LWIG Working Group
Internet-Draft
Intended status: Informational
Expires: September 12, 2019

J. Mattsson
F. Palombini
Ericsson AB

March 11, 2019

Comparison of CoAP Security Protocols
draft-ietf-lwig-security-protocol-comparison-03

Abstract

This document analyzes and compares the sizes of key exchange flights and the per-packet message size overheads when using different security protocols to secure CoAP. The analyzed security protocols are DTLS 1.2, DTLS 1.3, TLS 1.2, TLS 1.3, EDHOC, OSCORE, and Group OSCORE. The DTLS and TLS record layers are analyzed with and without 6LoWPAN-GHC compression. DTLS is analyzed with and without Connection ID.

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

Table of Contents

1. Introduction

This document analyzes and compares the sizes of key exchange flights and the per-packet message size overheads when using different security protocols to secure CoAP over UPD [RFC7252] and TCP [RFC8323]. The analyzed security protocols are DTLS 1.2 [RFC6347], DTLS 1.3 [I-D.ietf-tls-dtls13], TLS 1.2 [RFC5246], TLS 1.3 [RFC8446], EDHOC [I-D.selander-ace-cose-ecdhe], OSCORE [I-D.ietf-core-object-security], and Group OSCORE [I-D.ietf-core-oscore-groupcomm].

The DTLS and TLS record layers are analyzed with and without 6LoWPAN-GHC compression. DTLS is anlyzed with and without Connection ID [I-D.ietf-tls-dtls-connection-id]. Readers are expected to be familiar with some of the terms described in RFC 7925 [RFC7925], such as ICV. Section 2 compares the overhead of key exchange, while Section 3 covers the overhead for protection of application data.

2. Overhead of Key Exchange Protocols

This section analyzes and compares the sizes of key exchange flights for different protocols.

To enable a fair comparison between protocols, the following assumptions are made:

- All the overhead calculations in this section use AES-CCM with a tag length of 8 bytes (e.g. AES_128_CCM_8 or AES-CCM-16-64-128).

- A minimum number of algorithms and cipher suites is offered. The algorithm used/offered are Curve25519, ECDSA with P-256, AES-CCM_8, SHA-256.
- The length of key identifiers are 1 byte.
- The length of connection identifiers are 1 byte.
- DTLS RPK makes use of point compression, which saves 32 bytes.
- DTLS handshake message fragmentation is not considered.

- Only the DTLS mandatory extensions are considered, except for Connection ID.

Section 2.1 gives a short summary of the message overhead based on different parameters and some assumptions. The following sections detail the assumptions and the calculations.

2.1. Summary

The DTLS overhead is dependent on the parameter Connection ID. The following overheads apply for all Connection IDs of the same length, when Connection ID is used.

The EDHOC overhead is dependent on the key identifiers included. The following overheads apply for Sender IDs of the same length.

All the overhead are dependent on the tag length. The following overheads apply for tags of the same length.

Figure 1 compares the message sizes of EDHOC [I-D.selander-ace-cose-ecdhe] with the DTLS 1.3 [I-D.ietf-tls-dtls13] and TLS 1.3 [RFC8446] handshakes with connection ID.

Flight	#1	#2	#3	Total
DTLS 1.3 RPK + ECDHE	150	373	213	736
DTLS 1.3 Cached X.509/RPK + ECDHE	182	347	213	742
DTLS 1.3 PSK + ECDHE	184	190	57	431

DTLS 1.3 PSK	134	150	57	341
EDHOC RPK + ECDHE	39	114	80	233
EDHOC PSK + ECDHE	41	45	11	97

Figure 1: Comparison of message sizes in bytes with Connection ID

Figure 2 compares of message sizes of DTLS 1.3 [I-D.ietf-tls-dtls13] and TLS 1.3 [RFC8446] handshakes without connection ID.

Flight	#1	#2	#3	Total
DTLS 1.3 RPK + ECDHE	144	364	212	722
DTLS 1.3 PSK + ECDHE	178	183	56	417
DTLS 1.3 PSK	128	143	56	327
TLS 1.3 RPK + ECDHE	129	322	194	645
TLS 1.3 PSK + ECDHE	163	157	50	370
TLS 1.3 PSK	113	117	50	280

Figure 2: Comparison of message sizes in bytes without Connection ID

The details of the message size calculations are given in the following sections.

2.2. DTLS 1.3

This section gives an estimate of the message sizes of DTLS 1.3 with different authentication methods. Note that the examples in this section are not test vectors, the cryptographic parts are just

replaced with byte strings of the same length, while other fixed length fields are replace with arbitrary strings or omitted, in which case their length is indicated. Values that are not arbitrary are given in hexadecimal.

2.2.1. Message Sizes RPK + ECDHE

In this section, a Connection ID of 1 byte is used.

2.2.1.1. flight_1

```
Record Header - DTLSPlaintext (13 bytes):
16 fe fd EE EE SS SS SS SS SS SS LL LL

  Handshake Header - Client Hello (10 bytes):
  01 LL LL LL SS SS 00 00 00 LL LL LL

    Legacy Version (2 bytes):
    fe fd

    Client Random (32 bytes):
    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

    Legacy Session ID (1 bytes):
    00
```



```
    Legacy Cookie (1 bytes):
    00

    Cipher Suites (TLS_AES_128_CCM_8_SHA256) (4 bytes):
    00 02 13 05

    Compression Methods (null) (2 bytes):
    01 00

    Extensions Length (2 bytes):
    LL LL

      Extension - Supported Groups (x25519) (8 bytes):
      00 0a 00 04 00 02 00 1d
```

```
      Extension - Signature Algorithms (ecdsa_secp256r1_sha256)
      (8 bytes):
      00 0d 00 04 00 02 08 07

      Extension - Key Share (42 bytes):
      00 33 00 26 00 24 00 1d 00 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

      Extension - Supported Versions (1.3) (7 bytes):
      00 2b 00 03 02 03 04

      Extension - Client Certificate Type (Raw Public Key) (6 bytes):
      00 13 00 01 01 02

      Extension - Server Certificate Type (Raw Public Key) (6 bytes):
      00 14 00 01 01 02

      Extension - Connection Identifier (43) (6 bytes):
      XX XX 00 02 01 42

 13 + 10 + 2 + 32 + 1 + 1 + 4 + 2 + 2 + 8 + 8 + 42 + 7 + 6 + 6 + 6 = 150
 bytes
```

DTLS 1.3 RPK + ECDHE flight_1 gives 150 bytes of overhead.

