

DICE Working Group
Internet-Draft
Intended status: Informational
Expires: April 10, 2014

K. Hartke
Universitaet Bremen TZI
October 7, 2013

Practical Issues with
Datagram Transport Layer Security in Constrained Environments
draft-hartke-dice-practical-issues-00

Abstract

This document investigates practical issues around the implementation of Datagram Transport Layer Security (DTLS) in constrained environments, and explores some ideas for an optimized version of DTLS that is more friendly to constrained nodes and networks.

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 <http://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 April 10, 2014.

Copyright Notice

Copyright (c) 2013 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 (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
1.1. Background	3
1.2. Overview	3
1.3. Terminology	4
2. Potential Problems and Possible Solutions	4
2.1. Handshake Reliability and Fragmentation	4
2.2. Timer Values	7
2.3. Connection Initiation	8
2.4. Connection Closure	9
2.5. Data Size	10
2.6. Code Size	10
2.7. Application Data Fragmentation	11
2.8. Applications	12
3. A Comparison of Strategies for Handshake Reliability	13
4. A Strawman for Stateless Header Compression	16
4.1. Records	16
4.2. Handshake Messages	17
5. Security Considerations	19
6. IANA Considerations	19
7. Acknowledgements	19
8. References	19
8.1. Normative References	19
8.2. Informative References	19
Appendix A. Templates	21
Author's Address	22

1. Introduction

1.1. Background

Nodes taking part in the "Internet of Things" often have strict limitations regarding their computational power, memory size (both RAM and ROM) and power management [I-D.ietf-lwig-guidance]. Network communication, in particular if wireless, also imposes constraints that need to be considered during protocol design, such as low bitrates, variable delays and and possibly high packet loss.

Moreover, frames at the link layer might be much smaller than the IPv6 minimum MTU of 1280 bytes and therefore require additional adaptation mechanisms such as 6LoWPAN [RFC4944] for IEEE 802.15.4 wireless networks [IEEE.802-15-4], which in turn may exacerbate the limitations of the network: for instance, as high loss rates are anticipated by design, application protocols usually try to avoid fragmentation at the network layer.

However, application protocols often delegate security mechanisms to transport layer security protocols. More often than not, the protocol overhead from securing the communication is highly relevant to the overall performance of the systems.

One protocol that has received significant attention recently for constrained node/network applications is Datagram Transport Layer Security (DTLS) [RFC6347]. DTLS is derived from and inherits some characteristics from TLS [RFC5246]. Although it has clearly not been designed with constrained devices and lossy networks in mind, it is thought to be usable in these environments [I-D.gilger-smart-object-security-workshop]. There are still a few challenges when it comes to actually implement DTLS.

1.2. Overview

The present document investigates practical issues around the implementation of DTLS in constrained environments, and explores a few ideas that could lead to an optimized version of DTLS that is more friendly to constrained nodes and networks.

The ideas generally fall into one of the following categories:

Implementation guidance: Implementation techniques for achieving light-weight implementations of DTLS, without affecting conformance to the relevant specifications or interoperability with other implementations. This includes techniques for reducing complexity, memory footprint, or power usage. The result may eventually be incorporated into [I-D.ietf-lwig-guidance].

Protocol profile: Use of DTLS in a particular way, for example, by changing certain "MAY"s into "MUST"s or "MUST NOT"s, or by prescribing or precluding certain extensions and cipher suites. Existing DTLS implementations ought to be usable without change if they can be configured accordingly.

Stateless header compression: Compression of DTLS records without explicitly building any compression context state. This is done by using shorter forms to represent the same bits of information or relying on information that is already shared by the client and server. Existing DTLS implementations can continue to be used if a thin layer is added that handles compression and decompression.

Breaking changes: New implementations are required that do not interoperate with implementations of DTLS, although the overall operation of Transport Layer Security is not changed.

1.3. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119]. Note that this document itself is informational, but it is discussing normative statements.

2. Potential Problems and Possible Solutions

2.1. Handshake Reliability and Fragmentation

DTLS records can be large in size for a single 6LoWPAN [RFC4944] payload: IEEE 802.15.4 [IEEE.802-15-4] specifies a physical layer MTU of only 127 bytes, which yields about 60-80 bytes of payload after adding MAC layer and adaptation layer headers. Although 6LoWPAN supports the fragmentation of IPv6 packets into small link-layer frames, this is generally tried to be avoided in low-power, lossy networks.

DTLS offers fragmentation at the handshake layer and hence can help to prevent IP fragmentation. However, this can add a significant overhead on the number of datagrams and bytes transferred (see Table 1 below). Packet loss is also still a big problem for the constrained nodes: since fragments may arrive in any order, buffers must be large enough to hold all messages after reassembly, and losing a single fragment will cause all fragments of a message flight to be retransmitted. This is very likely especially during key and certificate exchanges as these will not fit within a packet without fragmentation in most 6LoWPANs.

UDP data size limit (bytes)	Number of datagrams transferred	Total number of bytes transferred	Proportion of header data
50	27	1,182	55 %
55	21	1,037	49 %
60	20	1,081	51 %
65	18	1,003	47 %
70	15	912	42 %
75	14	875	39 %
80	13	874	39 %
85	12	849	37 %
90	12	849	37 %
1,152	6	802	34 %

Table 1: Number of datagrams and bytes transferred using different limits for DTLS fragmentation in an example DTLS handshake (TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8 with Raw Public Key Certificate)

Possible Solutions include:

- o Perform the handshake using alternative mechanisms for reliability over UDP:
 - * Use IP fragmentation. If no X.509 certificates are involved, the handshake messages of one flight typically require less than 400 bytes combined. Since all messages of a flight in DTLS are retransmitted anyway when a single fragment is lost, the difference between performing the fragmentation at the DTLS layer and at the IP layer is probably not huge.
 - * Use DTLS fragmentation. When compared to, for example, the reliability mechanism of CoAP over UDP [I-D.ietf-core-coap] (where the receipt of each data fragment is confirmed by one acknowledgement message, and an acknowledgement message may opportunistically piggyback data in the opposite direction), DTLS actually performs better for a typical DTLS handshake in both lossy and non-lossy network environments (cf. Section 3).
 - * Extend DTLS with acknowledgment messages that confirm the receipt of fragments and allow an implementation to retransmit only the fragments that are missing. Section 3 explores a number of strategies for the reliable transmission of DTLS handshake messages with acknowledgements, including CoAP-style acknowledgements and cumulative acknowledgements.

