

Workgroup: LWIG Working Group  
Internet-Draft:  
draft-ietf-lwig-security-protocol-  
comparison-07  
Published: 24 January 2023  
Intended Status: Informational  
Expires: 28 July 2023  
Authors: J. Preuß Mattsson    F. Palombini    M. Vučinić  
         Ericsson               Ericsson       INRIA  
**Comparison of CoAP Security Protocols**

## 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. Small message sizes are very important for reducing energy consumption, latency, and time to completion in constrained radio network such as Low-Power Wide Area Networks (LPWANS). The analyzed security protocols are DTLS 1.2, DTLS 1.3, TLS 1.2, TLS 1.3, cTLS, 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 28 July 2023.

## Copyright Notice

Copyright (c) 2023 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of

publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

## Table of Contents

- [1. Introduction](#)
- [2. Overhead of Key Exchange Protocols](#)
  - [2.1. Summary](#)
  - [2.2. DTLS 1.3](#)
    - [2.2.1. Message Sizes RPK + ECDHE](#)
    - [2.2.2. Message Sizes PSK + ECDHE](#)
    - [2.2.3. Message Sizes PSK](#)
    - [2.2.4. Cached Information](#)
    - [2.2.5. Resumption](#)
    - [2.2.6. DTLS Without Connection ID](#)
    - [2.2.7. Raw Public Keys](#)
  - [2.3. TLS 1.3](#)
    - [2.3.1. Message Sizes RPK + ECDHE](#)
    - [2.3.2. Message Sizes PSK + ECDHE](#)
    - [2.3.3. Message Sizes PSK](#)
  - [2.4. TLS 1.2 and DTLS 1.2](#)
  - [2.5. cTLS](#)
  - [2.6. EDHOC](#)
    - [2.6.1. Message Sizes RPK](#)
    - [2.6.2. Summary](#)
  - [2.7. Conclusion](#)
- [3. Overhead for Protection of Application Data](#)
  - [3.1. Summary](#)
  - [3.2. DTLS 1.2](#)
    - [3.2.1. DTLS 1.2](#)
    - [3.2.2. DTLS 1.2 with 6LoWPAN-GHC](#)
    - [3.2.3. DTLS 1.2 with Connection ID](#)
    - [3.2.4. DTLS 1.2 with Connection ID and 6LoWPAN-GHC](#)
  - [3.3. DTLS 1.3](#)
    - [3.3.1. DTLS 1.3](#)
    - [3.3.2. DTLS 1.3 with 6LoWPAN-GHC](#)
    - [3.3.3. DTLS 1.3 with Connection ID](#)
    - [3.3.4. DTLS 1.3 with Connection ID and 6LoWPAN-GHC](#)
  - [3.4. TLS 1.2](#)
    - [3.4.1. TLS 1.2](#)
    - [3.4.2. TLS 1.2 with 6LoWPAN-GHC](#)
  - [3.5. TLS 1.3](#)
    - [3.5.1. TLS 1.3](#)
    - [3.5.2. TLS 1.3 with 6LoWPAN-GHC](#)
  - [3.6. OSCORE](#)

- [3.7. Group OSCORE](#)
- [3.8. Conclusion](#)
- [4. Security Considerations](#)
- [5. IANA Considerations](#)
- [6. Informative References](#)
- [Acknowledgments](#)
- [Authors' Addresses](#)

## 1. Introduction

Small message sizes are very important for reducing energy consumption, latency, and time to completion in constrained radio network such as Low-Power Personal Area Networks (LPPANs) and Low-Power Wide Area Networks (LPWANs). Constrained radio networks are not only characterized by very small frame sizes on the order of tens of bytes transmitted a few times per day at ultra-low speeds, but also high latency, and severe duty cycles constraints. Some constrained radio networks are also multi-hop where the already small frame sizes are additionally reduced for each additional hop. Too large payload sizes can easily lead to unacceptable completion times due to fragmentation into a large number of frames and long waiting times between frames can be sent (or resent in the case of transmission errors). In constrained radio networks, the processing energy costs are typically almost negligible compared to the energy costs for radio and the energy costs for sensor measurement. Keeping the number of bytes or frames low is also essential for low latency and time to completion as well as efficient use of spectrum to support a large number of devices. For an overview of LPWANs and their limitations, see [[RFC8376](#)].

To reduce overhead, processing, and energy consumption in constrained radio networks, IETF has created several working groups and technologies for constrained networks, e.g., (here technologies in parenthesis when the name is different from the working group): 6Lo, 6LoWPAN, 6TiSCH, ACE, CBOR, CoRE (CoAP, OSCORE), COSE, LAKE (EDHOC), LPWAN (SCHC), ROLL (RPL), and TLS (cTLS). Compact formats and protocol have also been suggested as a way to decrease the energy consumption of Internet Applications and Systems in general [[E-impact](#)].

