IOTOPS Working Group — J. Preuß Mattsson
Internet-Draft — F. Palombini
Intended status: Informational — Ericsson
Expires: 6 December 2025 — M. Vučinić
INRIA
4 June 2025

Comparison of CoAP Security Protocols
draft-ietf-iotops-security-protocol-comparison-09

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 (D)TLS~~ 1.2~~,

~~DTLS 1.3, TLS 1.2, TLS 1.3, cTLS, ~~EDHOC, and 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.

Commenté [MB1]: Please expand as this is not well-known per https://www.rfc-editor.org/rpc/wiki/doku.php?id=abbrev_list

Commenté [MB2]: There is no «formal» mapping documents of some proposals listed in the document (cTLS, for example). Can we refer to those as «CoAP Security Protocol» per se?

Commenté [MB3]: Please expand

Commenté [MB4]: This is to focus on the protocol families.

Commenté [MB5]: Please expand: Generic Header Compression

About This Document

This note is to be removed before publishing as an RFC.

Status information for this document may be found at https://datatracker.ietf.org/doc/draft-ietf-iotops-security-protocol-comparison/.

Discussion of this document takes place on the IOT Operations (iotops) Working Group mailing list (mailto:iotops@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/iotops/. Subscribe at https://www.ietf.org/mailman/listinfo/iotops/.

Source for this draft and an issue tracker can be found at https://github.com/lwig-wg/protocol-comparison.

Status of This Memo

material or to cite them other than as "work in progress."

This Internet-Draft will expire on 6 December 2025.

Copyright Notice

Table of Contents

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 by 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~~ may ~~easily~~ lead to unacceptable completion times due to fragmentation into a large number of frames and long waiting times between sending each frame (or resending frames, 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] and [I-D.ietf-lake-reqs].

To reduce overhead, processing, and energy consumption in constrained radio networks, the IETF ~~has created~~specified several ~~working groups and technologies~~ protocols 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, Group OSCORE~~)~~, COSE ~~(~~COSE, C509~~)~~, ~~LAKE (~~EDHOC~~)~~, LPWAN, SCHC, ~~ROLL (~~RPL~~)~~, ~~and~~ (D)TLS~~ (DTLS~~, and cTLS~~)~~. Compact ~~formats and~~ protocol messages have also been ~~suggested~~ explored as a ~~way~~ means to decrease the energy consumption of ~~Internet Applications~~ IP-based applications and ~~Systems~~ systems, in general [RFC9547].

This document analyzes and compares the sizes of Authenticated Key Exchange (AKE) flights and the per-packet message size overheads when using different security protocols to secure CoAP over ~~UPD~~ 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 [RFC9528] [RFC9668], and OSCORE

Commenté [MB6]: Add some motivation about how/why this analysis can be used (e.g., inform deployment decisions, else?).

Commenté [MB7]: What does that mean?

Commenté [MB8]: This is expired since 2020. Do we need this? Isn't 8376 sufficient here?

Commenté [MB9]: Better to focus on what is produced, not how.

Commenté [MB10]: Please expand and cite authoritative reference for each of those.

Commenté [MB11]: Do we have a document where this is formally defined?

Commenté [MB12]: Please cite those as normative.

[RFC8613]. For comprehensiveness, the analysis also includes cTLS [I-D.ietf-tls-ctls]~~,~~ and Group OSCORE [I-D.ietf-core-oscore-groupcomm]. An AKE and a protocol for the protection of application data serve distinct purposes. An AKE is responsible for establishing secure communication channels between ~~parties~~ endpoints and negotiating cryptographic keys used for authenticated encryption. AKE protocols typically involve a series of messages exchanged between ~~communicating parties~~endpoints to authenticate each other's identities and derive shared secret keys. TLS, DTLS, and EDHOC ~~cTLS~~ handshakes ~~as well as EDHOC~~ are examples of AKEs. Protocols for protection of application data are responsible for encrypting and authenticating application-layer data to ensure its confidentiality, integrity, and replay protection during transmission. ~~The~~ TLS and DTLS record layers, OSCORE, and Group OSCORE are examples of protocols for protection of application data.

Section 3 compares the overhead of mutually authenticated key exchange protocols, while Section 4 covers the overhead of protocols for protection of application data. 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).

Readers ~~of this document~~ also might be interested in the following documents: [Illustrated-TLS12], [Illustrated-TLS13], [Illustrated-DTLS13], and [RFC9529] explain 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 and TLS certificate compression [RFC8879] is at compressing example certificate and certificate chains. [I-D.ietf-tls-cert-abridge] and [I-D.kampanakis-tls-scas-latest] describe how TLS clients or servers can reduce the size of the TLS handshake by not sending certificate authority certificates. [I-D.mattsson-tls-compact-ecc] proposes new optimized encodings for key exchange and signatures with P-256 in TLS 1.3.