UDP data size limit (bytes)	Number of datagrams transferred	Total number of bytes transferred	Proportion of header data
50	15 (56 %)	592 (50 %)	10 %
55	13 (62 %)	585 (56 %)	9 %
60	13 (65 %)	621 (57 %)	14 %
65	11 (61 %)	588 (59 %)	10 %
70	11 (73 %)	573 (63 %)	7 %
75	11 (79 %)	573 (65 %)	7 %
80	10 (77 %)	567 (65 %)	6 %
85	10 (83 %)	567 (67 %)	6 %
90	10 (83 %)	567 (67 %)	6 %
1,152	6 (100 %)	617 (77 %)	14 %

Table 2: Number of datagrams and bytes transferred in the same example DTLS handshake as in Table 1 but using the strawman for Stateless Header Compression described in Section 4

- o Reduce the number of bytes to be transferred, so fewer packets need to be transmitted that could potentially be lost:
 - * Exchange large blobs using an out-of-band mechanism. The TLS Cached Information Extension [I-D.ietf-tls-cached-info], for example, allows to omit the exchange of fairly static data such as the server certificate, if this data is already available.
 - * Compress DTLS messages with 6LoWPAN General Header Compression [I-D.bormann-6lowpan-ghc], as proposed in [DCOSS12].
 - * Perform a DTLS-specific kind of Stateless Header Compression, as explored in Section 4. This can significantly reduce the number of datagrams and bytes transferred, and in particular also the proportion of header data within the number of bytes transferred (see Table 2 above).
 - * Mandate the use compressed point formats for elliptic curve points.
 - * Recover the Raw Public Key Certificate [I-D.ietf-tls-oob-pubkey] from the ECDSA signature in a ECDHE_ECDSA handshake instead of transmitting both the public key and the signature, as described in Section 4.1.6 of [SEC1].

"This is also useful in bandwidth constrained environments, when transmission of public keys cannot be afforded. Entity

U could send a signature to entity V, who recovers QU. Entity V can look up the public key in some certificate or directory, and if it matches then the signature can be accepted." [SEC1]

- * Transmit only the low-order N bits of the 48 bit sequence numbers and reconstruct the (48-N) high-order bits, as similarly done for sequence numbers in IPsec (see Appendix B of RFC 4302 [RFC4302]).
- * Use self-delimiting numeric values [RFC6256] instead of fixed-sized fields.
- * Use a single bit field instead of multiple type fields to indicate which handshake messages are present in a record.

2.2. Timer Values

RFC 6347 [RFC6347] leaves the choice of timer values to the implementation, but makes the following recommendation:

"Implementations SHOULD use an initial timer value of 1 second (the minimum defined in RFC 6298 [RFC6298]) and double the value at each retransmission, up to no less than the RFC 6298 maximum of 60 seconds." [RFC6347]

Given the time required by some algorithms when executed on a constrained devices (see Table 3), an initial timer value of 1 second can easily lead to spurious retransmissions.

Algorithm	Library	Memory footprint (bytes)	Execution time (seconds)	Comparable RSA key length
RSA 1024	AvrCryptolib	640	199.7	
RSA 2048	AvrCryptolib	1,280	1,587.6	
ECDSA 160r1	TinyECC	892	2.3	1024
ECDSA 192r1	TinyECC	1,008	3.6	1536
ECDSA 160r1	Wiselib	842	20.2	1024
ECDSA 192r1	Wiselib	952	34.6	1536
ECDSA 163k1	Relic	2,804	0.3	1024
ECDSA 233k1	Relic	3,675	1.8	2048

Table 3: RSA private key operation and ECDSA signature performance (from [I-D.aks-crypto-sensors])

Possible Solutions include:

- o Adjust the timer value to meet the conditions of constrained nodes and low-power, lossy networks.
- o Add acknowledgment messages to DTLS that allow an implementation to confirm the receipt of a message before starting to prepare its response message flight; see Section 3.

2.3. Connection Initiation

Nodes with very constrained main memory also suffer from the complexity of the DTLS handshake protocol. We envision that the acceptance of DTLS as security protocol for embedded devices would significantly increase if a less complex connection initiation procedure with a smaller number of handshake messages was defined.

Compared to TLS, DTLS exacerbates the connection initiation: A DTLS handshake has an additional roundtrip that results from the addition of a stateless cookie exchange. This exchange is designed to prevent certain denial-of-service attacks: consumption of excessive server resources caused by the transmission of a series of handshake initiation requests, and use of the server as an amplifier by sending connection initiation messages with a forged source of the victim.

Possible Solutions include:

- o Create the DTLS connection before it is needed, so it doesn't take a long time to set it up when it's actually needed. This works if a server has to deal with a relatively small overall number of clients that wish to interact with the server. Care must be taken such that not all clients perform their handshake at the same time, as a handshake requires considerably more memory than keeping a connection open. (See also Section 2.4 below.)
- o Shorten the handshake to four flights. This may be possible without losing the denial-of-service roundtrip if the cipher suite permits that the server remains stateless after sending the ServerHello and if the flight fits in one datagram (see Figure 1).
- o As an alternative, client puzzles could be used as a mechanism for mitigating denial-of-service attacks, resulting in a four-flight exchange similar to the one in HIP DEX [I-D.moskowitz-hip-rg-dex]. The application of client puzzles to TLS has been shown [USENIX01]. However, a puzzle would be needed that ideally takes less effort for a constrained device and more effort for an unconstrained device.

