc

Encoding:

ID: TBD42

Reference: [[this document]]
9. Security Considerations

9.1. Security Properties

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

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

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

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

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

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

9.2. Cryptographic Considerations

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

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

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

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

9.3. Mandatory to Implement Cipher Suite

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

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

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

9.5. Denial-of-Service

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

9.6. Implementation Considerations

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

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

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

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

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

9.7. Other Documents Referencing EDHOC

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

10. References

10.1. Normative References

[I-D.ietf-cbor-7049bis]

[I-D.ietf-cbor-cddl]

[I-D.ietf-core-echo-request-tag]


10.2. Informative References


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

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

A.1. CBOR and CDDL

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

CBOR data items are encoded to or decoded from byte strings using a type-length-value encoding scheme, where the three highest order bits of the initial byte contain information about the major type. CBOR supports several different types of data items, in addition to integers (int, uint), simple values (e.g. null), byte strings (bstr), and text strings (tstr), CBOR also supports arrays [] of data items and maps {} of pairs of data items. Some examples are given below. For a complete specification and more examples, see [I-D.ietf-cbor-7049bis] and [I-D.ietf-cbor-cddl]. We recommend implementors to get used to CBOR by using the CBOR playground [CborMe].
All EDHOC messages consist of a sequence of CBOR encoded data items. While an EDHOC message in itself is not a CBOR data item, it may be viewed as the CBOR encoding of an indefinite-length array \[ \text{message}_i \] without the first byte (0x9f) and the last byte (0xff), for \( i = 1, 2 \) and \( 3 \). The same applies to the EDHOC error message.

The message format specification uses the constructs ‘.cbor’ and ‘.cborseq’ enabling conversion between different CDDL types matching different CBOR items with different encodings. Some examples are given below.

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

<table>
<thead>
<tr>
<th>CDDL Type</th>
<th>Diagnostic</th>
<th>Encoded</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint</td>
<td>24</td>
<td>0x1818</td>
</tr>
<tr>
<td>bstr .cbor uint</td>
<td>&lt;&lt; 24 &gt;&gt;</td>
<td>0x421818</td>
</tr>
</tbody>
</table>

An array, say of an uint and a byte string, may be converted into a byte string (bstr):

<table>
<thead>
<tr>
<th>CDDL Type</th>
<th>Diagnostic</th>
<th>Encoded</th>
</tr>
</thead>
<tbody>
<tr>
<td>bstr</td>
<td>h’cd’</td>
<td>0x41cd</td>
</tr>
<tr>
<td>[ uint, bstr ]</td>
<td>[ 24, h’cd’ ]</td>
<td>0x82181841cd</td>
</tr>
<tr>
<td>bstr .cborseq [ uint, bstr ]</td>
<td>&lt;&lt; 24, h’cd’ &gt;&gt;</td>
<td>0x44181841cd</td>
</tr>
</tbody>
</table>
A.2. COSE

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

A.2.1. Encryption and Decryption

The COSE parameters used in COSE_Encrypt0 (see Section 5.2 of [RFC8152]) are constructed as described below. Note that "i" in "K_i", "IV_i" and "TH_i" is a variable with value i = 2 or 3, depending on whether the calculation is made over message_2 or message_3.

- The secret key K_i is a CBOR bstr, generated with the EDHOC-Key-Derivation function as defined in Section 3.3.
- The initialization vector IV_i is a CBOR bstr, also generated with the EDHOC-Key-Derivation function as defined in Section 3.3.
- The plaintext is a CBOR bstr. If the application data (UAD and PAD) is omitted, then plaintext = h’’ in the symmetric case, and plaintext = << ID_CRED_x, signature >> in the asymmetric case. Note that if ID_CRED_x contains a single ‘kid’ parameter, i.e., ID_CRED_x = { 4 : bstr }, only the bstr is conveyed in the plaintext. For instance, if ID_CRED_x = { 4 : h’40’ } (CBOR encoding 0xA1044140) and signature = h’050607’ (CBOR encoding 0x43050607), then plaintext = h’414043050607’.
- The external_aad is a CBOR bstr. It is always set to the transcript hash TH_i.