2.2.1.2. flight_2

```
 Record Header - DTLSPlaintext (13 bytes):
 16 fe fd EE EE SS SS SS SS SS SS LL LL

   Handshake Header - Server Hello (10 bytes):
   02 LL LL LL SS SS 00 00 00 LL LL LL
```



```
     Legacy Version (2 bytes):
     fe fd

     Server Random (32 bytes):
     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

     Legacy Session ID (1 bytes):
```

```
    00

    Cipher Suite (TLS_AES_128_CCM_8_SHA256) (2 bytes):
    13 05

    Compression Method (null) (1 bytes):
    00

    Extensions Length (2 bytes):
    LL LL

      Extension - Key Share (40 bytes):
      00 33 00 24 00 1d 00 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

      Extension - Supported Versions (1.3) (6 bytes):
      00 2b 00 02 03 04

      Extension - Connection Identifier (43) (6 bytes):
      XX XX 00 02 01 43

Record Header - DTLSCiphertext, Full (6 bytes):
HH ES SS 43 LL LL

  Handshake Header - Encrypted Extensions (10 bytes):
  08 LL LL LL SS SS 00 00 00 LL LL LL

    Extensions Length (2 bytes):
    LL LL

      Extension - Client Certificate Type (Raw Public Key) (6 bytes):
      00 13 00 01 01 02

      Extension - Server Certificate Type (Raw Public Key) (6 bytes):
      00 14 00 01 01 02

  Handshake Header - Certificate Request (10 bytes):
  0d LL LL LL SS SS 00 00 00 LL LL LL
```



```
    Request Context (1 bytes):
```

```
    00

    Extensions Length (2 bytes):
    LL LL

      Extension - Signature Algorithms (ecdsa_secp256r1_sha256)
      (8 bytes):
      00 0d 00 04 00 02 08 07

  Handshake Header - Certificate (10 bytes):
  0b LL LL LL SS SS 00 00 00 LL LL LL

    Request Context (1 bytes):
    00

    Certificate List Length (3 bytes):
    LL LL LL

    Certificate Length (3 bytes):
    LL LL LL

    Certificate (59 bytes) // Point compression
    ....

    Certificate Extensions (2 bytes):
    00 00

  Handshake Header - Certificate Verify (10 bytes):
  0f LL LL LL SS SS 00 00 00 LL LL LL

    Signature  (68 bytes):
    ZZ ZZ 00 40 ....

  Handshake Header - Finished (10 bytes):
  14 LL LL LL SS SS 00 00 00 LL LL LL

    Verify Data (32 bytes):
    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

  Record Type (1 byte):
  16

Auth Tag (8 bytes):
e0 8b 0e 45 5a 35 0a e5

13 + 102 + 6 + 24 + 21 + 78 + 78 + 42 + 1 + 8 = 373 bytes
```

DTLS 1.3 RPK + ECDHE flight_2 gives 373 bytes of overhead.

2.2.1.3. flight_3

```
Record Header (6 bytes) // DTLSCiphertext, Full:
ZZ ES SS 42 LL LL

  Handshake Header - Certificate (10 bytes):
  0b LL LL LL SS SS XX XX XX LL LL LL

    Request Context (1 bytes):
    00

    Certificate List Length (3 bytes):
    LL LL LL

    Certificate Length (3 bytes):
    LL LL LL

    Certificate (59 bytes) // Point compression
    ....

    Certificate Extensions (2 bytes):
    00 00

  Handshake Header - Certificate Verify (10 bytes):
  0f LL LL LL SS SS 00 00 00 LL LL LL

    Signature  (68 bytes):
    04 03 LL LL //ecdsa_secp256r1_sha256
    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 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

  Handshake Header - Finished (10 bytes):
  14 LL LL LL SS SS 00 00 00 LL LL LL

    Verify Data (32 bytes) // SHA-256:
    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

  Record Type (1 byte):
  16

Auth Tag (8 bytes) // AES-CCM_8:
00 01 02 03 04 05 06 07
```

6 + 78 + 78 + 42 + 1 + 8 = 213 bytes

DTLS 1.3 RPK + ECDHE flight_2 gives 213 bytes of overhead.

2.2.2. Message Sizes PSK + ECDHE

2.2.2.1. flight_1

The differences in overhead compared to Section 2.2.1.1 are:

The following is added:

```
+ Extension - PSK Key Exchange Modes (6 bytes):
  00 2d 00 02 01 01

+ Extension - Pre Shared Key (48 bytes):
  00 29 00 2F
  00 0a 00 01 ID 00 00 00 00
  00 21 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
```

The following is removed:

- Extension - Signature Algorithms (ecdsa_secp256r1_sha256) (8 bytes)

- Extension - Client Certificate Type (Raw Public Key) (6 bytes)

- Extension - Server Certificate Type (Raw Public Key) (6 bytes)

In total:

150 + 6 + 48 - 8 - 6 - 6 = 184 bytes

DTLS 1.3 PSK + ECDHE flight_1 gives 184 bytes of overhead.

2.2.2.2. flight_2

The differences in overhead compared to Section 2.2.1.2 are:

The following is added:

+ Extension - Pre Shared Key (6 bytes)
  00 29 00 02 00 00

The following is removed:

- Handshake Message Certificate (78 bytes)

- Handshake Message CertificateVerify (78 bytes)

- Handshake Message CertificateRequest (21 bytes)

- Extension - Client Certificate Type (Raw Public Key) (6 bytes)

- Extension - Server Certificate Type (Raw Public Key) (6 bytes)

In total:

373 - 78 - 78 - 21 - 6 - 6 + 6 = 190 bytes

DTLS 1.3 PSK + ECDHE flight_2 gives 190 bytes of overhead.

2.2.2.3. flight_3

The differences in overhead compared to Section 2.2.1.3 are:

The following is removed:

- Handshake Message Certificate (78 bytes)

- Handshake Message Certificate Verify (78 bytes)

In total:

213 - 78 - 78 = 57 bytes

DTLS 1.3 PSK + ECDHE flight_3 gives 57 bytes of overhead.