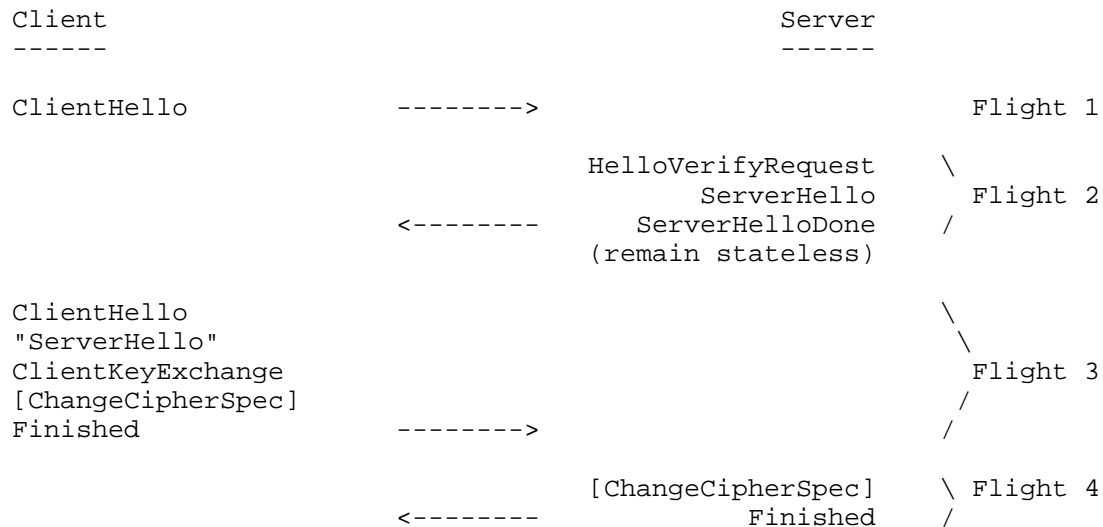


Figure 1: Artist's impression of a four-flight DTLS handshake with a Pre-Shared Key

2.4. Connection Closure

Although a connection needs considerably less memory after a handshake has finished, it still requires, for example, around 80 bytes with AES-128-CCM [RFC6655] for the keys, sequence numbers and anti-replay window. More memory is needed if session resumption is supported, to remember the 48-byte master secret and negotiated connection parameters. This limits how many connections a constrained device can maintain at a given time. Often, constrained devices will have a fixed number of "slots" for connections rather than allocating memory dynamically for each connection.

DTLS provides a facility for secure connection closure. When a valid closure alert is received, an implementation can be assured that no further data will be received on that connection. It is noteworthy, though, that the closure alert is not a handshake message and thus is not retransmitted when packet loss occurs.

Possible Solutions include:

- o Maintain the session for as long as possible. When the server runs out of resources, it can close connections, e.g., using a Least Frequently Used (LFU) eviction policy. The client simply assumes that the connection is active until the server rejects its application data, in which case the client initiates a new connection.

- o Use the DTLS Heartbeat Extension [RFC6520] to figure out from time to time if the connection is still active.

2.5. Data Size

As fragmented handshake messages can arrive at a constrained node in any order, the receiver must provide a message buffer that is large enough to hold multiple fragments. When several handshake messages forming a single flight are sent out in parallel, it is likely that the receiver's resources are too limited to order fragments from distinct handshake messages. Avoiding this might require additional resources on the server side to ensure serialization of a flight's messages.

Furthermore, since handshake messages can be fragmented arbitrarily and with overlaps, the receiver must, in addition to the message buffer, keep track of the fragments received so far. This also makes the computation of the Finished MAC difficult, which is computed as if each handshake message had been sent as a single fragment.

Possible retransmissions require even more buffer space as replay-protection requires encryption of every single packet that is to be transmitted. In particular, this renders destructive in-place encryption impossible as the source data must be preserved.

Possible Solutions include:

- o Use the same sequence number when retransmitting a message, so the plaintext could be encrypted in-place without the need for a second buffer. The security implications of this change need to be carefully analyzed.
- o Extend the exchange of handshake messages with acknowledgments that allow a receiver to confirm the receipt of fragments, and let the sender wait for the acknowledgment before it sends the next part of the flight; see Section 3.
- o Mandate non-overlapping handshake message fragments.
- o Favour cryptographic algorithms that use less memory, possibly resulting in a slower performance.

2.6. Code Size

Although probably not as severe as data size limits, the code size of a DTLS implementation also can play a role, in particular for constrained devices at the lower bound of Class 1 devices.

Possible Solutions include:

- o Use pre-composed messages instead of writing code for encoding or decoding ASN.1 structures, as shown for example in Appendix A.
- o Avoid static tables for cryptographic functions where possible, as typical embedded platforms are more restricted in RAM than in non-volatile memory such as flash ROM. Instead, their procedural equivalent is to be used, although less efficient during run-time.

2.7. Application Data Fragmentation

Messages larger than an IP fragment result in undesired packet fragmentation. DTLS does not support fragmentation of application data. If an implementation of an application layer protocol such as CoAP [I-D.ietf-core-coap] wants to avoid IP fragmentation, it must fit the application data (e.g., a CoAP message) and all headers in a single IP packet.

DTLS has a per-record overhead of 13 bytes for the record header. AEAD ciphers such as AES-CCM [RFC6655] eat up additional space to carry the explicit nonce and the authentication tag. Thus, cipher suites like TLS_PSK_WITH_AES_128_CCM_8 or TLS_ECDHE_ECDSA_AES_128_CCM_8 requires 16 additional bytes, leading to an overall overhead of 29 bytes for the header of each encrypted DTLS packet. With packet sizes of 60-80 bytes, this takes a considerable portion of the available packet size away (see Table 4 below).

UDP data size limit (bytes)	Number of bytes left for application data	... with Stateless Header Compression
50	21 (42 %)	39 (78 %)
55	26 (47 %)	44 (80 %)
60	31 (52 %)	49 (82 %)
65	36 (55 %)	54 (83 %)
70	41 (59 %)	59 (84 %)
75	46 (61 %)	64 (85 %)
80	51 (64 %)	69 (86 %)
85	56 (66 %)	74 (87 %)
90	61 (68 %)	79 (88 %)
1,152	1,123 (97 %)	1,141 (99 %)

Table 4: Number of bytes left for data in an ApplicationData record using DTLS and DTLS with Stateless Header Compression (Section 4)

Possible Solutions include:

- o Elide the GenericAEADCipher.nonce_explicit field when AES-CCM is used. The GenericAEADCipher.nonce_explicit field is set to the 16-bit epoch concatenated with the 48-bit sequence number, which means that the epoch and sequence number are unnecessarily included twice in each record.
- o Elide the DTLS version field where it is implicitly clear. Since the DTLS version is negotiated in the handshake, there should not be a need to specify the DTLS version in each and every record.
- o Elide the length field of the last record in a datagram. DTLS records specify their length, so multiple records can be transmitted in a single datagram. When DTLS is used with UDP (which preserves the boundaries of all message sent), the length field of the last record in a datagram can be calculated from the UDP payload length.