X. Terminology

Flight: A flight starts with the handshake message transmission of one peer and ends with the expected response from the other peer.

2. Underlying Layers

The described overheads in Sections 3 and ~~Section~~ 4 are independent of the underlying layers as they do not consider DTLS handshake message fragmentation, how to compose DTLS handshake messages into records,

Commenté [MB13]: All these should be listed as normative.

Commenté [MB14]: To focus on the main protocols with stable specs.

Commenté [MB15]: Better to comply with CoAP terminology.

Commenté [MB16]: cTLS falls under TLS branch, anyway.

Commenté [MB17]: Move this to a terminology section

Commenté [MB18]: This smells a normative reference. Please cite that RFC accordingly.

Commenté [MB19]: What is the purpose of this discussion?

Do we need all these details for the purpose of this draft? IF so, consider moving those to a dedicated section with clear goal.

Commenté [MB20]: Consider listing key terms here and how the underlying layers influence the choice of application plaintext sizes. The complete overhead for all layers depends on the combination of layers as well as assumptions regarding the devices and applications; these considerations are ~~and is~~ out of scope of the document.

This section gives a short overview of the overheads of UDP, TCP, and CoAP to give the reader a high-level overview.

DTLS and cTLS are 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. EDHOC is ~~typically~~ sent over CoAP which would typically add 12 bytes to flight #1, 5 bytes to flight #2, and 1 byte to flight #3 when used over OSCORE with the EDHOC + OSCORE combined request according to [RFC9668], see Appendix A. If EDHOC is used without OSCORE, the overhead would typically be 12 bytes to flight #1 and #3 and 5 bytes to flight #2. OSCORE and Group OSCORE are part of CoAP and are typically sent over UDP. A comparison of the total size for DTLS and EDHOC when transported over IEEE 802.15.4 and 6LoWPAN is provided in [Performance].

IPv6, UDP, and CoAP can be compressed with Static Context Header Compression (SCHC) for the Constrained Application Protocol (CoAP) [RFC8824][I-D.ietf-schc-8824-update]. The use of SCHC can significantly reduce the overhead. [SCHC-eval] gives an evaluation of how SCHC reduces this overhead for OSCORE and the DTLS 1.2 record layer when used in ~~four of the most widely used~~several LPWAN radio technologies

Fragmentation can ~~significantly~~ also increase the total overhead as many more packet headers have to be sent. CoAP, (D)TLS handshake, and IP support fragmentation. If, how, and where fragmentation is done depends heavily on the underlying layers.

3. Overhead of Authenticated Key Exchange Protocols

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

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

* The overhead calculations in this section use an ~~8~~ 8-byte ~~bytes~~ ICV (e.g., AES_128_CCM_8 [RFC6655] or AES-CCM-16-64-128 [RFC9053]) or a ~~16~~ 16-bytes~~s,~~ e.g., AES-CCM [SP-800-38C], AES-GCM [SP-800-38D], or ChaCha20-Poly1305 [RFC7539]).

* A minimum number of algorithms and cipher suites is offered. The algorithm used~~/offered~~ are: P-256 [SP-800-186] or Curve25519 [RFC7748]; ECDSA [FIPS-186-5] with P-256 and SHA-256, or Ed25519 [RFC8032]; AES-CCM_8; and SHA-256 [FIPS-180-4].

* The length of key identifiers (kids) is 1 byte.

Commenté [MB21]: cTLS can also be used with TCP. Why it is listed only with DTLS and not with TLS as well?

Overall, I'm for not citing much cTLS.

Commenté [MB22]: What does that mean?

Commenté [MB23]: Not yet introduced at this stage. Difficult to digest the intent here.

The rationale for the flights used in the document should be explained. I don't see these defined as such for example in rfc9147.html#name-dtls-handshake-flights

Commenté [MB24]: Of this document? Make the citation explicit to avoid confusion with RFC9668

Commenté [MB25]: May need to define what is «typical» as the text includes many many «typically»

Commenté [MB26]: This should be expanded at the first use.

Commenté [MB27]: As this may not age well.

Commenté [MB28]: Not sure this is always true. For example, the transport header may be only present in the first fragment.

As we don't deal with how fragmentation is done, I suggest we simply remove this.

Commenté [MB29]: Should this be AEAD_AES_128_CCM_8 ?

Commenté [MB30]: This is not used in the analysis.

Commenté [MB31]: expand

* The length of Connection IDs (CIDs)~~connection identifiers~~ is 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.

* Denial-of-Service (DoS) protection with DTLS HelloVerifyRequest [RFC6347], HelloRetryRequest [RFC9147]~~,~~ or the CoAP Echo Option [RFC9175] is not considered.

The choices of algorithms are based on the profiles in [RFC7925], [I-D.ietf-uta-tls13-iot-profile], and on the used EDHOC application profiles, see Section 3.9 of [RFC9528]. Many DTLS implementations splits flight #2 in two records.