2.2.3. Message Sizes PSK

2.2.3.1. flight_1

The differences in overhead compared to Section 2.2.2.1 are:

The following is removed:

- Extension - Supported Groups (x25519) (8 bytes)

- Extension - Key Share (42 bytes)

In total:

184 - 8 - 42 = 134 bytes

DTLS 1.3 PSK flight_1 gives 134 bytes of overhead.

2.2.3.2. flight_2

The differences in overhead compared to Section 2.2.2.2 are:

The following is removed:

- Extension - Key Share (40 bytes)

In total:

190 - 40 = 150 bytes

DTLS 1.3 PSK flight_2 gives 150 bytes of overhead.

2.2.3.3. flight_3

There are no differences in overhead compared to Section 2.2.2.3.

DTLS 1.3 PSK flight_3 gives 57 bytes of overhead.

2.2.4. Cached Information

In this section, we consider the effect of [RFC7924] on the message

size overhead.

Cached information together with server X.509 can be used to move bytes from flight #2 to flight #1 (cached RPK increases the number of bytes compared to cached X.509).

The differences compared to Section 2.2.1 are the following.

For the flight #1, the following is added:

```
+ Extension - Client Cashed Information (39 bytes):
  00 19 LL LL LL LL
  01 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
```

And the following is removed:

```
- Extension - Server Certificate Type (Raw Public Key) (6 bytes)
```

Giving a total of:

150 + 33 = 183 bytes

For the flight #2, the following is added:

```
+ Extension - Server Cashed Information (7 bytes):
  00 19 LL LL LL LL 01
```

And the following is removed:

```
- Extension - Server Certificate Type (Raw Public Key) (6 bytes)

- Server Certificate (59 bytes -> 32 bytes)
```

Giving a total of:

373 - 26 = 347 bytes

A summary of the calculation is given in Figure 3.

==========

Flight	#1	#2	#3	Total
DTLS 1.3 Cached X.509/RPK + ECDHE	183	347	213	743
DTLS 1.3 RPK + ECDHE	150	373	213	736

Figure 3: Comparison of message sizes in bytes for DTLS 1.3 RPK + ECDH with and without cached X.509

2.2.5. Resumption

To enable resumption, a 4th flight (New Session Ticket) is added to the PSK handshake.

Record Header - DTLSCiphertext, Full (6 bytes):
HH ES SS 43 LL LL

Handshake Header - New Session Ticket (10 bytes):
04 LL LL LL SS SS 00 00 00 LL LL LL

Ticket Lifetime (4 bytes):
00 01 02 03

Ticket Age Add (4 bytes):
00 01 02 03

```
      Ticket Nonce (2 bytes):
      01 00

      Ticket (6 bytes):
      00 04 ID ID ID ID

      Extensions (2 bytes):
      00 00

   Auth Tag (8 bytes) // AES-CCM_8:
   00 01 02 03 04 05 06 07

   6 + 10 + 4 + 4 + 2 + 6 + 2 + 8 = 42 bytes
```

The initial handshake when resumption is enabled is just a PSK handshake with 134 + 150 + 57 + 42 = 383 bytes.

2.2.6. Without Connection ID

Without a Connection ID the DTLS 1.3 flight sizes changes as follows.

DTLS 1.3 Flight #1: -6 bytes
DTLS 1.3 Flight #2: -7 bytes
DTLS 1.3 Flight #3: -1 byte

Flight	#1	#2	#3	Total
DTLS 1.3 RPK + ECDHE (no cid)	144	364	212	722
DTLS 1.3 PSK + ECDHE (no cid)	178	183	56	417
DTLS 1.3 PSK (no cid)	128	143	56	327

Figure 4: Comparison of message sizes in bytes for DTLS 1.3 without Connection ID

2.2.7. DTLS Raw Public Keys

TODO

2.2.7.1. SubjectPublicKeyInfo without point compression

```
0x30 // Sequence
0x59 // Size 89

0x30 // Sequence
0x13 // Size 19
0x06 0x07 0x2A 0x86 0x48 0xCE 0x3D 0x02 0x01
     // OID 1.2.840.10045.2.1 (ecPublicKey)
0x06 0x08 0x2A 0x86 0x48 0xCE 0x3D 0x03 0x01 0x07
     // OID 1.2.840.10045.3.1.7 (secp256r1)

0x03 // Bit string
0x42 // Size 66
0x00 // Unused bits 0
0x04 // Uncompressed
...... 64 bytes X and Y

Total of 91 bytes
```

2.2.7.2. SubjectPublicKeyInfo with point compression

```
0x30 // Sequence
0x59 // Size 89

0x30 // Sequence
0x13 // Size 19
0x06 0x07 0x2A 0x86 0x48 0xCE 0x3D 0x02 0x01
     // OID 1.2.840.10045.2.1 (ecPublicKey)
0x06 0x08 0x2A 0x86 0x48 0xCE 0x3D 0x03 0x01 0x07
     // OID 1.2.840.10045.3.1.7 (secp256r1)

0x03 // Bit string
0x42 // Size 66
0x00 // Unused bits 0
0x03 // Compressed
...... 32 bytes X

Total of 59 bytes
```

2.3. TLS 1.3

In this section, the message sizes are calculated for TLS 1.3. The major changes compared to DTLS 1.3 are that the record header is smaller, the handshake headers is smaller, and that Connection ID is not supported. Recently, additional work has taken shape with the goal to further reduce overhead for TLS 1.3 (see [I-D.schaad-ace-tls-cbor-handshake]).

TLS Assumptions:

- Minimum number of algorithms and cipher suites offered
- Curve25519, ECDSA with P-256, AES-CCM_8, SHA-256
- Length of key identifiers: 1 bytes
- TLS RPK with point compression (saves 32 bytes)
- Only mandatory TLS extensions

For the PSK calculations, [Ulfheim-TLS13] was a useful resource, while for RPK calculations we followed the work of [IoT-Cert].