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 UDP [[RFC7252](#)] and TCP [[RFC8323](#)]. The analyzed security protocols are DTLS 1.2 [[RFC6347](#)], DTLS 1.3 [[RFC9147](#)], TLS 1.2 [[RFC5246](#)], TLS 1.3 [[RFC8446](#)], cTLS [[I-D.ietf-tls-ctls](#)], EDHOC [[I-D.ietf-lake-edhoc](#)] [[I-D.ietf-core-oscore-edhoc](#)], OSCORE [[RFC8613](#)], and Group OSCORE [[I-D.ietf-core-oscore-groupcomm](#)].

The protocols are analyzed with different algorithms and options. The DTLS and TLS record layers are analyzed with and without 6LoWPAN-GHC compression [RFC7400]. DTLS is analyzed with and without Connection ID [RFC9146]. Readers are expected to be familiar with some of the terms described in RFC 7925 [RFC7925], such as Integrity Check Value (ICV). [Section 2](#) compares the overhead of mutually authenticated key exchange, while [Section 3](#) covers the overhead for protection of application data.

Readers of this document also might be interested in the following documents: [Illustrated-TLS12], [Illustrated-TLS13], [Illustrated-DTLS13], and [I-D.ietf-lake-traces] gives an explanation of every byte in example TLS 1.2, TLS 1.3, DTLS 1.3, and EDHOC instances. [RFC9191] looks at potential tools available for overcoming the deployment challenges induced by large certificates and long certificate chains and discusses solutions available to overcome these challenges. [I-D.ietf-cose-cbor-encoded-cert] gives examples of IoT and Web certificates as well as examples on how effective C509 an TLS certificate compression [RFC8879] is at compressing example certificate and certificate chains. [I-D.mattsson-tls-compact-ecc] proposes new optimized encodings for key exchange and signatures with P-256 in TLS 1.3.

## 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:

- \*The overhead calculations in this section use an 8 bytes ICV (e.g., AES\_128\_CCM\_8 or AES-CCM-16-64-128) or 16 bytes (e.g., AES-CCM, AES-GCM, or ChaCha20-Poly1305).

- \*A minimum number of algorithms and cipher suites is offered. The algorithm used/offered are P-256 or Curve25519, ECDSA with P-256 and SHA-256 or Ed25519, AES-CCM\_8, and SHA-256.

- \*The length of key identifiers are 1 byte.

- \*The length of connection identifiers are 1 byte.

- \*DTLS handshake message fragmentation is not considered.

- \*As many (D)TLS handshake messages as possible are sent in a single record.

- \*Only mandatory (D)TLS extensions are included.

The choices of algorithms are based on the profiles in [\[RFC7925\]](#), [\[I-D.ietf-uta-tls13-iot-profile\]](#), and [\[I-D.ietf-core-oscure-edhoc\]](#).

[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 and cTLS 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 TLS, DTLS, and cTLS overhead is dependent on the group used for key exchange and the signature algorithm. secp256r1 and ecdsa\_secp256r1\_sha256 have less optimized encoding than x25519, ed25519, and [\[I-D.mattsson-tls-compact-ecc\]](#).

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 DTLS 1.3, cTLS, and EDHOC handshakes with connection ID and the mandatory to implement algorithms CCM\_8, P-256, and ECDSA [\[I-D.ietf-uta-tls13-iot-profile\]](#) [\[I-D.ietf-core-oscure-edhoc\]](#). EDHOC is typically sent over CoAP which would add 4 bytes to flight #1 and #2 and 5 or 20 bytes to flight #3 depending on if OSCORE is used [\[I-D.ietf-core-oscure-edhoc\]](#).

Flight	#1	#2	#3	Total
DTLS 1.3 - RPKs, ECDHE	185	454	255	894
DTLS 1.3 - Compressed RPKs, ECDHE	185	422	223	830
DTLS 1.3 - Cached RPK, PRK, ECDHE	224	402	255	881
DTLS 1.3 - Cached X.509, RPK, ECDHE	218	396	255	869
DTLS 1.3 - PSK, ECDHE	219	226	56	501
DTLS 1.3 - PSK	136	153	56	345
EDHOC - X.509s, Signature, x5t, ECDHE	37	115	90	242
EDHOC - RPKs, Signature, kid, ECDHE	37	102	77	216
EDHOC - X.509s, Static DH, x5t, ECDHE	37	58	33	128
EDHOC - RPKs, Static DH, kid, ECDHE	37	45	19	101

Figure 1: Comparison of message sizes in bytes with CCM\_8, P-256, and ECDSA and with Connection ID

[Figure 2](#) compares of message sizes of DTLS 1.3 [[RFC9147](#)] and TLS 1.3 [[RFC8446](#)] handshakes without connection ID but with the same algorithms CCM\_8, P-256, and ECDSA. DTLS is typically sent over 8 bytes UDP datagram headers while TLS is typically sent over 20 bytes TCP segment headers. TCP also uses some more bytes for additional messages used in TCP internally.