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, EDHOC, and cTLS overhead is dependent on the CID parameter ~~Connection ID~~. The EDHOC and cTLS overhead is also dependent on the key or certificate identifiers included. Key identifiers are byte strings used to identity a cryptographic key~~ and~~. ~~certificate~~Certificate identifiers are used to identify a certificate. If ~~8~~ 8-byte~~s~~ identifiers are used instead of ~~1~~ 1-byte, the Raw Public Key (RPK) numbers for ~~flight~~ flights #2 and #3 increase by 7 bytes and the Pre-shared Key (PSK) numbers for flight #1 increase by 7 bytes.

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].

Figure 1 compares the message sizes of DTLS 1.3, cTLS, and EDHOC handshakes with ~~connection ID~~CID and the ~~mandatory~~ mandatory-~~to~~ to-implement algorithms CCM_8, P-256, and ECDSA [I-D.ietf-uta-tls13-iot-profile] [RFC9528].

Editor's note: This version of the document analyses the -10 version of cTLS, which seems relatively stable.

===

a mis en forme : Surlignage

Commenté [MB32]: Any reference to cite for those?

Commenté [MB33]: Any refs?

Commenté [MB34]: Do we need to cite this one as this is still individual work?

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 - Signature X.509s, x5t, ECDHE	37	115	90	242
EDHOC - Signature RPKs, kid, ECDHE	37	102	77	216
EDHOC - Static DH X.509s, x5t, ECDHE	37	58	33	128
EDHOC - Static DH RPKs, kid, ECDHE	37	45	19	101

Commenté [MB35]: Can we add a note where this is defined?

Figure 1: Comparison of message sizes in bytes with CCM_8, P-256, and ECDSA and with Connection ID

Figure 2 compares the message sizes of DTLS 1.3 [RFC9147] and TLS 1.3 [RFC8446] handshakes without ~~connection ID~~CID but with the same algorithms CCM_8, P-256, and ECDSA.

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-10 - X.509s by reference, ECDHE	107	200	98	405
cTLS-10 - PSK, ECDHE	108	120	20	250
cTLS-10 - PSK	43	57	20	120

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.

a mis en forme : Surlignage

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-10	- X.509s by reference, ECDHE	74	160	91	325
cTLS-10	- PSK, ECDHE	75	89	20	186
cTLS-10	- PSK	43	57	20	120

Figure 3: Comparison of message sizes in bytes with CCM_8, x25519, and ed25519 or PSK and without Connection ID

The numbers in Figures 1, ~~Figure~~ 2, and ~~Figure~~ 3 were calculated with ~~8~~ 8-byte~~s~~ tags, consistent with the algorithms that are ~~mandatory~~ mandatory-to-implement ~~as~~ per [I-D.ietf-uta-tls13-iot-profile] and Section 8 of [RFC9528]. If ~~16~~ 16-byte~~s~~ ~~tag~~ tags are used, the numbers in the flights #2 and #3 columns increase by 8 bytes and the numbers in the Total column increase by 16 bytes.

The numbers in Figures 1, ~~Figure~~ 2, and ~~Figure~~ 3 do not consider underlying layers~~, see~~ per the assumptions in Section 2.

3.2. DTLS 1.3

This section gives an estimate of the message sizes of DTLS 1.3 with different authentication methods.

~~Note that t~~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.

Commenté [MB36]: Should this be moved to Terminology/Notations section?

3.2.1. Message Sizes RPK + ECDHE

In this section, CCM_8, P-256, and ECDSA and a Connection ID of 1 byte are used.

3.2.1.1. Flight #1

Commenté [MB37]: For convenience, may be refer to rfc9147.html#hdr_examples for an overview of the messages and their size?

```
Record Header - DTLSPlaintext (13 bytes):
16 fe fd EE EE SS 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~~1 byte):
00

Legacy Cookie (~~1 bytes~~1 byte):

```
      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~~.~~:

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

With efficiently encoded key share such as x25519 or [I-D.mattsson-tls-compact-ecc], the overhead is 185 - 33 = 152 bytes.

Commenté [MB38]: Do we need to cite this one?

3.2.1.2. Flight #2

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

     Handshake Header - Server Hello (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 bytes1 byte):
    00

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

    Compression Method (null) (1 bytes1 byte):
    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 bytes1 byte):
    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 bytes1 byte):
      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 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 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~~.~~:

13 + 137 + 3 + 26 + 23 + 112 + 87 + 44 + 1 + 8 = 454 bytes

With a point compressed secp256r1 RPK, the overhead is 454 - 32 = 422 bytes, see Section 3.2.7.

With an ed25519 RPK and signature, the overhead is 454 - 47 - 7 = 400 bytes.

With an ~~efficiently~~ efficiently-encoded key share such as x25519 or [I-D.mattsson-tls-compact-ecc], the overhead is 454

Commenté [MB39]: There is no such section. I guess you meant 3.2.7?