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

- The key K is the value of the key K_i.
- The nonce N is the value of the initialization vector IV_i.
- The plaintext P is the value of the COSE plaintext. E.g. if the COSE plaintext = h’010203’, then P = 0x010203.
- The associated data A is the CBOR encoding of:
  
  [ "Encrypt0", h’’, TH_i ]

This equals the concatenation of 0x8368456e63727970743040 and the CBOR encoding of TH_i. For instance if TH_2 = h’010203’ (CBOR encoding 0x43010203), then A = 0x8368456e6372797074304043010203.

A.2.2. Signing and Verification

The COSE parameters used in COSE_Sign1 (see Section 4.2 of [RFC8152]) are constructed as described below. Note that "i" in "TH_i" is a variable with values i = 2 or 3, depending on whether the calculation is made over message_2 or message_3. Note also that "x" in "ID_CRED_x" and "CRED_x" is a variable with values x = U or V, depending on whether it is the credential of U or of V that is used in the relevant protocol message.

- The key is the private authentication key of U or V. This may be stored as a COSE_KEY object or as a certificate.
- The protected parameter is a map ID_CRED_x = { label : value } is wrapped in a byte string.
- The payload is a bstr containing the CBOR encoding of a COSE_KEY or a single certificate.
- external_aad = TH_i.

COSE constructs the input to the Signature Algorithm as follows:

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

  [ "Signature1", << { label : value } >>, TH_i, CRED_x ]

  For instance, if ID_CRED_x = { 4 : h’1111’ } (CBOR encoding 0xA104421111), TH_3 = h’222222’ (CBOR encoding 0x43222222), and CRED_U = h’55555555’ (CBOR encoding 0x4455555555), then M = 0x846a5369676e61747572653145A104421111432222224455555555.

A.2.3. Key Derivation

Assuming use of the mandatory-to-implement algorithms HKDF SHA-256 and AES-CCM-16-64-128, the extract phase of HKDF produces a pseudorandom key (PRK) as follows:

PRK = HMAC-SHA-256( salt, ECDH shared secret )

where salt = 0x (the empty byte string) in the asymmetric case and salt = PSK in the symmetric case. As the output length L is smaller...

Selander, et al. Expires September 12, 2019
than the hash function output size, the expand phase of HKDF consists of a single HMAC invocation, and \( K_i \) and \( IV_i \) are therefore the first 16 and 13 bytes, respectively, of

\[
\text{output parameter} = \text{HMAC-SHA-256( PRK, info || 0x01 )}
\]

where || means byte string concatenation, and info is the CBOR encoding of

\[
\text{COSE\_KDF\_Context} = [\text{AlgorithmID}, \{\text{null}, \text{null}, \text{null}\}, \{\text{null}, \text{null}, \text{null}\}, \{\text{keyDataLength}, \text{h''}, \text{TH}_i\}]
\]

If AES-CCM-16-64-128 then AlgorithmID = 10 and keyDataLength = 128 for \( K_i \), and AlgorithmID = "IV-GENERATION" (CBOR encoding 0x6d49562d47454e45524154494f4e) and keyDataLength = 104 for \( IV_i \). Hence, if \( \text{TH}_2 = \text{h''aaa} \) then

\[
\begin{align*}
K_2 &= \text{HMAC-SHA-256( PRK, 0x840a83f6f683f6f6f68318804042aaaa01 )} \\
IV_2 &= \text{HMAC-SHA-256( PRK, 0x846d49562d47454e45524154494f4e83f6f683f6f68318804042aaaa01 )}
\end{align*}
\]

Appendix B. Example Messages and Sizes

To help implementors, this appendix gives examples in CBOR diagnostic notation and hexadecimal of EDHOC messages and plaintexts with different authentication methods. The examples use 1 byte key identifiers, 1 byte connection IDs, and the default mapping to CoAP (corr = 1). Note that the examples in this appendix are not test vectors, the cryptographic parts are just replaced with byte strings of the same length.