For example, when using the Stateless Header Compression presented in Section 4 and eliminating the redundant epoch and sequence number information, the number of bytes left in an ApplicationData record for application data can be significantly increased (see Table 4).

2.8. Applications

When DTLS is used to secure a non-trivial application, there is potential for synergies that can arise from optimizing the stack of both protocols.

For example, an implementation of CoAP [I-D.ietf-core-coap] with DTLS security will need to implement both the reliability mechanism for the DTLS handshake and the reliability mechanism of CoAP. This not only increases code size, but also prevents efficient retransmissions as each CoAP retransmission of the same data is a new transmission in DTLS.

Possible Solutions include:

- o Make DTLS reliability and fragmentation available to applications.

Accordingly, the application should take advantage of DTLS record information where possible. For example, since DTLS sequence numbers uniquely identify a message in a connection, the 6-byte sequence number could be used in CoAP to correlate CoAP acknowledgements with CoAP messages (Message ID, 2 bytes), to correlate CoAP responses with CoAP requests (Token, 0-8 bytes), to provide an order among CoAP notifications (3 bytes), and to enable message deduplication.

3. A Comparison of Strategies for Handshake Reliability

A DTLS handshake consists of multiple messages that are fragmented and grouped in so-called "flights". As the previous sections have shown, the strategy employed by DTLS to transmit these flights can lead to circumstances that are acceptable for existing uses of DTLS but pose a challenge in constrained environments:

- o The loss of a single packet causes the whole flight of fragments to be retransmitted, and not just the fragments that were lost.
- o Long processing times can lead to spurious retransmissions.
- o The possibility of arbitrarily reordered fragments requires the recipient to maintain potentially large buffers.

This section compares the following strategies for reliability:

Bulk without acknowledgements (Figure 2):

All fragments are retransmitted in exponentially increasing intervals until the first fragment of the next flight from the other side is received. This is the reliability mechanism used in DTLS 1.2 [RFC6347].

Stop-and-wait with one acknowledgement per fragment (Figure 3):

Each fragment is retransmitted individually until a matching acknowledgement for the fragment is received. Only one fragment is transmitted at a time, and each acknowledgement message confirms the receipt of one fragment. This is the reliability mechanism used in CoAP [I-D.ietf-core-coap].

Bulk with one cumulative acknowledgement per flight (Figure 4):

Transmit all unacknowledged fragments of the flight using a sliding window until all fragments have been acknowledged. Acknowledgements specify all fragments that have been received so far (highest sequence number seen + a bit field).

Table 5 shows the average number of transmissions needed for these three strategies to successfully complete an example DTLS handshake. (Every DTLS handshake is eventually successful if one side doesn't give up after a number of retransmission attempts.)

The results were obtained using a simple simulator that randomly drops packets according to the given loss rate, but otherwise provides ideal conditions. To avoid spurious retransmissions, timer values are selected larger than the processing times for flights; this may be impractical if sensible retransmission intervals and processing times differ in orders of magnitudes.

Loss rate	Figure 2	Figure 3	Figure 4
0.0%	18.0	36.0	19.0
5.0%	22.2	39.7	20.5
10.0%	25.9	41.8	23.8
15.0%	27.6	44.7	25.1
20.0%	33.3	51.6	27.1
25.0%	40.0	57.2	33.3
30.0%	39.2	64.0	37.4
35.0%	45.6	66.4	44.0
40.0%	55.4	74.7	46.2
45.0%	54.4	90.0	47.9
50.0%	67.2	102.2	57.2
55.0%	76.8	124.3	62.3
60.0%	96.9	151.3	74.4
65.0%	109.4	170.5	86.4
70.0%	115.8	248.2	106.8
75.0%	159.1	348.5	141.5
80.0%	199.6	528.6	169.9
85.0%	343.4	804.4	278.0

Table 5: Average number of transmissions for different strategies in an example ECDHE_ECDSA handshake with Raw Public Key Certificate

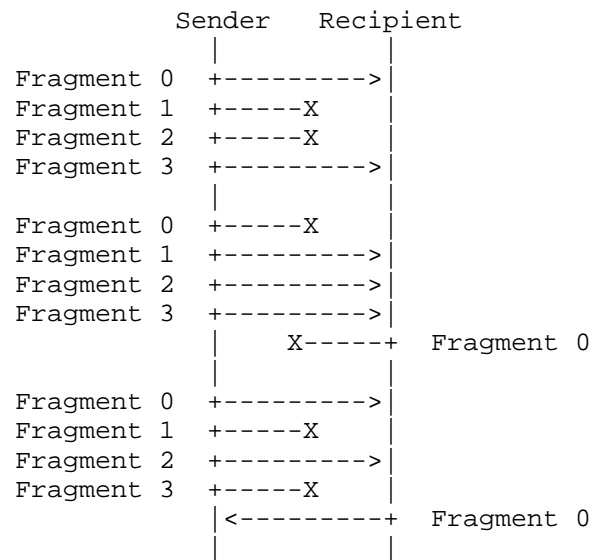


Figure 2: Bulk transmission without acknowledgements (DTLS)