Please check.

Commenté [MB40]: Can this be explained?

Commenté [MB41]: May be add in parenthesize the meaning of each length.

The comment applies for other similar uses in the document.

This helps readers walk through. Thanks.

Commenté [MB42]: Do we need to cite this one?

- 33 = 421 bytes.

With an ~~efficiently~~ 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.

Commenté [MB43]: Maybe add a comment about this figure

Commenté [MB44]: Idem as previous comments.

3.2.1.3. Flight #3

Record Header (3 bytes): // DTLSCiphertext
ZZ 43 SS

Handshake Header - Certificate (12 bytes):
0b LL LL LL SS SS XX XX XX LL LL LL

Request Context (~~1 bytes~~1 byte):
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~~ DER-encoded RPK, See Section 2.2.7.

Commenté [MB45]: Maybe add to the notations section the meaning

Commenté [MB46]: There is no such section

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~~ 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~~.~~:

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

With a

point compressed secp256r1 RPK, the overhead is 255 - 32 = 223 bytes, see Section 3.2.7.

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.

3.2.2. Message Sizes PSK + ECDHE

3.2.2.1. Flight #1

The differences in overhead compared to Section 3.2.1.1 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 extensions ~~is~~ are 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~~.~~:

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

3.2.2.2. Flight #2

The differences in overhead compared to Section 3.2.1.2 are:

The following is added:

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

The following ~~is~~ are 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 bytes~~

DTLS 1.3 PSK + ECDHE flight #2 gives 226 bytes of overhead~~.~~:

454 + 6 - 112 - 87 - 23 - 6 - 6 = 226 bytes

3.2.2.3. Flight #3

The differences in overhead compared to Section 3.2.1.3 are:

The following ~~is~~ re removed:

- Handshake Message Certificate (112 bytes)

- Handshake Message Certificate Verify (87 bytes)

~~In total:~~

~~255 - 112 - 87 = 56 bytes~~

DTLS 1.3 PSK + ECDHE flight #3 gives 56 bytes of overhead~~.~~:

255 - 112 - 87 = 56 bytes

3.2.3. Message Sizes PSK

3.2.3.1. Flight #1

The differences in overhead compared to Section 3.2.2.1 are:

The following ~~is~~ are removed:

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

- Extension - Key Share (75 bytes)

~~In total:~~

~~219 - 8 - 75 = 136 bytes~~

DTLS 1.3 PSK flight #1 gives 136 bytes of overhead~~.~~:

219 - 8 - 75 = 136 bytes

3.2.3.2. Flight #2

The differences in overhead compared to Section 3.2.2.2 are:

The following is removed:

- Extension - Key Share (73 bytes)

~~In total:~~

~~226 - 73 = 153 bytes~~

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

226 - 73 = 153 bytes

3.2.3.3. Flight #3

There are no differences in overhead compared to Section 3.2.2.3.

DTLS 1.3 PSK flight #3 gives 56 bytes of overhead.

3.2.4. Cached Information

~~In t~~This section~~, we~~ considers the effect of [RFC7924] on the message size overhead.

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

The differences compared to Section 3.2.1 are ~~the following~~detailed in the following subsections.

3.2.4.1. Flight #1

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
```

Giving a total of:

185 + 39 = 224 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

3.2.4.2. Flight #2

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 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

3.2.5. Resumption

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

```
Record Header - DTLSCiphertext (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.

3.2.6. 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
```

3.2.7. Raw Public Keys (RPKs)

~~Raw Public Keys~~RPKs in TLS consist of a ~~DER~~ DER-encoded ASN.1 SubjectPublicKeyInfo structure [RFC7250]. 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~~ defined in Section 2.2 [RFC5480] and is, therefore, theoretically possible to use in ~~PRKs~~ RPKs and X.509 certificates used in (D)TLS but does not seem to be supported by (D)TLS implementations.

Commenté [MB47]: Please check

Commenté [MB48]: Do we need this mention?

3.2.7.1. 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
```

Commenté [MB49]: RFC5480

Commenté [MB50]: Where the OID is defined?

3.2.7.2. 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
```

3.2.7.3. 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
```

3.3. 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~~ are smaller, and that Connection ID is not supported. ~~Recently, a~~Additional work has taken shape with the goal to further reduce overhead for TLS 1.3 (~~see~~ e.g., [I-D.ietf-tls-ctls]).

3.3.1. Message Sizes RPK + ECDHE

~~In this section,~~ CCM_8, x25519, and ed25519 are used in this section.

3.3.1.1. 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~~1 byte):
    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~~.~~:

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

3.3.1.2. 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~~1 byte):
       00

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

       Compression Method (null) (~~1 bytes~~1 byte):
       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 bytes1 byte):
    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 bytes1 byte):
    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 DER-encoded RPK, see Section 2.2.7.