B.1. Message Sizes RPK

B.1.1. message_1

\[
\begin{align*}
\text{message}_1 &= \{ \\
1, \\
0, \\
h'000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d9e262f2728292a2b2c2d2e2f2g2h2i2j2k2l2m2n2o2p2q2r2s2t2u2v2w2x2y2z2A2B2C2D2E2F2G2H2I2J2K2L2M2N2O2P2Q2R2S2T2U2V2W2X2Y2Z2a2b2c2d2e2f2g2h2i2j2k2l2m2n2o2p2q2r2s2t2u2v2w2x2y2z2A2B2C2D2E2F2G2H2I2J2K2L2M2N2O2P2Q2R2S2T2U2V2W2X2Y2Z2
\}
\end{align*}
\]
message_1 (38 bytes):
01 00 58 20 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 41 C3

B.1.2. message_2

plaintext = <<
h'a1',
h'000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d
1elf202122232425262728292a2b2c2d2e2f303132333435363738393a3b
3c3d3e3f'
>>

In the plaintext, the header map { 4 : h’al’ } is encoded as the two bytes h’al’. The length of plaintext is 68 bytes so assuming a 64-bit MAC value the length of ciphertext is 76 bytes.

message_2 = {
  h’000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d
  1elf202122232425262728292a2b2c2d2e2f303132333435363738393a3b
  3c3d3e3f404142434445464748494a4b'
}

message_2 (114 bytes):
58 20 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11
12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 41 C4 58 51 00 01
02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15
16 17 18 19 1A 1B 1C 1D 1E 1F 20 21 22 23 24 25 26 27 28 29
2A 2B 2C 2D 2E 2F 30 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D
3E 3F 40 41 42 43 44 45 46 47 48 49 4A 4B

B.1.3. message_3

The plaintext and ciphertext in message_3 are assumed to be of equal sizes as in message_2.

message_3 = {
  h’c4’,
  h’000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d
  1elf202122232425262728292a2b2c2d2e2f303132333435363738393a3b
  3c3d3e3f404142434445464748494a4b'
}
message_3 (80 bytes):
41 C4 58 51 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20 21 22 23
24 25 26 27 28 29 2A 2B 2C 2D 2E 2F 30 31 32 33 34 35 36 37
38 39 3A 3B 3C 3D 3E 3F 40 41 42 43 44 45 46 47 48 49 4A 4B

B.2. Message Sizes Certificates

When the certificates are distributed out-of-band and identified with
the x5t header parameter and a SHA256/64 hash value, the header map
will be 13 bytes (assuming labels in the range -24...23).

{ TDB1 : [ TDB6, h’0001020304050607’ ] }

When the certificates are identified with the x5chain header
parameter, the message sizes depend on the size of the (truncated)
certificate chains. The header map will be 3 bytes + the size of the
certificate chain (assuming a label in the range -24...23).

{ TDB3 : h’0001020304050607...’ }

B.3. Message Sizes PSK

B.3.1. message_1

message_1 = (4,
0,
h’000102030405060708090A0b0c0d0e0f101112131415161718191a1b1c1d
1elf’,
h’c3’,
h’a2’
)

message_1 (40 bytes):
04 00 58 20 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 41 C3 41 A2

B.3.2. message_2

Assuming a 0 byte plaintext and a 64-bit MAC value the ciphertext is
8 bytes
message_2 = {
    h’000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d
    le1f’,
    h’c4’,
    h’00010203040506070’
}

message_2 (45 bytes):
58 20 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11
12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 41 C4 48 61 62 63
64 65 66 67 68

B.3.3. message_3

The plaintext and ciphertext in message_3 are assumed to be of equal
sizes as in message_2.

message_3 = {
    h’c4’,
    h’00010203040506070’
}

message_3 (11 bytes):
41 C4 48 00 01 02 03 04 05 06 07

B.4. Summary

The previous examples of typical message sizes are summarized in
Figure 9.