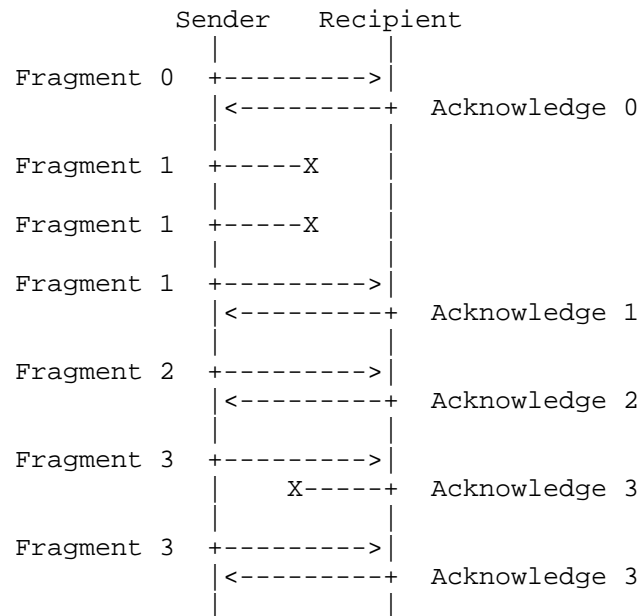


Figure 3: Stop-and-wait transmission with one acknowledgement per fragment

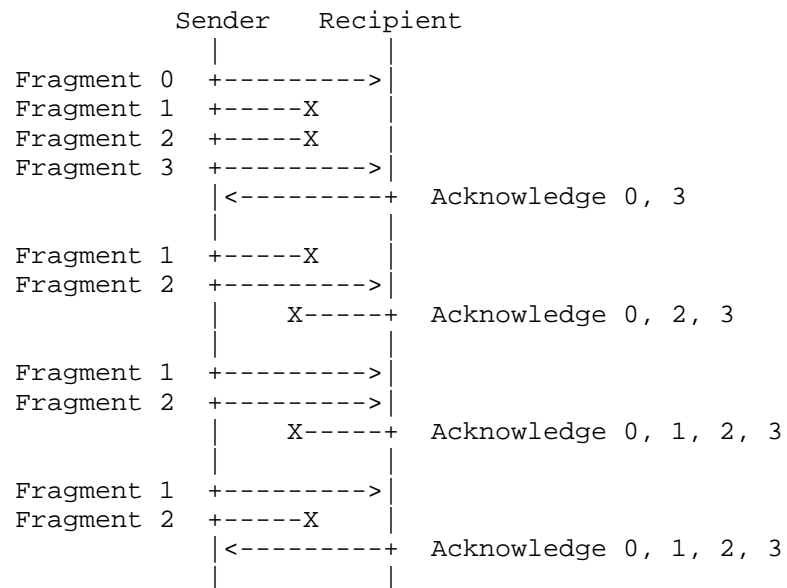


Figure 4: Bulk transmission with one acknowledgement per flight

4. A Strawman for Stateless Header Compression

Stateless Header Compression compresses the headers of DTLS 1.2 records and handshake messages. The compression is lossless, does not increase the record length and is done without explicitly building any compression context state.

The Finished MAC is computed as if each handshake message had been sent uncompressed.

4.1. Records

Records are compressed by specifying the type, version, epoch, sequence_number and length fields using a variable number of bytes. A prefix is added in front of the structure to indicate the length of each field or to specify the value of the field directly. If the value is specified directly, the field itself is elided. The format of the prefix is as follows:

```

          0                               1
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
+---+---+---+---+---+---+---+---+---+
|0| T | V | E | 1 1 0 | S | L |
+---+---+---+---+---+---+---+---+

```

The fields in the prefix are defined as follows:

T: Describes the type field.

- 0 - Content Type 20 (ChangeCipherSpec)
- 1 - 8-bit type field
- 2 - Content Type 22 (Handshake)
- 3 - Content Type 23 (Application Data)

V: Describes the version field.

- 0 - Version 254.255 (DTLS 1.0)
- 1 - 16-bit version field
- 2 - Version 254.253 (DTLS 1.2)
- 3 - Reserved for future use

E: Describes the epoch field.

- 0 - Epoch 0
- 1 - Epoch 1
- 2 - Epoch 2
- 3 - Epoch 3
- 4 - Epoch 4

- 5 - 8-bit epoch field
- 6 - 16-bit epoch field
- 7 - Implicit -- same as previous record in the datagram

S: Describes the sequence_number field.

- 0 - Sequence number 0
- 1 - 8-bit sequence_number field
- 2 - 16-bit sequence_number field
- 3 - 24-bit sequence_number field
- 4 - 32-bit sequence_number field
- 5 - 40-bit sequence_number field
- 6 - 48-bit sequence_number field
- 7 - Implicit -- number of previous record in the datagram + 1

L: Describes the length field.

- 0 - Length 0
- 1 - 8-bit length field
- 2 - 16-bit length field
- 3 - Implicit -- last record in the datagram

4.2. Handshake Messages

Handshake messages are compressed in a similar way. A prefix is added in front of the structure to indicate the length of each field or to specify the value of the field directly. If the value is specified directly, the field itself is elided. The format of the prefix is as follows:

```

      0                               1
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
+---+---+---+---+---+---+---+---+
| 0 0 |   T   | L |   S   | O | C |
+---+---+---+---+---+---+---+---+

```

The fields in the prefix are defined as follows:

T: Describes the msg_type field.

- 0 - 8-bit msg_type field
- 1 - Handshake Type 1 (Client Hello)
- 2 - Handshake Type 2 (Server Hello)
- 3 - Handshake Type 3 (Hello Verify Request)
- 4 - Reserved for future use
- 5 - Reserved for future use
- 6 - Reserved for future use
- 7 - Handshake Type 11 (Certificate)

- 8 - Handshake Type 12 (Server Key Exchange)
- 9 - Handshake Type 13 (Certificate Request)
- 10 - Handshake Type 14 (Server Hello Done)
- 11 - Handshake Type 15 (Certificate Verify)
- 12 - Handshake Type 16 (Client Key Exchange)
- 13 - Reserved for future use
- 14 - Reserved for future use
- 15 - Handshake Type 20 (Finished)

L: Describes the length field.

- 0 - Implicit -- last message in the record
- 1 - 8-bit length field
- 2 - 16-bit length field
- 3 - 24-bit length field

S: Describes the message_seq field.

- 0 - Message sequence number 0
- 1 - Message sequence number 1
- 2 - Message sequence number 2
- 3 - Message sequence number 3
- 4 - Message sequence number 4
- 5 - Message sequence number 5
- 6 - Message sequence number 6
- 7 - Message sequence number 7
- 8 - Message sequence number 8
- 9 - Message sequence number 9
- 10 - Message sequence number 10
- 11 - Message sequence number 11
- 12 - Message sequence number 12
- 13 - 8-bit message_seq field
- 14 - 16-bit message_seq field
- 15 - Implicit -- number of previous message in the record + 1

O: Describes the fragment_offset field.

- 0 - Offset 0
- 1 - 8-bit fragment_offset field
- 2 - 16-bit fragment_offset field
- 3 - 24-bit fragment_offset field

C: Describes the fragment_length field.