Flight	#1	#2	#3	Total
DTLS 1.3 - RPKs, ECDHE	179	447	254	880
DTLS 1.3 - PSK, ECDHE	213	219	55	487
DTLS 1.3 - PSK	130	146	55	331
TLS 1.3 - RPKs, ECDHE	162	394	233	789
TLS 1.3 - PSK, ECDHE	196	190	50	436
TLS 1.3 - PSK	113	117	50	280
cTLS - X.509s by reference, ECDHE	104	195	96	395
cTLS - PSK, ECDHE	105	119	20	226
cTLS - PSK	40	58	20	118

Figure 2: Comparison of message sizes in bytes with CCM\_8, secp256r1, and ecdsa\_secp256r1\_sha256 or PSK and without Connection ID

[Figure 3](#) is the same as [Figure 2](#) but with more efficiently encoded key shares and signatures such as x25519 and ed25519. The algorithms in [[I-D.mattsson-tls-compact-ecc](#)] with point compressed secp256r1 RPKs would add 15 bytes to #2 and #3 in the rows with RPKs.

Flight	#1	#2	#3	Total
DTLS 1.3 - RPKs. ECDHE	146	360	200	706
DTLS 1.3 - PSK. ECDHE	180	186	55	421
DTLS 1.3 - PSK	130	146	55	331
TLS 1.3 - RPKs. ECDHE	129	307	179	615
TLS 1.3 - PSK. ECDHE	163	157	50	370
TLS 1.3 - PSK	113	117	50	280
cTLS - X.509s bv reference. ECDHE	71	155	89	315
cTLS - PSK. ECDHE	72	86	20	178
cTLS - PSK	40	58	20	118

Figure 3: Comparison of message sizes in bytes with CCM\_8, x25519, and ed25519 or PSK and without Connection ID

The numbers in [Figure 2](#), [Figure 2](#), and [Figure 3](#) were calculated with 8 bytes tags which is the mandatory to implement in [\[I-D.ietf-uta-tls13-iot-profile\]](#) and [\[I-D.ietf-core-oscure-edhoc\]](#). If 16 bytes tag are used, the numbers in the #2 and #3 columns increases with 8 and the numbers in the Total column increases with 16.

## 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 replaced 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, CCM\_8, P-256, and ECDSA and a Connection ID of 1 byte are used.

#### **2.2.1.1. Flight #1**



Record Header - DTLSPlaintext (13 bytes):

16 fe fd EE EE SS SS SS SS SS LL LL

Handshake Header - Client Hello (12 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 (secp256r1) (8 bytes):

00 0a 00 04 00 02 00 17

Extension - Signature Algorithms (ecdsa\_secp256r1\_sha256)  
(8 bytes):

00 0d 00 04 00 02 04 03

Extension - Key Share (secp256r1) (75 bytes):

00 33 00 27 00 25 00 1d 00 41  
04 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

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 02 01 02

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

00 14 00 02 01 02

Extension - Connection Identifier (42) (6 bytes):

00 36 00 02 01 42

13 + 12 + 2 + 32 + 1 + 1 + 4 + 2 + 2 + 8 + 8 + 75 + 7 + 6 + 6 + 6  
= 185 bytes

DTLS 1.3 RPK + ECDHE flight #1 gives 185 bytes of overhead. With efficiently encoded key share such as x25519 or [\[I-D.mattsson-tls-compact-ecc\]](#) the overhead is  $185 - 33 = 152$  bytes.

#### 2.2.1.2. Flight #2

Record Header - DTLSPlaintext (13 bytes):

16 fe fd EE EE SS SS SS SS SS LL LL

Handshake Header - Server Hello (12 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 (secp256r1) (73 bytes):

00 33 00 45 00 1d 00 41  
04 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

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

00 2b 00 02 03 04

Extension - Connection Identifier (43) (6 bytes):

00 36 00 02 01 43

Record Header - DTLSCiphertext (3 bytes):

HH 42 SS

Handshake Header - Encrypted Extensions (12 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 (12 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 (12 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 (Uncompressed secp256r1 RPK) (91 bytes):  
30 59 30 13 ... // DER encoded RPK, See Section 2.2.7.