2.3.1. Message Sizes RPK + ECDHE

2.3.1.1. flight_1

```
Record Header - TLSPlaintext (5 bytes):
16 03 03 LL LL

  Handshake Header - Client Hello (4 bytes):
  01 LL LL LL

    Legacy Version (2 bytes):
    03 03

    Client Random (32 bytes):
    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

    Legacy Session ID (1 bytes):
    00

    Cipher Suites (TLS_AES_128_CCM_8_SHA256) (4 bytes):
    00 02 13 05

    Compression Methods (null) (2 bytes):
    01 00

    Extensions Length (2 bytes):
    LL LL

      Extension - Supported Groups (x25519) (8 bytes):
      00 0a 00 04 00 02 00 1d

      Extension - Signature Algorithms (ecdsa_secp256r1_sha256) (8 bytes):
      00 0d 00 04 00 02 08 07

      Extension - Key Share (42 bytes):
      00 33 00 26 00 24 00 1d 00 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

      Extension - Supported Versions (1.3) (7 bytes):
      00 2b 00 03 02 03 04

      Extension - Client Certificate Type (Raw Public Key) (6 bytes):
```

```
      00 13 00 01 01 02

      Extension - Server Certificate Type (Raw Public Key) (6 bytes):
      00 14 00 01 01 02
```

5 + 4 + 2 + 32 + 1 + 4 + 2 + 2 + 8 + 8 + 42 + 7 + 6 + 6 = 129 bytes

TLS 1.3 RPK + ECDHE flight_1 gives 129 bytes of overhead.

2.3.1.2. flight_2

```
Record Header - TLSPlaintext (5 bytes):
16 03 03 LL LL

  Handshake Header - Server Hello (4 bytes):
  02 LL LL LL

    Legacy Version (2 bytes):
    fe fd

    Server Random (32 bytes):
    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

    Legacy Session ID (1 bytes):
    00

    Cipher Suite (TLS_AES_128_CCM_8_SHA256) (2 bytes):
    13 05

    Compression Method (null) (1 bytes):
    00

    Extensions Length (2 bytes):
    LL LL

      Extension - Key Share (40 bytes):
      00 33 00 24 00 1d 00 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

      Extension - Supported Versions (1.3) (6 bytes):
```

```
      00 2b 00 02 03 04

Record Header - TLSCiphertext (5 bytes):
17 03 03 LL LL

  Handshake Header - Encrypted Extensions (4 bytes):
  08 LL LL LL

    Extensions Length (2 bytes):
    LL LL

      Extension - Client Certificate Type (Raw Public Key) (6 bytes):
      00 13 00 01 01 02

      Extension - Server Certificate Type (Raw Public Key) (6 bytes):
```



```
      00 14 00 01 01 02

  Handshake Header - Certificate Request (4 bytes):
  0d LL LL LL

    Request Context (1 bytes):
    00

    Extensions Length (2 bytes):
    LL LL

      Extension - Signature Algorithms (ecdsa_secp256r1_sha256) (8 bytes):
      00 0d 00 04 00 02 08 07

  Handshake Header - Certificate (4 bytes):
  0b LL LL LL

    Request Context (1 bytes):
    00

    Certificate List Length (3 bytes):
    LL LL LL

    Certificate Length (3 bytes):
    LL LL LL
```

```
    Certificate (59 bytes) // Point compression
    ....

    Certificate Extensions (2 bytes):
    00 00

  Handshake Header - Certificate Verify (4 bytes):
  0f LL LL LL

    Signature  (68 bytes):
    ZZ ZZ 00 40 ....

  Handshake Header - Finished (4 bytes):
  14 LL LL LL

    Verify Data (32 bytes):
    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

  Record Type (1 byte):
  16
```



```
Auth Tag (8 bytes):
e0 8b 0e 45 5a 35 0a e5
```

5 + 90 + 5 + 18 + 15 + 72 + 72 + 36 + 1 + 8 = 322 bytes

TLS 1.3 RPK + ECDHE flight_2 gives 322 bytes of overhead.

2.3.1.3. flight_3

```
   Record Header - TLSCiphertext (5 bytes):
   17 03 03 LL LL

     Handshake Header - Certificate (4 bytes):
     0b LL LL LL

       Request Context (1 bytes):
       00

       Certificate List Length (3 bytes):
       LL LL LL
```

```
      Certificate Length (3 bytes):
      LL LL LL

      Certificate (59 bytes) // Point compression
      ....

      Certificate Extensions (2 bytes):
      00 00

    Handshake Header - Certificate Verify (4 bytes):
    0f LL LL LL

      Signature  (68 bytes):
      04 03 LL LL //ecdsa_secp256r1_sha256
      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 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

    Handshake Header - Finished (4 bytes):
    14 LL LL LL

      Verify Data (32 bytes) // SHA-256:
      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

    Record Type (1 byte)
    16

  Auth Tag (8 bytes) // AES-CCM_8:
  00 01 02 03 04 05 06 07

  5 + 72 + 72 + 36 + 1 + 8 = 194 bytes

  TLS 1.3 RPK + ECDHE flight_3 gives 194 bytes of overhead.
```


2.3.2. Message Sizes PSK + ECDHE

2.3.2.1. flight_1

The differences in overhead compared to Section 2.3.1.3 are:

The following is added:

```
+ Extension - PSK Key Exchange Modes (6 bytes):
  00 2d 00 02 01 01

+ Extension - Pre Shared Key (48 bytes):
  00 29 00 2F
  00 0a 00 01 ID 00 00 00 00
  00 21 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
```

The following is removed:

- Extension - Signature Algorithms (ecdsa_secp256r1_sha256) (8 bytes)

- Extension - Client Certificate Type (Raw Public Key) (6 bytes)

- Extension - Server Certificate Type (Raw Public Key) (6 bytes)

In total:

129 + 6 + 48 - 8 - 6 - 6 = 163 bytes

TLS 1.3 PSK + ECDHE flight_1 gives 166 bytes of overhead.

2.3.2.2. flight_2

The differences in overhead compared to Section 2.3.1.2 are:

The following is added:

```
+ Extension - Pre Shared Key (6 bytes)
  00 29 00 02 00 00
```

The following is removed:

- Handshake Message Certificate (72 bytes)

- Handshake Message CertificateVerify (72 bytes)

- Handshake Message CertificateRequest (15 bytes)

- Extension - Client Certificate Type (Raw Public Key) (6 bytes)

- Extension - Server Certificate Type (Raw Public Key) (6 bytes)

In total:

322 - 72 - 72 - 15 - 6 - 6 + 6 = 157 bytes

TLS 1.3 PSK + ECDHE flight_2 gives 157 bytes of overhead.