    Certificate Extensions (2 bytes):
    00 00

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

Commenté [MB51]: There is no such section

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 bytes~~

TLS 1.3 RPK + ECDHE flight #2 gives 307 bytes of overhead~~.~~:

5 + 90 + 5 + 18 + 15 + 57 + 72 + 36 + 1 + 8 = 307 bytes

3.3.1.3. 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~~1 byte):
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~~ DER-encoded RPK, see Section 2.2.7.

Commenté [MB52]: There is no such section

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~~.~~:

5 + 57 + 72 + 36 + 1 + 8 = 179 bytes

3.3.2. Message Sizes PSK + ECDHE

3.3.2.1. Flight #1

The differences in overhead compared to Section 3.3.1.3 are:

The following ~~is~~ are 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~~ are 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~~.~~:

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

3.3.2.2. Flight #2

The differences in overhead compared to Section 3.3.1.2 are:

The following is added:

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

The following ~~is~~ are 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~~.~~:

307 - 57 - 72 - 15 - 6 - 6 + 6 = 157 bytes

3.3.2.3. Flight #3

The differences in overhead compared to Section 3.3.1.3 are:

The following ~~is~~ are 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~~.~~:

179 - 57 - 72 = 50 bytes

3.3.3. Message Sizes PSK

3.3.3.1. Flight #1

The differences in overhead compared to Section 3.3.2.1 are:

The following ~~is~~ are 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:

163 - 8 - 42 = 113 bytes~~.~~

3.3.3.2. Flight #2

The differences in overhead compared to Section 3.3.2.2 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~~.~~: 157 - 40 = 117 bytes.

3.3.3.3. Flight #3

There are no differences in overhead compared to Section 3.3.2.3.

TLS 1.3 PSK flight #3 gives 50 bytes of overhead.

3.4. 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 of 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~~ The number of bytes for TLS 1.2 in the four flights are 170, 1188, 117, and 75 for a total of 1550 bytes. ~~In TLS 1.3 t~~The number of bytes for TLS 1.3 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.

3.5. cTLS

Version -10 of the cTLS specification [I-D.ietf-tls-ctls] has a single example with CCM_8, x25519, and ed25519 in Appendix A. This document uses that example and calculates numbers for different parameters as follows:

Using secp256r1 instead of x25519 adds 33 bytes to the KeyShareEntry.key_exchange in flight #1 and flight #2.

Using ecdsa_secp256r1_sha256 instead of ed25519 adds an average of 7 bytes to CertificateVerify.signature in flight #2 and flight #3.

Using PSK authentication instead of ed25519 adds 1 byte (psk identifier) to flight #1 and removes 71 bytes (certificate and certificate_verify) from flights ~~flight~~ #2 and #3.

Using PSK key exchange x25519 removes 32 bytes (KeyShareEntry.key_exchange) from flights~~flight~~ #1 and #2.

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

3.6. EDHOC

This section gives an estimate of the message sizes of EDHOC [RFC9528] authenticated with static Diffie-Hellman keys and where the static Diffie-Hellman keys 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 [RFC9529].

Commenté [MB53]: Add a reference

3.6.1. Message Sizes RPK

Commenté [MB54]: Nit: use consistent hex: some with upper/lower case.

I have a preference for upper case as this is eases checking against the output provided for all the three messages when using https://cbor.me/ to validate the encoding.

Same comment apples for all other hex representations.

3.6.1.1. 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
```

3.6.1.2. 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
```