Certificate Extensions (2 bytes):  
00 00

Handshake Header - Certificate Verify (12 bytes):  
0f LL LL LL SS SS 00 00 00 LL LL LL

Signature (ecdsa\_secp256r1\_sha256) (average 75 bytes):  
04 03 LL LL  
30 LL 02 LL ... 02 LL ... // DER encoded signature

Handshake Header - Finished (12 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 + 137 + 3 + 26 + 23 + 112 + 87 + 44 + 1 + 8 = 454 bytes

DTLS 1.3 RPK + ECDHE flight #2 gives 454 bytes of overhead. With a point compressed secp256r1 RPK the overhead is  $454 - 32 = 422$  bytes, see [Section 2.2.7](#). With an ed25519 RPK and signature the overhead is  $454 - 47 - 7 = 400$  bytes. With an efficiently encoded key share such as x25519 or [\[I-D.mattsson-tls-compact-ecc\]](#) the overhead is  $454 - 33 = 421$  bytes. With an efficiently encoded signature such [\[I-D.mattsson-tls-compact-ecc\]](#) the overhead is  $454 - 7 = 447$  bytes. With x25519 and ed25519 the overhead is  $454 - 47 - 33 - 7 = 367$  bytes.



### 2.2.1.3. Flight #3

Record Header (3 bytes): // DTLS Ciphertext  
ZZ 43 SS

Handshake Header - Certificate (12 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 (Uncompressed secp256r1 RPK) (91 bytes):  
30 59 30 13 ... // DER encoded RPK, See Section 2.2.7.

Certificate Extensions (2 bytes):  
00 00

Handshake Header - Certificate Verify (12 bytes):  
0f LL LL LL SS SS 00 00 00 LL LL LL

Signature (ecdsa\_secp256r1\_sha256) (average 75 bytes):  
04 03 LL LL  
30 LL 02 LL ... 02 LL ... // // DER encoded signature

Handshake Header - Finished (12 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

$3 + 112 + 87 + 44 + 1 + 8 = 255$  bytes

DTLS 1.3 RPK + ECDHE flight #3 gives 255 bytes of overhead. With a point compressed secp256r1 RPK the overhead is  $255 - 32 = 223$  bytes, see [Section 2.2.7](#). With an ed25519 RPK and signature the overhead is  $255 - 47 - 7 = 201$  bytes. With an efficiently encoded signature such

as [[I-D.mattsson-tls-compact-ecc](#)] the overhead is  $255 - 7 = 248$  bytes.

## 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:

$185 + 6 + 48 - 8 - 6 - 6 = 219$  bytes

DTLS 1.3 PSK + ECDHE flight #1 gives 219 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 (112 bytes)
- Handshake Message CertificateVerify (87 bytes)
- Handshake Message CertificateRequest (23 bytes)
- Extension - Client Certificate Type (Raw Public Key) (6 bytes)
- Extension - Server Certificate Type (Raw Public Key) (6 bytes)

In total:

$$454 + 6 - 112 - 87 - 23 - 6 - 6 = 226 \text{ bytes}$$

DTLS 1.3 PSK + ECDHE flight #2 gives 226 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 (112 bytes)
- Handshake Message Certificate Verify (87 bytes)

In total:

$$255 - 112 - 87 = 56 \text{ bytes}$$

DTLS 1.3 PSK + ECDHE flight #3 gives 56 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 (75 bytes)

In total:

$$219 - 8 - 75 = 136 \text{ bytes}$$

DTLS 1.3 PSK flight #1 gives 136 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 (73 bytes)

In total:

$$226 - 73 = 153 \text{ bytes}$$

DTLS 1.3 PSK flight #2 gives 153 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 56 bytes of overhead.

#### 2.2.4. Cached Information

In this section, we consider the effect of [\[RFC7924\]](#) on the message size overhead.

Cached information can be used to use a cached server certificate from a previous connection and move bytes from flight #2 to flight #1. The cached certificate can be a RPK or X.509.

The differences compared to [Section 2.2.1](#) are the following.

##### 2.2.4.1. Flight #1

For the flight #1, the following is added:

- + Extension - Client Cached 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
```

Giving a total of:

$$185 + 39 = 224 \text{ bytes}$$

In the case the cached certificate is X.509 the following is removed:

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

Giving a total of:

$224 - 6 = 218$  bytes

#### **2.2.4.2. Flight #2**

For the flight #2, the following is added:

+ Extension - Server Cached Information (7 bytes):

00 19 LL LL LL LL 01

And the following is reduced:

- Server Certificate (91 bytes -> 32 bytes)

Giving a total of:

$454 + 7 - 59 = 402$  bytes

In the case the cached certificate is X.509 the following is removed:

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

Giving a total of:

$402 - 6 = 396$  bytes

#### **2.2.5. Resumption**

To enable resumption, a 4th flight with the handshake message New Session Ticket is added to the DTLS handshake.

Record Header - DTLS Ciphertext (3 bytes):

HH 42 SS

Handshake Header - New Session Ticket (12 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

$3 + 12 + 4 + 4 + 2 + 6 + 2 + 8 = 41$  bytes

Enabling resumption adds 41 bytes to the initial DTLS handshake. The resumption handshake is an ordinary PSK handshake with or without ECDHE.

#### 2.2.6. DTLS Without Connection ID

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

DTLS 1.3 flight #1: -6 bytes

DTLS 1.3 flight #2: -7 bytes

DTLS 1.3 flight #3: -1 byte

#### 2.2.7. Raw Public Keys

This section illustrates the format of P-256 (secp256r1) SubjectPublicKeyInfo [RFC5480] with and without point compression as well as an ed25519 SubjectPublicKeyInfo. Point compression in SubjectPublicKeyInfo is standardized in [RFC5480] and is therefore theoretically possible to use in PRKs and X.509 certificates used in (D)TLS but does not seem to be supported by (D)TLS implementations.

#### **2.2.7.1. secp256r1 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. secp256r1 SubjectPublicKeyInfo With Point Compression**

```
0x30 // Sequence
0x39 // Size 57

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
0x22 // Size 34
0x00 // Unused bits 0
0x03 // Compressed
..... 32 bytes X

Total of 59 bytes
```

### 2.2.7.3. ed25519 SubjectPublicKeyInfo

```
0x30 // Sequence
0x2A // Size 42

0x30 // Sequence
0x05 // Size 5
0x06 0x03 0x2B 0x65 0x70
    // OID 1.3.101.112 (ed25519)

0x03 // Bit string
0x21 // Size 33
0x00 // Unused bits 0
..... 32 bytes
```

Total of 44 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 a different record header, 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.ietf-tls-ctls](#)]).

### 2.3.1. Message Sizes RPK + ECDHE

In this section, CCM\_8, x25519, and ed25519 are used.



### 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 (ed25519)  
(8 bytes):

00 0d 00 04 00 02 08 07

Extension - Key Share (x25519) (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 (x25519) (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 (ed25519)  
(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 (ed25519 RPK) (44 bytes):

30 2A 30 05 ... // DER encoded RPK, see Section 2.2.7.

Certificate Extensions (2 bytes):

00 00

Handshake Header - Certificate Verify (4 bytes):

0f LL LL LL

Signature (ed25519) (68 bytes):

08 07 LL LL

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):

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

$$5 + 90 + 5 + 18 + 15 + 57 + 72 + 36 + 1 + 8 = 307 \text{ bytes}$$

TLS 1.3 RPK + ECDHE flight #2 gives 307 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 (ed25519 RPK) (44 bytes):

30 2A 30 05 ... // DER encoded RPK, see Section 2.2.7.

Certificate Extensions (2 bytes):

00 00

Handshake Header - Certificate Verify (4 bytes):

0f LL LL LL

Signature (ed25519) (68 bytes):

08 07 LL LL

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 + 57 + 72 + 36 + 1 + 8 = 179 bytes

TLS 1.3 RPK + ECDHE flight #3 gives 179 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 163 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 (57 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:

$307 - 57 - 72 - 15 - 6 - 6 + 6 = 157$  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 (57 bytes)
- Handshake Message Certificate Verify (72 bytes)

In total:

$179 - 57 - 72 = 50$  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 113 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 50 bytes of overhead.

#### **2.4. TLS 1.2 and DTLS 1.2**

The TLS 1.2 and DTLS 1.2 handshakes are not analyzed in detail in this document. One rough comparison on expected size between the TLS 1.2 and TLS 1.3 handshakes can be found by counting the number of bytes in the example handshakes of [[Illustrated-TLS12](#)] and [[Illustrated-TLS13](#)]. In these examples the server authenticates with a certificate and the client is not authenticated.

In TLS 1.2 the number of bytes in the four flights are 170, 1188, 117, and 75 for a total of 1550 bytes. In TLS 1.3 the number of bytes in the three flights are 253, 1367, and 79 for a total of 1699 bytes. In general, the (D)TLS 1.2 and (D)TLS 1.3 handshakes can be expected to have similar number of bytes.

#### **2.5. cTLS**

The cTLS specification [[I-D.ietf-tls-ctls](#)] has a single example in Appendix A. The numbers given are correct for the algorithms CCM\_8, x25519, and ed25519 but are missing overhead from CTLSCiphertext which adds 11 bytes to flight #2 and #3. The sizes for flights are therefore 71, 155 (66 + 79 + 11), and 89 (78 + 11) bytes for a total of 315 bytes.

Using secp256r1 instead x25519 add 33 bytes to flight #1 and flight #2.

Using ecdsa\_secp256r1\_sha256 instead ed25519 add an average of 7 bytes to flight #2 and flight #3.

Using PSK authentication instead of ed25519 add 1 byte (psk identifier) to flight #1 and removes 69 bytes from flight #2 and #3.

Using Connection ID adds 1 byte to flight #1 and #3, and 2 bytes to flight #2.

#### **2.6. EDHOC**

This section gives an estimate of the message sizes of EDHOC [[I-D.ietf-lake-edhoc](#)] authenticated with static Diffie-Hellman keys and where the static Diffie-Hellman are identified with a key identifier (kid). All examples are given in CBOR diagnostic notation

and hexadecimal, and are based on the test vectors in Section 4 of [\[I-D.ietf-lake-traces\]](#).

### 2.6.1. Message Sizes RPK

#### 2.6.1.1. message\_1