=====================================================================
<table>
<thead>
<tr>
<th>PSK</th>
<th>RPK</th>
<th>x5t</th>
<th>x5chain</th>
</tr>
</thead>
<tbody>
<tr>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>---------</td>
</tr>
<tr>
<td>message_1</td>
<td>40</td>
<td>38</td>
<td>38</td>
</tr>
<tr>
<td>message_2</td>
<td>45</td>
<td>114</td>
<td>126</td>
</tr>
<tr>
<td>message_3</td>
<td>11</td>
<td>80</td>
<td>91</td>
</tr>
<tr>
<td>---------</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
</tr>
<tr>
<td>Total</td>
<td>96</td>
<td>232</td>
<td>255</td>
</tr>
</tbody>
</table>
=====================================================================

Figure 9: Typical message sizes in bytes

These examples use 1 byte key identifiers and connection IDs, this is
realistic in many scenarios as most constrained devices only have a
few keys and connection. In cases where a node only have one
connection or key, the identifiers may even be the empty byte string.
For a comparison with other protocols, see [I-D.ietf-lwig-security-protocol-comparison].

Appendix C. Test Vectors

This appendix provides a wealth of test vectors to ease implementation and ensure interoperability.

TODO: This section needs to be updated.

Acknowledgments

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

Authors’ Addresses

Goeran Selander
Ericsson AB

Email: goran.selander@ericsson.com

John Mattsson
Ericsson AB

Email: john.mattsson@ericsson.com

Francesca Palombini
Ericsson AB

Email: francesca.palombini@ericsson.com
Abstract

This document compiles the requirements for a lightweight authenticated key exchange protocol for OSCORE.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on October 19, 2019.

Copyright Notice

Copyright (c) 2019 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 Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.
1. Introduction

OSCORE [I-D.ietf-core-object-security] is a lightweight communication security protocol providing end-to-end security for constrained IoT settings (cf. [RFC7228]). It is expected to be deployed with standards and platforms using CoAP such as 6TiSCH, LPWAN, OMA Specworks LwM2M, Fairhair Alliance and Open Connectivity Foundation. OSCORE lacks a matching authenticated key exchange protocol (AKE). This document compiles the requirements for such an AKE for the purpose of dispatching this work in the IETF.

2. Problem description

2.1. Credentials

IoT deployments differ in terms of what credentials can be supported. Currently many systems use pre-shared keys (PSK) provisioned out of band. PSK based provisioning has inherent weaknesses; there has been reports of massive breaches of PSK provisioning systems, and as many systems use PSK without perfect forward secrecy (PFS) they are vulnerable to passive pervasive monitoring.

The security of these systems can be improved by adding PFS through an AKE authenticated by the provisioned PSK. Reusing the provisioning scheme for raw public keys (RPK) instead of PSK, together with an AKE authenticated with the RPKs provides a more relaxed trust model since RPKs need not be secret. By running the asymmetric key AKE with public key certificates instead of RPK, key provisioning can be omitted leading to a more automated bootstrapping procedure.

These steps provide an example of a migration path in limited scoped steps from simple to more robust provisioning schemes where each step improves the overall security and/or simplicity of deployment of the
IoT system, although not all steps are necessarily feasible for the most constrained settings. With this in mind the AKE should support PSK, RPK and certificate based authentication.

2.2. Crypto Agility

Motivated by long deployment lifetimes, the AKE is required to support crypto agility, including modularity of COSE crypto algorithms and negotiation of preferred crypto algorithms for OSCORE and the AKE. The AKE negotiation should be protected against downgrade attacks.

2.3. AKE for OSCORE

In order to be suitable for OSCORE, at the end of the AKE the two parties should have agreed on:

- a shared secret (OSCORE Master Secret) with PFS and a good amount of randomness
- identifiers providing a hint to the receiver of what security context it should use when decrypting the message (OSCORE Sender IDs of peer endpoints), arbitrarily short
- COSE algorithms to use with OSCORE

Moreover, the AKE should support the same transport as OSCORE, in particular any protocol where CoAP can be transported.

To ensure that the AKE is efficient for the expected applications of OSCORE, we list the relevant public specifications of technologies where OSCORE is included:

- The IETF 6TiSCH WG charter (-02) identifies the need to "secure[e] the join process and mak[e] that fit within the constraints of high latency, low throughput and small frame sizes that characterize IEEE802.15.4 TSCH". OSCORE protects the join protocol as described in 6TiSCH Minimal Security [I-D.ietf-6tisch-minimal-security].