3.6.1.3. 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
```

3.6.2. 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~~ flights #2 and #3 by 57 (64 - 8 + 1) or 58 bytes, while x5t increases the size by 13-14 bytes compared to kid.

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 [RFC9529].

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

Figure 4: Typical EDHOC message sizes in bytes

3.7. Summary

~~To do a~~Performing a fair comparison~~, one has to~~ requires ~~-~~choos~~e~~ing 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 due to 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.

4. Overhead for Protection of Application Data

To enable comparison, all the overhead calculations in this section use an ~~8~~ 8-~~bytes~~ byte ICV (e.g., AES_128_CCM_8 [RFC6655] or AES-CCM-16-64-128 [RFC9053]) or 16 bytes (e.g., AES-CCM [SP-800-38C], AES-GCM [SP-800-38D], or ChaCha20-Poly1305 [RFC7539]), a plaintext of 6 bytes, and the sequence number '05'. This follows the example in Figure 16 of [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 also that the OSCORE overhead calculations are dependent on the CoAP O~~o~~ption 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 4.1 gives a short summary of the message overhead based on different parameters and some assumptions. The following sections detail the assumptions and the calculations.

4.1. Summary

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

The compression overhead (GHC) is dependent on the ~~parameters~~ epoch, sequence number, Connection ID, and length (where applicable) parameters. 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 Sender Sequence Number~~sequence number~~ parameters.

The following overheads apply for all sequence numbers and Sender IDs with the same length, and for an ID Context of zero-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	11	11

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	6	
Group OSCORE pairwise response	3	

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

The numbers in Figures 5, ~~Figure ~~6, and ~~Figure ~~7 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 mechanism in [I-D.ietf-tls-super-jumbo-record-limit] reduces the overhead of uncompressed TLS 1.3 records by 3 bytes.

The numbers in Figures 5 and ~~Figure ~~6 were calculated with 8 bytes ICV which is the ~~mandatory ~~mandatory-~~to ~~to-implement in [I-D.ietf-uta-tls13-iot-profile]~~, and~~and Section 8 of [RFC9528]. If
16-

bytes tag are used, all numbers increases by 8.

The numbers in Figures 5, ~~Figure~~ 6, and ~~Figure~~ 7 do not consider underlying layers, see Section 2.

4.2. DTLS 1.2

4.2.1. DTLS 1.2 (Basic)

This section analyzes the overhead of DTLS 1.2 [RFC6347]. The nonce follows 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.

4.2.2. 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.

4.2.3. 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 use 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.

4.2.4. 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.

4.3. DTLS 1.3

4.3.1. DTLS 1.3 (Basic)

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.

4.3.2. 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.

4.3.3. 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.

4.3.4. 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.

4.4. TLS 1.2

4.4.1. TLS 1.2 (Basic)

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.

4.4.2. 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.

4.5. TLS 1.3

4.5.1. TLS 1.3 (Basic)

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.

4.5.2. 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.

4.6. 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 and payload marker):
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 ~~Sequence ~~number~~Number, and Sender
ID):
   09 05 42
   Payload marker:
   ff
   Ciphertext (including encrypted code and payload marker):
   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.

4.7. 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. An additional requirement compared to [RFC8613] is that ID Context is always included in requests. For example, if the length of the ID Context is 1 byte and the length of the Sender ID is 1 byte, this adds 2 bytes to requests.

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 19 05 00 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 nr, ID Context length, Sender ID):
19 05 00 42
Payload marker:
ff
Ciphertext (including encrypted code and payload marker):
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 11 bytes overhead for responses.

4.8. Summary

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 s~~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 it can be seen in Figure 7, Group OSCORE for pairwise communication increases the overhead of OSCORE requests by 20%.

Note that the compared protocols have slightly different use cases. TLS and DTLS are designed for the transport layer and are terminated at 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.

X. Operational Considerations

Commenté [MB55]: Are there relevant consideration we should call out here?

For example, the overhead has implications that may lead to fragmentations, etc.

5. Security Considerations

When using the security protocols outlined in this document, it is important to adhere to the latest requirements and recommendations for the respective protocol. It is also crucial to utilize supported versions of libraries that continue to receive security updates in response to identified vulnerabilities.

While the security considerations provided in DTLS 1.2 [RFC6347], DTLS 1.3 [RFC9147], TLS 1.2 [RFC5246], TLS 1.3 [RFC8446], cTLS [I-D.ietf-tls-ctls], EDHOC [RFC9528] [RFC9668], OSCORE [RFC8613], Group OSCORE [I-D.ietf-core-oscore-groupcomm], and X.509 [RFC5280] serve as a good starting point, they are not sufficient due to the fact that some of these specifications were authored many years ago. For instance, being compliant to the TLS 1.2 [RFC5246] specification is considered very poor security practice, given that the mandatory-to-implement cipher suite TLS_RSA_WITH_AES_128_CBC_SHA possesses at least three major weaknesses.

Therefore, implementations and configurations must also align with the latest recommendations and best practices. Notable examples when this document was published include [BCP 195] ~~[RFC9325] [RFC8996]~~, [SP-800-52], and [BSI-TLS].

Commenté [MB56]: Cite directly the BCP

6. IANA Considerations

This document has no actions for IANA.

7. Informative References

Commenté [MB57]: Please update to reflect the comments raised above.

[BSI-TLS] Bundesamt für Sicherheit in der Informationstechnik, "Technical Guideline TR-02102-2 Cryptographic Mechanisms: Recommendations and Key Lengths Part 2 - Use of Transport Layer Security (TLS)", January 2025, <https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/TechGuidelines/TG02102/BSI-TR-02102-2.pdf>.

[FIPS-180-4] NIST, "Secure Hash Standard (SHS)", August 2015, <https://doi.org/10.6028/NIST.FIPS.180-4>.

[FIPS-186-5] NIST, "Digital Signature Standard (DSS)", February 2023, <https://doi.org/10.6028/NIST.FIPS.186-5>.

[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-05, 22 December 2024, <https://datatracker.ietf.org/doc/html/draft-ietf-core-attacks-on-coap-05>.