```
message_1 = (  
    3,  
    2,  
    h'8af6f430ebe18d34184017a9a11bf511c8dff8f834730b96c1b7c8dbca2f  
    c3b6',  
    -24  
)
```

message\_1 (37 bytes):

```
03 02 58 20 8a f6 f4 30 eb e1 8d 34 18 40 17 a9 a1 1b f5 11 c8  
df f8 f8 34 73 0b 96 c1 b7 c8 db ca 2f c3 b6 37
```

#### 2.6.1.2. message\_2

```
message_2 = (  
    h'419701D7F00A26C2DC587A36DD752549F33763C893422C8EA0F955A13A4F  
    F5D5042459E2DA6C75143F35',  
    -8  
)
```

message\_2 (45 bytes):

```
58 2a 41 97 01 d7 f0 0a 26 c2 dc 58 7a 36 dd 75 25 49 f3 37  
63 c8 93 42 2c 8e a0 f9 55 a1 3a 4f f5 d5 04 24 59 e2 da 6c  
75 14 3f 35 27
```

#### 2.6.1.3. message\_3

```
message_3 = (  
    h'C2B62835DC9B1F53419C1D3A2261EEED3505'  
)
```

message\_3 (19 bytes):

```
52 c2 b6 28 35 dc 9b 1f 53 41 9c 1d 3a 22 61 ee ed 35 05
```

### 2.6.2. Summary

Based on the example above it is relatively easy to calculate numbers also for EDHOC authenticated with signature keys and for authentication keys identified with a SHA-256/64 hash (x5t). Signatures increase the size of flight #2 and #3 with  $(64 - 8 + 1)$  bytes while x5t increases the size with 13-14 bytes. The typical message sizes for the previous example and for the other combinations are summarized in [Figure 4](#). Note that EDHOC treats

authentication keys stored in RPK and X.509 in the same way. More detailed examples can be found in [[I-D.ietf-lake-traces](#)].

	Static DH Kevs		Signature Kevs	
	kid	x5t	kid	x5t
message 1	37	37	37	37
message 2	45	58	102	115
message 3	19	33	77	90
Total	101	128	216	242

Figure 4: Typical message sizes in bytes

## 2.7. 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 bytes in each frame that is available for the key exchange protocol may depend on the underlying protocol layers as well as on the number of hops in multi-hop networks. The packet loss may depend on how many other devices 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 an 8 bytes ICV (e.g., AES\_128\_CCM\_8 or AES-CCM-16-64-128) or 16 bytes (e.g., AES-CCM, AES-GCM, or ChaCha20-Poly1305), 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 (where applicable), 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, ID Context, and Sequence Number (where applicable). cTLS

uses the DTLS 1.3 record layer. 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	11	11
DTLS 1.2 (GHC)	16	16	16
DTLS 1.3 (GHC)	12	12	12
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
Group OSCORE pairwise request	14	15	16
Group OSCORE pairwise response	11	11	11

Figure 5: Overhead (8 bytes ICV) 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
Group OSCORE pairwise request	14	15	16
Group OSCORE pairwise response	11	13	14

Figure 6: Overhead (8 bytes ICV) 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	
Group OSCORE pairwise request	7	
Group OSCORE pairwise response	4	

Figure 7: Overhead (excluding ICV) in bytes (Connection/Sender ID = ' ', Sequence Number = '05')

The numbers in [Figure 5](#), [Figure 6](#), and {fig-overhead3} do not consider the different Token processing requirements for clients [[RFC9175](#)] required for secure operation as motivated by [[I-D.ietf-core-attacks-on-coap](#)]. As reuse of Tokens is easier in OSCORE than DTLS, OSCORE might have slightly lower overhead than DTLS 1.3 for long connection even if DTLS 1.3 has slightly lower overhead than OSCORE for short connections.

The numbers in [Figure 5](#), [Figure 6](#), and {fig-overhead3} do not consider underlying layers. DTLS is typically sent over 8 bytes UDP datagram headers while TLS is typically sent over 20 bytes TCP segment headers. TCP also uses some more bytes for additional

messages used in TCP internally. The total overhead for DTLS 1.3 over UDP is significantly less than TLS 1.3 over TCP.

The numbers in [Figure 5](#) and [Figure 6](#) were calculated with 8 bytes ICV which is the mandatory to implement in [\[I-D.ietf-uta-tls13-iot-profile\]](#), and [\[I-D.ietf-core-oscure-edhoc\]](#). If 16 bytes tag are used, all numbers increases with 8.

### 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 [[RFC9146](#)]. The overhead calculations in this section uses Connection ID = '42'. DTLS record 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 [[RFC9146](#)] 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 [[RFC9147](#)]. The changes compared to DTLS 1.2 are: omission of version number, merging of epoch into the first byte containing signaling bits, optional omission of length, reduction of sequence number into a 1 or 2-bytes field.

DTLS 1.3 is only analyzed with an omitted length field and with an 8-bit sequence number (see Figure 4 of [[RFC9147](#)]).

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 [[RFC9147](#)] 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 [[RFC9147](#)] with Connection ID [[RFC9146](#)].

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 [[RFC9147](#)]), 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 [[RFC9147](#)] with Connection ID [[RFC9146](#)] 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 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 [[RFC8613](#)].

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](#)]. Group OSCORE defines a pairwise mode where each member of the group can efficiently derive a symmetric pairwise key with any other member of the group for pairwise OSCORE communication. Additional requirements compared to [[RFC8613](#)] is that ID Context is always included in requests and that Sender ID is always included in responses. Assuming 1 byte ID Context and Sender ID this adds 2 bytes to requests and 1 byte to responses.

The below calculation Option Delta = '9', ID Context = '', Sender ID = '42', and Sequence Number = '05', and is only an example. ID Context = '' would be the standard for local deployments only having a single group.

OSCORE request (21 bytes, 15 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, ID Context length, sequence nr, Sender ID):