2.3.2.3. flight_3

The differences in overhead compared to Section 2.3.1.3 are:

The following is removed:

- Handshake Message Certificate (72 bytes)

- Handshake Message Certificate Verify (72 bytes)

In total:

194 - 72 - 72 = 50 bytes

TLS 1.3 PSK + ECDHE flight_3 gives 50 bytes of overhead.

2.3.3. Message Sizes PSK

2.3.3.1. flight_1

The differences in overhead compared to Section 2.3.2.1 are:

The following is removed:

- Extension - Supported Groups (x25519) (8 bytes)

- Extension - Key Share (42 bytes)

In total:

163 - 8 - 42 = 113 bytes

TLS 1.3 PSK flight_1 gives 116 bytes of overhead.

2.3.3.2. flight_2

The differences in overhead compared to Section 2.3.2.2 are:

The following is removed:

- Extension - Key Share (40 bytes)

In total:

157 - 40 = 117 bytes

TLS 1.3 PSK flight_2 gives 117 bytes of overhead.

2.3.3.3. flight_3

There are no differences in overhead compared to Section 2.3.2.3.

TLS 1.3 PSK flight_3 gives 57 bytes of overhead.

2.4. EDHOC

This section gives an estimate of the message sizes of EDHOC with different authentication methods. Note that the examples in this section are not test vectors, the cryptographic parts are just replaced with byte strings of the same length. All examples are given in CBOR diagnostic notation and hexadecimal.

2.4.1. Message Sizes RPK

2.4.1.1. message_1

```
message_1 = (
  1,
  0,
  h'000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d
    1e1f',
  h'c3'
)
```

```
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
```


2.4.1.2. message_2

```
plaintext = <<
  h'a1',
  h'000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d
    1e1f202122232425262728292a2b2c2d2e2f303132333435363738393a3b
    3c3d3e3f'
>>
```

The header map { 4 : h'a1' } is encoded as the two bytes h'a1'. 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
    1e1f',
  h'c4',
  h'000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d
    1e1f202122232425262728292a2b2c2d2e2f303132333435363738393a3b
    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
```

2.4.1.3. 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
    1e1f202122232425262728292a2b2c2d2e2f303132333435363738393a3b
    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
```


2.4.2. 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 depends 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...' }
```

2.4.3. Message Sizes PSK

2.4.4. message_1

```
message_1 = (
  4,
  0,
  h'000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d
    1e1f',
  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
```

2.4.5. message_2

Assuming a 0 byte plaintext and a 64-bit MAC value the ciphertext is 8 bytes

```
message_2 = (
  h'000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d
    1e1f',
  h'c4',
  h'0001020304050607'
)

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
```


2.4.6. 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'0001020304050607'
)

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

2.4.7. Summary

The previous examples of typical message sizes are summarized in Figure 5.

```
=============================================================================
                PSK       RPK       x5t     x5chain
-----------------------------------------------------------------------------
```

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

Figure 5: Typical message sizes in bytes

2.5. Conclusion

To do a fair comparison, one has to choose a specific deployment and look at the topology, the whole protocol stack, frame sizes (e.g. 51 or 128 bytes), how and where in the protocol stack fragmentation is done, and the expected packet loss. Note that the number of byte in each frame that is available for the key exchange protocol may depend on the underlying protocol layers as well as the number of hops in multi-hop networks. The packet loss depends may depend on how many other devices that are transmitting at the same time, and may increase during network formation. The total overhead will be larger due to mechanisms for fragmentation, retransmission, and packet ordering. The overhead of fragmentation is roughly proportional to the number of fragments, while the expected overhead due to retransmission in noisy environments is a superlinear function of the flight sizes.

3. Overhead for Protection of Application Data

To enable comparison, all the overhead calculations in this section use AES-CCM with a tag length of 8 bytes (e.g. AES_128_CCM_8 or AES-CCM-16-64), a plaintext of 6 bytes, and the sequence number '05'. This follows the example in [RFC7400], Figure 16.

Note that the compressed overhead calculations for DLTS 1.2, DTLS 1.3, TLS 1.2 and TLS 1.3 are dependent on the parameters epoch, sequence number, and length, and all the overhead calculations are dependent on the parameter Connection ID when used. Note that the OSCORE overhead calculations are dependent on the CoAP option numbers, as well as the length of the OSCORE parameters Sender ID and Sequence Number. The following calculations are only examples.

Section 3.1 gives a short summary of the message overhead based on different parameters and some assumptions. The following sections detail the assumptions and the calculations.

3.1. Summary

The DTLS overhead is dependent on the parameter Connection ID. The following overheads apply for all Connection IDs with the same length.

The compression overhead (GHC) is dependent on the parameters epoch, sequence number, Connection ID, and length (where applicable). The following overheads should be representative for sequence numbers and Connection IDs with the same length.

The OSCORE overhead is dependent on the included CoAP Option numbers as well as the length of the OSCORE parameters Sender ID and sequence number. The following overheads apply for all sequence numbers and Sender IDs with the same length.

Sequence Number	'05'	'1005'	'100005'
DTLS 1.2	29	29	29
DTLS 1.3	11	12	12
DTLS 1.2 (GHC)	16	16	16
DTLS 1.3 (GHC)	12	13	13

TLS 1.2	21	21	21
TLS 1.3	14	14	14
TLS 1.2 (GHC)	17	18	19
TLS 1.3 (GHC)	15	16	17
OSCORE request	13	14	15
OSCORE response	11	11	11

Figure 6: Overhead in bytes as a function of sequence number (Connection/Sender ID = '')

Connection/Sender ID	''	'42'	'4002'
DTLS 1.2	29	30	31
DTLS 1.3	11	12	13
DTLS 1.2 (GHC)	16	17	18
DTLS 1.3 (GHC)	12	13	14
OSCORE request	13	14	15
OSCORE response	11	11	11

Figure 7: Overhead in bytes as a function of Connection/Sender ID (Sequence Number = '05')

Protocol	Overhead	Overhead (GHC)
DTLS 1.2	21	8
DTLS 1.3	3	4
TLS 1.2	13	9
TLS 1.3	6	7
OSCORE request	5	
OSCORE response	3	

Figure 8: Overhead (excluding ICV) in bytes (Connection/Sender ID = '', Sequence Number = '05')

3.2. DTLS 1.2

3.2.1. DTLS 1.2

This section analyzes the overhead of DTLS 1.2 [RFC6347]. The nonce follow the strict profiling given in [RFC7925]. This example is taken directly from [RFC7400], Figure 16.