[I-D.ietf-core-oscore-groupcomm] Tiloca, M., Selander, G., Palombini, F., Mattsson, J. P., and R. Höglund, "Group Object Security for Constrained RESTful Environments (Group OSCORE)", Work in Progress, Internet-Draft, draft-ietf-core-oscore-groupcomm-25, 16 March 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-core-oscore-groupcomm-25>.

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

Mattsson, J. P., Selander, G., Raza, S., Höglund, J., and M. Furuhed, "CBOR Encoded X.509 Certificates (C509 Certificates)", Work in Progress, Internet-Draft, draft-ietf-cose-cbor-encoded-cert-13, 3 March 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-cose-cbor-encoded-cert-13>.

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

[I-D.ietf-schc-8824-update]
Tiloca, M., Toutain, L., Martinez, I., and A. Minaburo, "Static Context Header Compression (SCHC) for the Constrained Application Protocol (CoAP)", Work in Progress, Internet-Draft, draft-ietf-schc-8824-update-04, 3 March 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-schc-8824-update-04>.

[I-D.ietf-tls-cert-abridge]
Jackson, D., "Abridged Compression for WebPKI Certificates", Work in Progress, Internet-Draft, draft-ietf-tls-cert-abridge-02, 16 September 2024, <https://datatracker.ietf.org/doc/html/draft-ietf-tls-cert-abridge-02>.

[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-10, 17 April 2024, <https://datatracker.ietf.org/doc/html/draft-ietf-tls-ctls-10>.

[I-D.ietf-tls-super-jumbo-record-limit]
Mattsson, J. P., Tschofenig, H., and M. Tüxen, "Large Record Sizes for TLS and DTLS with Reduced Overhead", Work in Progress, Internet-Draft, draft-ietf-tls-super-jumbo-record-limit-01, 28 May 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-tls-super-jumbo-record-limit-01>.

[I-D.ietf-uta-tls13-iot-profile]
Tschofenig, H., Fossati, T., and M. Richardson, "TLS/DTLS 1.3 Profiles for the Internet of Things", Work in Progress, Internet-Draft, draft-ietf-uta-tls13-iot-profile-14, 5 May 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-uta-tls13-iot-profile-14>.

[I-D.kampanakis-tls-scas-latest]
Kampanakis, P., Bytheway, C., Westerbaan, B., and M. Thomson, "Suppressing CA Certificates in TLS 1.3", Work in Progress, Internet-Draft, draft-kampanakis-tls-scas-latest-03, 5 January 2023, <https://datatracker.ietf.org/doc/html/draft-kampanakis-

tls-scas-latest-03>.

[I-D.mattsson-tls-compact-ecc]
Mattsson, J. P. and H. Tschofenig, "Compact ECDHE and ECDSA Encodings for TLS 1.3", Work in Progress, Internet-Draft, draft-mattsson-tls-compact-ecc-06, 23 February 2024, <https://datatracker.ietf.org/doc/html/draft-mattsson-tls-compact-ecc-06>.

[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>.

[Performance]
Fedrecheski, G., Vučinić, M., and T. Watteyne, "Performance Comparison of EDHOC and DTLS 1.3 in Internet-of-Things Environments", January 2024, <https://hal.science/hal-04382397>.

[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>.

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

[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>.

[RFC6655] McGrew, D. and D. Bailey, "AES-CCM Cipher Suites for Transport Layer Security (TLS)", RFC 6655, DOI 10.17487/RFC6655, July 2012,

<https://www.rfc-editor.org/info/rfc6655>.

[RFC7250] Wouters, P., Ed., Tschofenig, H., Ed., Gilmore, J., Weiler, S., and T. Kivinen, "Using Raw Public Keys in Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", RFC 7250, DOI 10.17487/RFC7250, June 2014, <https://www.rfc-editor.org/info/rfc7250>.

[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>.

[RFC7539] Nir, Y. and A. Langley, "ChaCha20 and Poly1305 for IETF Protocols", RFC 7539, DOI 10.17487/RFC7539, May 2015, <https://www.rfc-editor.org/info/rfc7539>.

[RFC7748] Langley, A., Hamburg, M., and S. Turner, "Elliptic Curves for Security", RFC 7748, DOI 10.17487/RFC7748, January 2016, <https://www.rfc-editor.org/info/rfc7748>.

[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>.

[RFC8032] Josefsson, S. and I. Liusvaara, "Edwards-Curve Digital Signature Algorithm (EdDSA)", RFC 8032, DOI 10.17487/RFC8032, January 2017, <https://www.rfc-editor.org/info/rfc8032>.

[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>.

[RFC8824] Minaburo, A., Toutain, L., and R. Andreasen, "Static Context Header Compression (SCHC) for the Constrained Application Protocol (CoAP)", RFC 8824, DOI 10.17487/RFC8824, June 2021, <https://www.rfc-editor.org/info/rfc8824>.

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

[RFC8996] Moriarty, K. and S. Farrell, "Deprecating TLS 1.0 and TLS 1.1", BCP 195, RFC 8996, DOI 10.17487/RFC8996, March 2021, <https://www.rfc-editor.org/info/rfc8996>.

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

[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>.

[RFC9325] Sheffer, Y., Saint-Andre, P., and T. Fossati, "Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", BCP 195, RFC 9325, DOI 10.17487/RFC9325, November 2022, <https://www.rfc-editor.org/info/rfc9325>.

[RFC9528] Selander, G., Preuß Mattsson, J., and F. Palombini, "Ephemeral Diffie-Hellman Over COSE (EDHOC)", RFC 9528, DOI 10.17487/RFC9528, March 2024, <https://www.rfc-editor.org/info/rfc9528>.

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

[RFC9547] Arkko, J., Perkins, C. S., and S. Krishnan, "Report from the IAB Workshop on Environmental Impact of Internet Applications and Systems, 2022", RFC 9547, DOI 10.17487/RFC9547, February 2024, <https://www.rfc-editor.org/info/rfc9547>.

[RFC9668] Palombini, F., Tiloca, M., Höglund, R., Hristozov, S., and G. Selander, "Using Ephemeral Diffie-Hellman Over COSE (EDHOC) with the Constrained Application Protocol (CoAP) and Object Security for Constrained RESTful Environments (OSCORE)", RFC 9668, DOI 10.17487/RFC9668, November 2024, <https://www.rfc-editor.org/info/rfc9668>.

[SCHC-eval] Dumay, M., Barthel, D., Toutain, L., and J. Lecoeuvre, "Effective interoperability and security support for constrained IoT networks", December 2021, <https://ieeexplore.ieee.org/document/9685592>.

[SP-800-186] Chen, L., Moody, D., Randall, K., Regenscheid, A., and A. Robinson, "Recommendations for Discrete Logarithm-based Cryptography: Elliptic Curve Domain Parameters", NIST Special Publication 800-186, February 2023, <https://doi.org/10.6028/NIST.SP.800-186>.

[SP-800-38C] Dworkin, M., "Recommendation for Block Cipher Modes of Operation: the CCM Mode for Authentication and Confidentiality", NIST Special Publication 800-38C, May 2004, <https://doi.org/10.6028/NIST.SP.800-38C>.

[SP-800-38D] Dworkin, M., "Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC", NIST Special Publication 800-38D, November 2007, <https://doi.org/10.6028/NIST.SP.800-38D>.

[SP-800-52] McKay, K. and D. Cooper, "Guidelines for the Selection, Configuration, and Use of Transport Layer Security (TLS) Implementations", NIST Special Publication 800-52 Revision 2, August 2019, <https://doi.org/10.6028/NIST.SP.800-52r2>.

Appendix A. EDHOC Over CoAP and OSCORE

The overhead of CoAP and OSCORE when used to transport EDHOC is~~ a bit~~ more complex than the overhead of ~~UPD ~~UDP and TCP. Assuming that the CoAP Token has a length of 0 byte~~s~~, that the CoAP Content-Format is not used, that the EDHOC Initiator is the CoAP client, that the connection identifiers have 1-byte encodings, and that the CoAP URI path is "edhoc", the additional overhead due to CoAP being used as transport is:

For EDHOC message_1

Commenté [MB58]: Point to 3.6.1.1?