- The IETF LPWAN WG charter (-01) identifies the need to improve the transport capabilities of LPWA networks such as NB-IoT and LoRa whose "common traits include ... frame sizes ... [on] the order of tens of bytes transmitted a few times per day at ultra-low speeds". The application of OSCORE is described in [I-D.ietf-lpwan-coap-static-context-hc].
Other industry fora which plan to use OSCORE:

- Fairhair Alliance has defined an architecture [Fairhair] which adopts OSCORE for multicast, but it is not clear whether the architecture will support unicast OSCORE.
- Open Connectivity Foundation (OCF) has been actively involved in the OSCORE development for the purpose of deploying OSCORE, but no public reference is available since OCF only references RFCs. We believe that these OSCORE consumers reflect similar levels of constraints on the devices and networks in question.

The solution will presumably be useful in other scenarios as well since a low security overhead improves the overall performance, but we do not require the solution to necessarily be applicable anywhere else.

2.4. Lightweight

As motivated in Section 2.3 we target an AKE which is efficiently deployable in 6TiSCH multi-hop networks, LoRaWAN 1.0 networks and NB-IoT networks. The desire is to optimize the AKE to be ‘as lightweight as reasonably achievable’ in these environments, where ‘lightweight’ refers to:

- resource consumption, measured by bytes on the wire, wall-clock time to complete (i.e., the initial latency added to application protocols by the AKE), or power
- the amount of new code required on end systems which already have an OSCORE stack

It may be necessary to evaluate options that make different trade-offs across these dimensions. The properties needs to be evaluated in the context of the use of an existing CoAP/OSCORE stack in the targeted networks.

Per ‘bytes on the wire’, it is desirable for these AKE messages to fit into the MTU size of these protocols; and if not possible, within as few frames as possible, since using multiple MTUs can have significant costs in terms of time and power.

Per ‘time’, it is desirable for the AKE message exchange(s) to complete in a reasonable amount of time, both for a single
Internet-Draft

Requirements for a Lightweight AKE for OSCORE.  April 2019

uncongested exchange and when multiple exchanges are running in an interleaved fashion. This latency may not be a linear function depending on congestion and the specific radio technology used. As these are relatively low data rate networks, the latency contribution due to computation is in general not expected to be dominant.

Per ‘power’, it is desirable for the transmission of AKE messages and crypto to draw as little power as possible. The best mechanism for doing so differs across radio technologies. For example, NB-IoT uses licensed spectrum and thus can transmit at higher power to improve coverage, making the transmitted byte count relatively more important than for other radio technologies. In other cases, the radio transmitter will be active for a full MTU frame regardless of how much of the frame is occupied by message content, which makes the byte count less sensitive for the power consumption. Increased power consumption is unavoidable in poor network conditions, such as most wide-area settings including LoRaWAN.

Per ‘new code’, it is desirable to introduce as little new code as possible onto OSCORE-enabled devices to support this new AKE. These devices have on the order of 10s of kB of memory and 100 kB of storage on which an embedded OS; a COAP stack; CORE and AKE libraries; and target applications would run. It is expected that the majority of this space is available for actual application logic, as opposed to the support libraries. In a typical OSCORE implementation COSE encrypt and signature structures will be available, as will support for COSE algorithms relevant for IoT enabling the same algorithms as is used for OSCORE (e.g. COSE algorithm no. 10 = CCM* used by 6TiSCH). The use of those, or CBOR or CoAP, would not add to the footprint.

While the large variety of settings and capabilities of the devices and networks makes it challenging to produce exact values of some these dimensions, there are some key benchmarks that are tractable for security protocol engineering and which have a significant impact.

2.4.1.  LoRaWAN

LoRaWAN employs unlicensed radio frequency bands in the 868MHz ISM band, in Europe regulated by ETSI EN 300 220. For LoRaWAN the most relevant metric is the Time-on-Air, which determines the back-off times and can be used an indicator to calculate energy consumption. LoRaWAN is legally required to use a 1% (or smaller) duty cycle, a payload split into two fragments instead of one increases the time to complete the sending of this payload by at least 10,000%. The use of an AKE for providing end-to-end security on application layer need to comply with the duty cycle. One relevant benchmark is performance in
low coverage with Data Rates 0-2 corresponding to a packet size of 51 bytes [LoRaWAN]. While larger frame sizes are also defined, their use depend on good radio conditions. Some libraries/providers only support 51 bytes packet size.