```
DTLS 1.2 record layer (35 bytes, 29 bytes overhead):
17 fe fd 00 01 00 00 00 00 00 05 00 16 00 01 00
00 00 00 00 05 ae a0 15 56 67 92 4d ff 8a 24 e4
cb 35 b9

Content type:
17
Version:
fe fd
Epoch:
00 01
Sequence number:
00 00 00 00 00 05
Length:
00 16
Nonce:
00 01 00 00 00 00 00 05
Ciphertext:
ae a0 15 56 67 92
ICV:
4d ff 8a 24 e4 cb 35 b9
```

DTLS 1.2 gives 29 bytes overhead.

3.2.2. DTLS 1.2 with 6LoWPAN-GHC

This section analyzes the overhead of DTLS 1.2 [RFC6347] when compressed with 6LoWPAN-GHC [RFC7400]. The compression was done with [OlegHahm-ghc].

Note that the sequence number '01' used in [RFC7400], Figure 15 gives an exceptionally small overhead that is not representative.

Note that this header compression is not available when DTLS is used over transports that do not use 6LoWPAN together with 6LoWPAN-GHC.

Compressed DTLS 1.2 record layer (22 bytes, 16 bytes overhead):
b0 c3 03 05 00 16 f2 0e ae a0 15 56 67 92 4d ff
8a 24 e4 cb 35 b9

Compressed DTLS 1.2 record layer header and nonce:
b0 c3 03 05 00 16 f2 0e
Ciphertext:
ae a0 15 56 67 92
ICV:
4d ff 8a 24 e4 cb 35 b9

When compressed with 6LoWPAN-GHC, DTLS 1.2 with the above parameters (epoch, sequence number, length) gives 16 bytes overhead.

3.2.3. DTLS 1.2 with Connection ID

This section analyzes the overhead of DTLS 1.2 [RFC6347] with Connection ID [I-D.ietf-tls-dtls-connection-id]. The overhead calculations in this section uses Connection ID = '42'. DTLS recored layer with a Connection ID = '' (the empty string) is equal to DTLS without Connection ID.

DTLS 1.2 record layer (36 bytes, 30 bytes overhead):
17 fe fd 00 01 00 00 00 00 00 05 42 00 16 00 01
00 00 00 00 00 05 ae a0 15 56 67 92 4d ff 8a 24
e4 cb 35 b9

Content type:
17
Version:
fe fd
Epoch:
00 01
Sequence number:
00 00 00 00 00 05
Connection ID:
42
Length:
00 16
Nonce:
00 01 00 00 00 00 00 05
Ciphertext:
ae a0 15 56 67 92

ICV:
4d ff 8a 24 e4 cb 35 b9

DTLS 1.2 with Connection ID gives 30 bytes overhead.

3.2.4. DTLS 1.2 with Connection ID and 6LoWPAN-GHC

This section analyzes the overhead of DTLS 1.2 [RFC6347] with Connection ID [I-D.ietf-tls-dtls-connection-id] when compressed with 6LoWPAN-GHC [RFC7400] [OlegHahm-ghc].

Note that the sequence number '01' used in [RFC7400], Figure 15 gives an exceptionally small overhead that is not representative.

Note that this header compression is not available when DTLS is used over transports that do not use 6LoWPAN together with 6LoWPAN-GHC.

Compressed DTLS 1.2 record layer (23 bytes, 17 bytes overhead):
b0 c3 04 05 42 00 16 f2 0e ae a0 15 56 67 92 4d
ff 8a 24 e4 cb 35 b9

Compressed DTLS 1.2 record layer header and nonce:
b0 c3 04 05 42 00 16 f2 0e
Ciphertext:
ae a0 15 56 67 92
ICV:
4d ff 8a 24 e4 cb 35 b9

When compressed with 6LoWPAN-GHC, DTLS 1.2 with the above parameters (epoch, sequence number, Connection ID, length) gives 17 bytes overhead.

3.3. DTLS 1.3

3.3.1. DTLS 1.3

This section analyzes the overhead of DTLS 1.3 [I-D.ietf-tls-dtls13]. The changes compared to DTLS 1.2 are: omission of version number, merging of epoch into the first byte containing signalling bits, optional omission of length, reduction of sequence number into a 1 or 2-bytes field.

Only the minimal header format for DTLS 1.3 is analyzed (see Figure 4 of [I-D.ietf-tls-dtls13]). The minimal header formal omit the length field and only a 1-byte field is used to carry the 8 low order bits of the sequence number

DTLS 1.3 record layer (17 bytes, 11 bytes overhead):
21 05 ae a0 15 56 67 92 ec 4d ff 8a 24 e4 cb 35 b9

First byte (including epoch):
21
Sequence number:
05
Ciphertext (including encrypted content type):
ae a0 15 56 67 92 ec
ICV:
4d ff 8a 24 e4 cb 35 b9

DTLS 1.3 gives 11 bytes overhead.

3.3.2. DTLS 1.3 with 6LoWPAN-GHC

This section analyzes the overhead of DTLS 1.3 [I-D.ietf-tls-dtls13] when compressed with 6LoWPAN-GHC [RFC7400] [OlegHahm-ghc].

Note that this header compression is not available when DTLS is used over transports that do not use 6LoWPAN together with 6LoWPAN-GHC.

Compressed DTLS 1.3 record layer (18 bytes, 12 bytes overhead):
11 21 05 ae a0 15 56 67 92 ec 4d ff 8a 24 e4 cb
35 b9

Compressed DTLS 1.3 record layer header and nonce:
11 21 05
Ciphertext (including encrypted content type):

```
ae a0 15 56 67 92 ec
ICV:
4d ff 8a 24 e4 cb 35 b9
```

When compressed with 6LoWPAN-GHC, DTLS 1.3 with the above parameters (epoch, sequence number, no length) gives 12 bytes overhead.

3.3.3. DTLS 1.3 with Connection ID

This section analyzes the overhead of DTLS 1.3 [I-D.ietf-tls-dtls13] with Connection ID [I-D.ietf-tls-dtls-connection-id].

In this example, the length field is omitted, and the 1-byte field is used for the sequence number. The minimal DTLSCiphertext structure is used (see Figure 4 of [I-D.ietf-tls-dtls13]), with the addition of the Connection ID field.