- 0 - Implicit -- last message in the record
- 1 - 8-bit fragment_length field
- 2 - 16-bit fragment_length field
- 3 - 24-bit fragment_length field

5. Security Considerations

Beyond implementation techniques and stateless header compression, any changes to the TLS/DTLS protocol need to be performed extremely carefully. No analysis has been done in the present version of this draft.

6. IANA Considerations

This draft includes no request to IANA.

7. Acknowledgements

Olaf Bergmann was an original author of this draft and is acknowledged for significant contribution to this document.

Thanks to Angelo P. Castellani, Stefan Jucker, Shahid Raza, and Silke Schaefer for helpful comments and discussions that have shaped the document.

8. References

8.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.
- [RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, January 2012.

8.2. Informative References

- [DCOSS12] Raza, S., Trabalza, D., and T. Voigt, "6LoWPAN Compressed DTLS for CoAP", 8th IEEE International Conference on Distributed Computing in Sensor Systems, May 2012.
- [I-D.aks-crypto-sensors] Sethi, M., Arkko, J., Keranen, A., and H. Rissanen, "Practical Considerations and Implementation Experiences in Securing Smart Object Networks", draft-aks-crypto-sensors-02 (work in progress), March 2012.

- [I-D.bormann-6lowpan-ghc]
Bormann, C., "6LoWPAN Generic Compression of Headers and Header-like Payloads", draft-bormann-6lowpan-ghc-06 (work in progress), March 2013.
- [I-D.gilger-smart-object-security-workshop]
Gilger, J. and H. Tschofenig, "Report from the 'Smart Object Security Workshop', March 23, 2012, Paris, France", draft-gilger-smart-object-security-workshop-01 (work in progress), February 2013.
- [I-D.ietf-core-coap]
Shelby, Z., Hartke, K., and C. Bormann, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-18 (work in progress), June 2013.
- [I-D.ietf-lwig-guidance]
Bormann, C., "Guidance for Light-Weight Implementations of the Internet Protocol Suite", draft-ietf-lwig-guidance-03 (work in progress), February 2013.
- [I-D.ietf-tls-cached-info]
Santesson, S. and H. Tschofenig, "Transport Layer Security (TLS) Cached Information Extension", draft-ietf-tls-cached-info-14 (work in progress), March 2013.
- [I-D.ietf-tls-oob-pubkey]
Wouters, P., Tschofenig, H., Gilmore, J., Weiler, S., and T. Kivinen, "Out-of-Band Public Key Validation for Transport Layer Security (TLS)", draft-ietf-tls-oob-pubkey-09 (work in progress), July 2013.
- [I-D.mcgrew-tls-aes-ccm-ecc]
McGrew, D., Bailey, D., Campagna, M., and R. Dugal, "AES-CCM ECC Cipher Suites for TLS", draft-mcgrew-tls-aes-ccm-ecc-07 (work in progress), August 2013.
- [I-D.moskowitz-hip-rg-dex]
Moskowitz, R., "HIP Diet EXchange (DEX)", draft-moskowitz-hip-rg-dex-06 (work in progress), May 2012.
- [IEEE.802-15-4]
"Information technology - Telecommunications and information exchange between systems - Local and

metropolitan area networks - Specific requirements - Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs)", IEEE Standard 802.15.4, September 2006, <<http://standards.ieee.org/getieee802/download/802.15.4-2006.pdf>>.

- [RFC4302] Kent, S., "IP Authentication Header", RFC 4302, December 2005.
- [RFC4944] Montenegro, G., Kushalnagar, N., Hui, J., and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks", RFC 4944, September 2007.
- [RFC6256] Eddy, W. and E. Davies, "Using Self-Delimiting Numeric Values in Protocols", RFC 6256, May 2011.
- [RFC6298] Paxson, V., Allman, M., Chu, J., and M. Sargent, "Computing TCP's Retransmission Timer", RFC 6298, June 2011.
- [RFC6520] Seggelmann, R., Tuexen, M., and M. Williams, "Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS) Heartbeat Extension", RFC 6520, February 2012.
- [RFC6655] McGrew, D. and D. Bailey, "AES-CCM Cipher Suites for Transport Layer Security (TLS)", RFC 6655, July 2012.
- [SEC1] Brown, D., "Standards for Efficient Cryptography 1 (SEC 1): Elliptic Curve Cryptography", Version 2.0, May 2009.
- [USENIX01] Dean, D. and A. Stubblefield, "Using Client Puzzles to Protect TLS", 10th USENIX Security Symposium, August 2001, <http://static.usenix.org/events/sec01/full_papers/dean/dean.pdf>.

Appendix A. Templates

When elliptic curve cryptography is used, building and parsing the bodies of Certificate, ServerKeyExchange and ClientKeyExchange messages mainly involves the encoding and decoding of elliptic curve points. The points are encapsulated in a mix of DTLS structures and ASN.1 sequences. For a given elliptic curve, some parts of a message body are static, which allows using pre-composed messages instead of writing lots of memory consuming code pertaining to DTLS and ASN.1.

This appendix provides templates for the SubjectPublicKeyInfo structures for the named curves secp256r1, secp384r1 and secp521r1, also known as NIST P-256, P-384 and P-521, respectively. These curves are the ones required in [I-D.mcgregor-tls-aes-ccm-ecc]. Points are represented in uncompressed point format.

Note: Previous versions of the document provided templates for ServerKeyExchange and ClientKeyExchange messages. These templates were not correct, as the messages are actually variable in length depending on the sign of the encoded points.