2.4.2. 6TiSCH

For 6TiSCH specifically, as a time-sliced network, bytes of the wire (or rather, the quantization into frame count) is particularly noteworthy, since more frames contribute to congestion for spectrum (and concomitant error rates) in a non-linear way, especially in scenarios when large numbers of independent nodes are attempting to execute an AKE to join a network.

The available size for key exchange messages depends the topology of the network and other parameters. One benchmark which is relevant for studying AKE is the network formation setting. For a 6TiSCH production network 5 hops deep in a network formation setting, the available CoAP overhead to avoid fragmentation is 47/45 bytes (uplink/downlink) [AKE-for-6TiSCH].

2.4.3. NB-IoT

For NB-IoT, in contrast to the other two technologies below, the radio bearers are not characterized by a fixed sized PDU. Concatenation, segmentation and reassembly are part of the service provided by the radio layer. Furthermore, since NB-IoT is operating in licensed spectrum, the packets on the radio interface can be transmitted back-to-back, so the time before sending OSCORE protected data is dependent on the number of round trips/messages of the AKE. An AKE providing challenge-response based mutual authentication requires at least three messages/one round trip before it is possible to encrypt traffic data between peers meeting for the first time. NB-IoT has a high per byte energy consumption component for uplink transfers, implying that those messages should be as small as possible.

2.4.4. Discussion

While "as small protocol messages as possible" does not lend itself to a sharp boundary threshold, "as few protocol messages as possible" does and is relevant in all settings above.

The penalty is high for not fitting into the frame sizes of 6TiSCH and LoRaWAN networks. Fragmentation is not defined within these technologies so requires fragmentation scheme on a higher layer in the stack. With fragmentation increases the number of frames per message, each with its associated overhead in terms of power.
consumption and latency. Additionally the probability for errors increases, which leads to retransmissions of frames or entire messages that in turn increases the power consumption and latency.

There are trade-offs between "few messages" and "few frames"; if overhead is spread out over more messages such that each message fits into a particular frame this may reduce the overall power consumption. While it may be possible to engineer such a solution for a particular radio technology and signature algorithm, the benefits in terms of fewer messages/round trips in general and for NB-IoT in particular (see Section 2.4.3) are considered more important than optimizing for a specific scenario. Hence an optimal AKE protocol has 3 messages and each message fits into as few frames as possible, ideally 1 frame per message.

3. Requirements Summary
   o The AKE should support PSK, RPK and certificate based authentication and crypto agility, be 3-pass and support the same transport as OSCORE.
   o After the AKE run, the peers should agree on a shared secret with PFS and good amount of randomness, peer identifiers (potentially short), and COSE algorithms to use.
   o The AKE should reuse CBOR, CoAP and COSE primitives and algorithms for low code complexity of a combined OSCORE and AKE implementation.
   o The messages should be as small as reasonably achievable and fit into as few LoRaWAN packets and 6TiSCH frames as possible, optimally 1 for each message.

4. Security Considerations
   This document compiles the requirements for an AKE and provides some related security considerations to support the dispatch process. Further security considerations are out of scope of this document.

5. IANA Considerations
   None.

6. Informative References
   [AKE-for-6TiSCH]
   "AKE for 6TiSCH", n.d., <https://docs.google.com/document/d/1wLoIexMLG3U9iy5hGzKjki-VDndQBbYRNsMU1h-k>.
Internet-Draft

Requirements for a Lightweight AKE for OSCORE. April 2019

[Fairhair]

[I-D.ietf-6tisch-minimal-security]

[I-D.ietf-core-object-security]

[I-D.ietf-lpwan-coap-static-context-hc]


Author's Address

Goeran Selander
Ericsson AB

Email: goran.selander@ericsson.com