```
DTLS 1.3 record layer (18 bytes, 12 bytes overhead):
31 42 05 ae a0 15 56 67 92 ec 4d ff 8a 24 e4 cb 35 b9

First byte (including epoch):
31
Connection ID:
42
Sequence number:
05
Ciphertext (including encrypted content type):
ae a0 15 56 67 92 ec
ICV:
4d ff 8a 24 e4 cb 35 b9
```

DTLS 1.3 with Connection ID gives 12 bytes overhead.

3.3.4. DTLS 1.3 with Connection ID and 6LoWPAN-GHC

This section analyzes the overhead of DTLS 1.3 [I-D.ietf-tls-dtls13] with Connection ID [I-D.ietf-tls-dtls-connection-id] when compressed with 6LoWPAN-GHC [RFC7400] [OlegHahm-ghc].

Note that this header compression is not available when DTLS is used over transports that do not use 6LoWPAN together with 6LoWPAN-GHC.

Compressed DTLS 1.3 record layer (19 bytes, 13 bytes overhead):
12 31 05 42 ae a0 15 56 67 92 ec 4d ff 8a 24 e4
cb 35 b9

Compressed DTLS 1.3 record layer header and nonce:
12 31 05 42
Ciphertext (including encrypted content type):
ae a0 15 56 67 92 ec
ICV:
4d ff 8a 24 e4 cb 35 b9

When compressed with 6LoWPAN-GHC, DTLS 1.3 with the above parameters (epoch, sequence number, Connection ID, no length) gives 13 bytes overhead.

3.4. TLS 1.2

3.4.1. TLS 1.2

This section analyzes the overhead of TLS 1.2 [RFC5246]. The changes compared to DTLS 1.2 is that the TLS 1.2 record layer does not have epoch and sequence number, and that the version is different.

TLS 1.2 Record Layer (27 bytes, 21 bytes overhead):
17 03 03 00 16 00 00 00 00 00 00 00 05 ae a0 15
56 67 92 4d ff 8a 24 e4 cb 35 b9

Content type:
17
Version:
03 03
Length:
00 16
Nonce:
00 00 00 00 00 00 00 05
Ciphertext:
ae a0 15 56 67 92
ICV:

4d ff 8a 24 e4 cb 35 b9

TLS 1.2 gives 21 bytes overhead.

3.4.2. TLS 1.2 with 6LoWPAN-GHC

This section analyzes the overhead of TLS 1.2 [RFC5246] when compressed with 6LoWPAN-GHC [RFC7400] [OlegHahm-ghc].

Note that this header compression is not available when TLS is used over transports that do not use 6LoWPAN together with 6LoWPAN-GHC.

Compressed TLS 1.2 record layer (23 bytes, 17 bytes overhead):
05 17 03 03 00 16 85 0f 05 ae a0 15 56 67 92 4d
ff 8a 24 e4 cb 35 b9

Compressed TLS 1.2 record layer header and nonce:
05 17 03 03 00 16 85 0f 05
Ciphertext:
ae a0 15 56 67 92
ICV:
4d ff 8a 24 e4 cb 35 b9

When compressed with 6LoWPAN-GHC, TLS 1.2 with the above parameters (epoch, sequence number, length) gives 17 bytes overhead.

3.5. TLS 1.3

3.5.1. TLS 1.3

This section analyzes the overhead of TLS 1.3 [RFC8446]. The change compared to TLS 1.2 is that the TLS 1.3 record layer uses a different version.

TLS 1.3 Record Layer (20 bytes, 14 bytes overhead):
17 03 03 00 16 ae a0 15 56 67 92 ec 4d ff 8a 24
e4 cb 35 b9

Content type:
17
Legacy version:
03 03

Length:
00 0f
Ciphertext (including encrypted content type):
ae a0 15 56 67 92 ec
ICV:
4d ff 8a 24 e4 cb 35 b9

TLS 1.3 gives 14 bytes overhead.

3.5.2. TLS 1.3 with 6LoWPAN-GHC

This section analyzes the overhead of TLS 1.3 [RFC8446] when compressed with 6LoWPAN-GHC [RFC7400] [OlegHahm-ghc].

Note that this header compression is not available when TLS is used over transports that do not use 6LoWPAN together with 6LoWPAN-GHC.

Compressed TLS 1.3 record layer (21 bytes, 15 bytes overhead):
14 17 03 03 00 0f ae a0 15 56 67 92 ec 4d ff 8a
24 e4 cb 35 b9

Compressed TLS 1.3 record layer header and nonce:
14 17 03 03 00 0f
Ciphertext (including encrypted content type):
ae a0 15 56 67 92 ec
ICV:
4d ff 8a 24 e4 cb 35 b9

When compressed with 6LoWPAN-GHC, TLS 1.3 with the above parameters (epoch, sequence number, length) gives 15 bytes overhead.

3.6. OSCORE

This section analyzes the overhead of OSCORE [I-D.ietf-core-object-security].

The below calculation Option Delta = '9', Sender ID = '' (empty string), and Sequence Number = '05', and is only an example. Note that Sender ID = '' (empty string) can only be used by one client per server.

OSCORE request (19 bytes, 13 bytes overhead):

```
92 09 05
ff ec ae a0 15 56 67 92 4d ff 8a 24 e4 cb 35 b9

CoAP option delta and length:
92
Option value (flag byte and sequence number):
09 05
Payload marker:
ff
Ciphertext (including encrypted code):
ec ae a0 15 56 67 92
ICV:
4d ff 8a 24 e4 cb 35 b9

The below calculation Option Delta = '9', Sender ID = '42', and
Sequence Number = '05', and is only an example.