SubjectPublicKeyInfo: secp256r1

```

30 59 30 13 06 07 2a 86 48 ce 3d 02 01 06 08 2a
86 48 ce 3d 03 01 07 03 42 00 04  _ _ _ _ _
_ _ _ _ _
_ _ _ _ _
_ _ _ _ _
_ _ _ _ _

```

SubjectPublicKeyInfo: secp384r1

```

30 76 30 10 06 07 2a 86 48 ce 3d 02 01 06 05 2b
81 04 00 22 03 62 00 04  _ _ _ _ _
_ _ _ _ _
_ _ _ _ _
_ _ _ _ _
_ _ _ _ _
_ _ _ _ _
_ _ _ _ _

```

SubjectPublicKeyInfo: secp521r1

```

30 81 9b 30 10 06 07 2a 86 48 ce 3d 02 01 06 05
2b 81 04 00 23 03 81 86 00 04  _ _ _ _ _
_ _ _ _ _
_ _ _ _ _
_ _ _ _ _
_ _ _ _ _
_ _ _ _ _
_ _ _ _ _
_ _ _ _ _
_ _ _ _ _

```

Author's Address

Klaus Hartke
Universitaet Bremen TZI
Postfach 330440
Bremen D-28359
Germany

Phone: +49-421-218-63905
Email: hartke@tzi.org

DICE
Internet-Draft
Updates: 5077, 5246 (if approved)
Intended status: Experimental
Expires: April 21, 2014

R. Hummen, Ed.
COMSYS, RWTH Aachen
J. Gilger
IT-Security, RWTH Aachen
H. Shafagh
ETH Zurich
October 18, 2013

Extended DTLS Session Resumption for Constrained Network Environments
draft-hummen-dtls-extended-session-resumption-01

Abstract

This draft defines two extensions for the existing session resumption mechanisms of TLS that specifically apply to Datagram TLS (DTLS) in constrained network environments. Session resumption type negotiation enables the client and the server to explicitly agree on the session resumption mechanism for subsequent handshakes, thus avoiding unnecessary overheads occurring with the existing specifications. Session resumption without client-side state additionally enables a constrained DTLS client to resume a session without the need to maintain state while the session is inactive. The extensions defined in this draft update [RFC5077] and [RFC5246].

Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

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 <http://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 April 21, 2014.

Copyright Notice

Copyright (c) 2013 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 (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
2. Session Resumption Type Negotiation	5
2.1. Protocol	6
2.2. ResumptionType Extension	7
3. Session Resumption Without Client-Side State	8
3.1. Protocol	8
4. Revised Recommended Ticket Construction	10
5. Security Considerations	12
5.1. Session Resumption Type Negotiation	12
6. IANA Considerations	12
7. Acknowledgements	13
8. Changelog	13
8.1. Version 1	13
8.2. Version 0	13
9. Informative References	13
Authors' Addresses	14

1. Introduction

The complex processing of DTLS handshake packets and the non-negligible computational overhead of cryptographic handshake operations - especially in case of public-key cryptography - render the use of the DTLS protocol in constrained network environments challenging. One of the main goals of the DICE WG therefore is to reduce computation and transmission overheads by defining a lightweight DTLS profile that considers the special characteristics of constrained network environments.

In addition to these efforts that mainly target the properties of the base protocol, DTLS extensions afford a further adaptation of the protocol to constrained network environments. Session resumption as

defined in [RFC5077] and [RFC5246] denotes one of these extensions. Session resumption is useful in the following scenarios considering constrained environments:

- o On-path soft-state middleboxes: Middleboxes such as stateful firewalls may require periodic keep-alive messages to allow for a bidirectional packet flow. If application data is transmitted in significantly larger time intervals than the keep-alive interval, session resumption allows to reduce the overall transmission overhead throughout the lifetime of a constrained device by tearing down a connection and resuming it when required.
- o Short-lived server sessions: Especially large-scale Internet services often employ short-lived server sessions at the security layer to efficiently handle a multitude of clients in parallel. For periodic application data transfers, this implies that constrained clients need to perform the full DTLS handshake on a regular basis. With session resumption, constrained clients can leverage a less complex abbreviated handshake to resume a session at decreased computation and transmission cost.
- o Limited server memory: A constrained server, e.g., a constrained CoAP server [I-D.ietf-core-coap], may be equipped with insufficient memory resources to handle connections for multiple clients in parallel. Session resumption allows to efficiently manage the limited memory for the per session security context by tearing down and resuming a session when required.

However, not surprisingly, the existing session resumption specifications have not specifically been designed with constrained devices (client and/or server) and networks in mind. More precisely, the abbreviated handshake in [RFC5246] requires both communication end-points to store session state across connections opportunistically. As a result of this opportunism, a constrained device may store its session state without a return on its memory investment if the DTLS peer did not maintain session state across connections as well. This is due to the lack of explicit session resumption signaling during the full handshake.

[RFC5077] enables a DTLS server to offload its state to the DTLS client for safe-keeping while the session is inactive. This mechanism largely supports the resource asymmetry when a constrained DTLS server communicates with an unconstrained DTLS client. However, it falls short for the reverse resource asymmetry, i.e., when a constrained DTLS client communicates with an unconstrained DTLS server. To leverage the vast resource difference between the DTLS client and the DTLS server in constrained network environments, there is the additional need for session resumption without client-side state.

Moreover, the roles of a DTLS client and a DTLS server may not always be readily apparent. For example, a CoAP server may not be restricted to the single role of a DTLS server, but may need to re-establish connections to other nodes due to asynchronous communication as provided by the CoAP Observe extension [I-D.ietf-core-observe]. In such situations, the CoAP server would act as a DTLS client. Hence, session resumption with state offloading also has to cover this interchangeability in roles at the DTLS layer. However, this is currently not possible when purely relying on session resumption as defined in [RFC5077].

Finally, the recommended ticket structure for stored session state as defined in [RFC5077] does not yet fully consider constrained network environments. As a result, especially certificate-based authentication leads to large ticket structures if the recommendations are followed. This in turn considerably increases transmission and memory overhead, thus requiring revised recommendations for constrained network environments.

To overcome the above shortcomings in constrained network environments, this document proposes two extensions for the existing session resumption mechanisms:

1. session resumption type negotiation, and
2. session resumption without client-side state.

Session resumption type negotiation enables the DTLS peers to explicitly negotiate the use and the type of the session resumption mechanism for the subsequent DTLS handshakes. As a result, opportunistic storing of session state is no longer required and an agreement for a specific state offloading type becomes possible. Moreover, this document specifies the required handshake signaling for session resumption without client-side state. This enables unconstrained DTLS servers to store session state on behalf of constrained DTLS clients. In combination with the existing session resumption extension specified in [RFC5077], this also allows for

session resumption when the client and server roles change at the DTLS layer.

Regarding the proposed protocol extensions, this document aims at keeping the changes to [RFC5077] minimal. To this end, the existing SessionTicket extension and the NewSessionTicket message are reused. Moreover, while this document only refers to the DTLS protocol, the defined extensions are similarly applicable to the TLS protocol.