```
--- CoAP header: 4 bytes
--- CoAP token: 0 bytes
--- URI-Path option with value "edhoc": 6 bytes
--- Payload marker 0xff: 1 byte
--- Dummy connection identifier "true": 1 byte

Total: 12 bytes
```

For EDHOC message_2

Commenté [MB59]: Point to 3.6.1.2?

```
--- CoAP header: 4 bytes
--- CoAP token: 0 bytes
--- Payload marker 0xff: 1 byte

Total: 5 bytes
```

For EDHOC message_3 without the combined request

Commenté [MB60]: Point to the section where this is defined

```
--- CoAP header: 4 bytes
--- CoAP token: 0 bytes
--- URI-Path option with value "edhoc": 6 bytes
--- Payload marker 0xff: 1 byte
--- Connection identifier C_R (wire encoding): 1 byte

Total: 12 bytes
```

For EDHOC message_3 over OSCORE with the EDHOC + OSCORE combined request [RFC9668] all the overhead contributions from the previous case is gone. The only additional overhead is 1 byte due to the EDHOC CoAP option.

Acknowledgments

The authors want to thank Carsten Bormann, Thomas Fossati, Russ Housley, Ari Keränen, Erik Kline, Stephan Koch, Achim Kraus, Michael Richardsson, Göran Selander, Bill Silverajan, Akram Sheriff, Marco Tiloca, 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
Email: john.mattsson@ericsson.com

Francesca Palombini
Ericsson
Email: francesca.palombini@ericsson.com

Mališa Vučinić
INRIA

Email: malisa.vucinic@inria.fr