```
19 00 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' and Sender ID = '69', and is only an example.

OSCORE response (18 bytes, 12 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 (flag byte, Sender ID):  
08 69  
Payload marker:  
ff  
Ciphertext (including encrypted code):  
ec ae a0 15 56 67 92  
ICV:  
4d ff 8a 24 e4 cb 35 b9

The pairwise mode OSCORE with the above parameters gives 15 bytes overhead for requests and 12 bytes overhead for responses.

### 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. As can be seen in [Figure 7](#), Group OSCORE for pairwise communication increases the overhead of OSCORE requests with 20% and OSCORE responses with 33%.



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 communication in a group.

#### 4. Security Considerations

This document is purely informational.

#### 5. IANA Considerations

This document has no actions for IANA.

#### 6. Informative References

[E-impact] Internet Architecture Board, "Workshop on Environmental Impact of Internet Applications and Systems", December 2022, <<https://www.iab.org/activities/workshops/e-impact/>>.

##### [I-D.ietf-core-attacks-on-coap]

Mattsson, J. P., Fornehed, J., Selander, G., Palombini, F., and C. Amsüss, "Attacks on the Constrained Application Protocol (CoAP)", Work in Progress, Internet-Draft, draft-ietf-core-attacks-on-coap-02, 22 December 2022, <<https://www.ietf.org/archive/id/draft-ietf-core-attacks-on-coap-02.txt>>.

[I-D.ietf-core-oscore-edhoc] Palombini, F., Tiloca, M., Höglund, R., Hristozov, S., and G. Selander, "Profiling EDHOC for CoAP and OSCORE", Work in Progress, Internet-Draft, draft-ietf-core-oscore-edhoc-06, 23 November 2022, <<https://www.ietf.org/archive/id/draft-ietf-core-oscore-edhoc-06.txt>>.

##### [I-D.ietf-core-oscore-groupcomm]

Tiloca, M., Selander, G., Palombini, F., Mattsson, J. P., and J. Park, "Group OSCORE - Secure Group Communication for CoAP", Work in Progress, Internet-Draft, draft-ietf-core-oscore-groupcomm-17, 20 December 2022, <<https://www.ietf.org/archive/id/draft-ietf-core-oscore-groupcomm-17.txt>>.

##### [I-D.ietf-cose-cbor-encoded-cert]

Mattsson, J. P., Selander, G., Raza, S., Höglund, J., and M. Furuheid, "CBOR Encoded X.509 Certificates (C509 Certificates)", Work in Progress, Internet-Draft, draft-ietf-cose-cbor-encoded-cert-05, 10 January 2023,

<<https://www.ietf.org/archive/id/draft-ietf-cose-cbor-encoded-cert-05.txt>>.