OSCORE request (20 bytes, 14 bytes overhead):
93 09 05 42
ff ec ae a0 15 56 67 92 4d ff 8a 24 e4 cb 35 b9

CoAP option delta and length:
93
Option Value (flag byte, sequence number, and Sender ID):
09 05 42
Payload marker:
ff
Ciphertext (including encrypted code):
ec ae a0 15 56 67 92
ICV:
4d ff 8a 24 e4 cb 35 b9

The below calculation uses Option Delta = '9'.
```

OSCORE response (17 bytes, 11 bytes overhead):
90
ff ec ae a0 15 56 67 92 4d ff 8a 24 e4 cb 35 b9

CoAP delta and option length:
90
Option value:
-
Payload marker:
ff
Ciphertext (including encrypted code):
ec ae a0 15 56 67 92
ICV:
4d ff 8a 24 e4 cb 35 b9

OSCORE with the above parameters gives 13-14 bytes overhead for requests and 11 bytes overhead for responses.

Unlike DTLS and TLS, OSCORE has much smaller overhead for responses than requests.

3.7. Group OSCORE

This section analyzes the overhead of Group OSCORE [I-D.ietf-core-oscore-groupcomm].

TODO

3.8. Conclusion

DTLS 1.2 has quite a large overhead as it uses an explicit sequence number and an explicit nonce. TLS 1.2 has significantly less (but not small) overhead. TLS 1.3 has quite a small overhead. OSCORE and DTLS 1.3 (using the minimal structure) format have very small overhead.

The Generic Header Compression (6LoWPAN-GHC) can in addition to DTLS 1.2 handle TLS 1.2, and DTLS 1.2 with Connection ID. The Generic Header Compression (6LoWPAN-GHC) works very well for Connection ID and the overhead seems to increase exactly with the length of the Connection ID (which is optimal). The compression of TLS 1.2 is not as good as the compression of DTLS 1.2 (as the static dictionary only contains the DTLS 1.2 version number). Similar compression levels as for DTLS could be achieved also for TLS 1.2, but this would require different static dictionaries. For TLS 1.3 and DTLS 1.3, GHC increases the overhead. The 6LoWPAN-GHC header compression is not

available when (D)TLS is used over transports that do not use 6LoWPAN together with 6LoWPAN-GHC.

New security protocols like OSCORE, TLS 1.3, and DTLS 1.3 have much lower overhead than DTLS 1.2 and TLS 1.2. The overhead is even smaller than DTLS 1.2 and TLS 1.2 over 6LoWPAN with compression, and therefore the small overhead is achieved even on deployments without 6LoWPAN or 6LoWPAN without compression. OSCORE is lightweight because it makes use of CoAP, CBOR, and COSE, which were designed to have as low overhead as possible.

Note that the compared protocols have slightly different use cases. TLS and DTLS are designed for the transport layer and are terminated in CoAP proxies. OSCORE is designed for the application layer and protects information end-to-end between the CoAP client and the CoAP server. Group OSCORE is designed for group communication and protects information between a CoAP client and any number of CoAP servers.

4. Security Considerations

This document is purely informational.

5. IANA Considerations

This document has no actions for IANA.

6. Informative References

[I-D.ietf-core-object-security]
Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security for Constrained RESTful Environments (OSCORE)", draft-ietf-core-object-security-15 (work in progress), August 2018.

[I-D.ietf-core-oscore-groupcomm]
Tiloca, M., Selander, G., Palombini, F., and J. Park, "Group OSCORE - Secure Group Communication for CoAP", draft-ietf-core-oscore-groupcomm-03 (work in progress), October 2018.

[I-D.ietf-tls-dtls-connection-id]

Rescorla, E., Tschofenig, H., Fossati, T., and T. Gondrom, "Connection Identifiers for DTLS 1.2", draft-ietf-tls-dtls-connection-id-02 (work in progress), October 2018.

[I-D.ietf-tls-dtls13]
Rescorla, E., Tschofenig, H., and N. Modadugu, "The Datagram Transport Layer Security (DTLS) Protocol Version 1.3", draft-ietf-tls-dtls13-30 (work in progress), November 2018.

[I-D.schaad-ace-tls-cbor-handshake]
Schaad, J., "TLS Handshake in CBOR", draft-schaad-ace-tls-cbor-handshake-00 (work in progress), March 2019.

[I-D.selander-ace-cose-ecdhe]
Selander, G., Mattsson, J., and F. Palombini, "Ephemeral Diffie-Hellman Over COSE (EDHOC)", draft-selander-ace-cose-ecdhe-12 (work in progress), February 2019.

[IoT-Cert]
Forsby, F., "Digital Certificates for the Internet of Things", June 2017, <https://kth.diva-portal.org/smash/get/diva2:1153958/FULLTEXT01.pdf>.

[OlegHahm-ghc]
Hahm, O., "Generic Header Compression", July 2016, <https://github.com/OlegHahm/ghc>.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <https://www.rfc-editor.org/info/rfc5246>.

[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <https://www.rfc-editor.org/info/rfc6347>.

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

[RFC7400] Bormann, C., "6LoWPAN-GHC: Generic Header Compression for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs)", RFC 7400, DOI 10.17487/RFC7400, November 2014, <https://www.rfc-editor.org/info/rfc7400>.

[RFC7924] Santesson, S. and H. Tschofenig, "Transport Layer Security (TLS) Cached Information Extension", RFC 7924, DOI 10.17487/RFC7924, July 2016, <https://www.rfc-editor.org/info/rfc7924>.

[RFC7925] Tschofenig, H., Ed. and T. Fossati, "Transport Layer Security (TLS) / Datagram Transport Layer Security (DTLS) Profiles for the Internet of Things", RFC 7925, DOI 10.17487/RFC7925, July 2016, <https://www.rfc-editor.org/info/rfc7925>.

[RFC8323] Bormann, C., Lemay, S., Tschofenig, H., Hartke, K., Silverajan, B., and B. Raymor, Ed., "CoAP (Constrained Application Protocol) over TCP, TLS, and WebSockets", RFC 8323, DOI 10.17487/RFC8323, February 2018, <https://www.rfc-editor.org/info/rfc8323>.

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

[Ulfheim-TLS13] Driscoll, M., "Every Byte Explained The Illustrated TLS 1.3 Connection", March 2018, <https://tls13.ulfheim.net>.

Acknowledgments

The authors want to thank Ari Keraenen, Carsten Bormann, Goeran Selander, and Hannes Tschofenig for comments and suggestions on previous versions of the draft.

All 6LoWPAN-GHC compression was done with [OlegHahm-ghc].

Authors' Addresses

John Mattsson

Ericsson AB

Email: john.mattsson@ericsson.com

Francesca Palombini
Ericsson AB

Email: francesca.palombini@ericsson.com