[**I-D.ietf-lake-edhoc**] Selander, G., Mattsson, J. P., and F. Palombini, "Ephemeral Diffie-Hellman Over COSE (EDHOC)", Work in Progress, Internet-Draft, draft-ietf-lake-edhoc-18, 28 November 2022, <<https://www.ietf.org/archive/id/draft-ietf-lake-edhoc-18.txt>>.

[**I-D.ietf-lake-traces**] Selander, G., Mattsson, J. P., Serafin, M., and M. Tiloca, "Traces of EDHOC", Work in Progress, Internet-Draft, draft-ietf-lake-traces-03, 24 October 2022, <<https://www.ietf.org/archive/id/draft-ietf-lake-traces-03.txt>>.

[**I-D.ietf-tls-ctls**] Rescorla, E., Barnes, R., Tschofenig, H., and B. M. Schwartz, "Compact TLS 1.3", Work in Progress, Internet-Draft, draft-ietf-tls-ctls-07, 3 January 2023, <<https://www.ietf.org/archive/id/draft-ietf-tls-ctls-07.txt>>.

[**I-D.ietf-uta-tls13-iot-profile**] Tschofenig, H. and T. Fossati, "TLS/DTLS 1.3 Profiles for the Internet of Things", Work in Progress, Internet-Draft, draft-ietf-uta-tls13-iot-profile-05, 6 July 2022, <<https://www.ietf.org/archive/id/draft-ietf-uta-tls13-iot-profile-05.txt>>.

[**I-D.mattsson-tls-compact-ecc**] Mattsson, J. P., "Compact ECDHE and ECDSA Encodings for TLS 1.3", Work in Progress, Internet-Draft, draft-mattsson-tls-compact-ecc-03, 18 January 2023, <<https://www.ietf.org/archive/id/draft-mattsson-tls-compact-ecc-03.txt>>.

[**Illustrated-DTLS13**] Driscoll, M., "The Illustrated DTLS 1.3 Connection", n.d., <<https://dtls.xargs.org/>>.

[**Illustrated-TLS12**] Driscoll, M., "The Illustrated TLS 1.2 Connection", n.d., <<https://tls12.xargs.org/>>.

[**Illustrated-TLS13**] Driscoll, M., "The Illustrated TLS 1.3 Connection", n.d., <<https://tls13.xargs.org/>>.

[**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>>.

**[RFC5480]**

Turner, S., Brown, D., Yiu, K., Housley, R., and T. Polk, "Elliptic Curve Cryptography Subject Public Key Information", RFC 5480, DOI 10.17487/RFC5480, March 2009, <<https://www.rfc-editor.org/info/rfc5480>>.

**[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>>.

**[RFC8376]**

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

**[RFC8446]**

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

**[RFC8613]**

Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security for Constrained RESTful Environments (OSCORE)", RFC 8613, DOI 10.17487/RFC8613, July 2019, <<https://www.rfc-editor.org/info/rfc8613>>.

**[RFC8879]**

Ghedini, A. and V. Vasiliev, "TLS Certificate Compression", RFC 8879, DOI 10.17487/RFC8879, December 2020, <<https://www.rfc-editor.org/info/rfc8879>>.

**[RFC9146]**

Rescorla, E., Ed., Tschofenig, H., Ed., Fossati, T., and A. Kraus, "Connection Identifier for DTLS 1.2", RFC 9146, DOI 10.17487/RFC9146, March 2022, <<https://www.rfc-editor.org/info/rfc9146>>.

**[RFC9147]**

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

**[RFC9175]**

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

**[RFC9191]**

Sethi, M., Preuß Mattsson, J., and S. Turner, "Handling Large Certificates and Long Certificate Chains in TLS-Based EAP Methods", RFC 9191, DOI 10.17487/RFC9191, February 2022, <<https://www.rfc-editor.org/info/rfc9191>>.

## Acknowledgments

The authors want to thank Carsten Bormann, Ari Keränen, Stephan Koch, Achim Kraus, Göran Selander, and Hannes Tschofenig for comments and suggestions on previous versions of the draft.

All 6LoWPAN-GHC compression was done with [[OlegHahm-ghc](#)]. [[Illustrated-TLS13](#)] as a was a useful resource for the TLS handshake content and formatting and [[IoT-Cert](#)] was a useful resource for SubjectPublicKeyInfo formatting.

## Authors' Addresses

John Preuß Mattsson  
Ericsson AB

Email: [john.mattsson@ericsson.com](mailto:john.mattsson@ericsson.com)

Francesca Palombini

Ericsson AB

Email: [francesca.palombini@ericsson.com](mailto:francesca.palombini@ericsson.com)

Mališa Vučinić

INRIA

Email: [malisa.vucinic@inria.fr](mailto:malisa.vucinic@inria.fr)