2. Session Resumption Type Negotiation

Regarding session resumption with an abbreviated DTLS handshake as defined in [RFC5246], i.e., when both peers maintain session state across connections, DTLS currently neither provides a guarantee to the client nor to the server during the full handshake that the peer is in fact willing to store session state beyond the lifetime of the current connection. Specifically, the DTLS peers only discover during the subsequent handshake if both of them kept their session state for session resumption. However, this delayed signaling may lead to a constrained device needlessly occupying its constrained memory resources with state information while the session is inactive.

In case of session resumption without server-side state [RFC5077], the client already signals its support for this extension early during the initial full handshake by including the SessionTicket extension in the ClientHello message. The server acknowledges its own support by including the SessionTicket in the ServerHello message. Towards the end of the full handshake, the server then offloads its state to the client by means of the NewSessionTicket message. Due to this explicit negotiation in the current handshake, the client and the server do not store session state unnecessarily.

With the introduction of a third session resumption type in this document, i.e., session resumption without client-side state (see Section 3), this simple signaling mechanism introduced in [RFC5077] no longer suffices to clearly differentiate between the available session resumption types early during the Hello-phase of the DTLS handshake. Hence, additional signaling is required when reusing the SessionTicket extension for the signaling of session resumption without client-side state.

To explicitly signal the use of session resumption and to differentiate between the different state offloading types, this document defines a new session resumption type negotiation extension for the ClientHello and ServerHello messages, i.e., the ResumptionType extension. This ResumptionType extension enables the DTLS peers to clearly indicate which of the three available resumption types they support:

1. The regular abbreviated handshake (with client & server state),
2. session resumption without client-side state, and
3. session resumption without server-side state.

The integration of this extension in the DTLS handshake and the extension structure are defined in the following sections.

2.1. Protocol

The DTLS client and server use the ResumptionType extension in order to negotiate the session resumption type for the subsequent handshakes. The remaining handshake concludes as originally specified for the negotiated session resumption type. Hence, the session resumption type negotiation extends, but does not modify existing DTLS session resumption mechanisms.

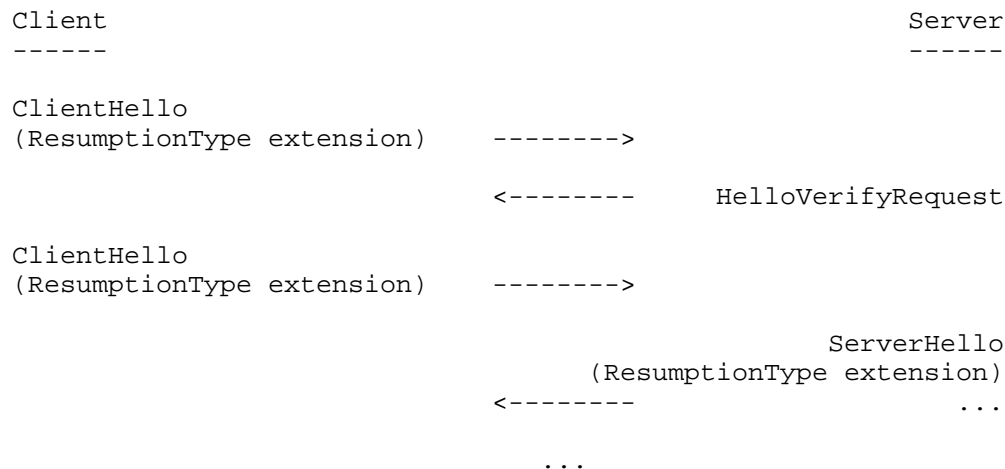


Figure 1: Message Flow for Negotiating the Session Resumption Type during a DTLS Handshake

The client adds the ResumptionType extension to its ClientHello message and indicates its supported session resumption types in the

order of preference. The server concludes the negotiation by selecting its preferred session resumption type considering the preference of the client. It signals the chosen session resumption type in the ResumptionType extension of the ServerHello message.

Each ResumptionType negotiation refers to the subsequent session resumptions. Hence, a session resumption handshake MAY omit the session resumption type negotiation. In this case, both client and server simply keep using the previously negotiated session resumption type, as long as the client and server roles have not changed. However, it is important to note that both, client and server, can resume the same DTLS session. Hence, if the roles of the client and the server have changed when the session is resumed, the ResumptionType implicitly adapts accordingly in order to keep storing session state at the same communication end-point as negotiated before. More precisely, in case of a negotiated session resumption without client-side state, state offloading follows the specified signaling of session resumption without server-side state. A negotiated session resumption without server-side state adapts vice versa. If both peers maintain session state with the regular abbreviated handshake, the change in roles does not impact this resumption type.

2.2. ResumptionType Extension

The ResumptionType extension is based on [RFC6066]. The "extension_data" field of this extension SHALL contain "ResumptionTypeList" where:

```
enum {
    abbreviated(0),
    without_client_state(1),
    without_server_state(2), (255)
} ResumptionType;

struct {
    ResumptionType resumption_type_list<1..3>
} ResumptionTypeList;
```

The ResumptionType extension may be sent in the ClientHello and ServerHello messages. The client adds the ResumptionType extension to the ClientHello message. It thereby orders the resumption types by preference. When receiving the ResumptionType extension, the server select its preferred session resumption type considering the indicated preference of the client. The server then signals the chosen session resumption type in the ResumptionType extension of the ServerHello message. Thus, the ResumptionType extension in the

ServerHello message MUST only contain a single session resumption type.

The ResumptionType extension has been assigned the number of "TBD".

3. Session Resumption Without Client-Side State

Traditional client-server communication protocols and architectures typically make the assumption of a number of clients opening connections to a single more powerful server. Scaling the system means to ensure that the server can handle the load of additional clients. With this mindset, [RFC5077] enables a DTLS server to remain stateless while the session is inactive by offloading its session state to the DTLS client.

However, in the domain of constrained network environments, not only do some devices have vastly different capabilities and resources, they regularly take the role of both client and server. In terms of higher-layer protocols such as CoAP, the distinction between client and server may still be intact while on the lower layers a device will have to accept inbound as well as establish outbound connections. This fact blurs the distinction between client and server roles at the DTLS layer.

For the communication of two devices with highly differing capabilities and resources, e.g., an unconstrained Internet host and a constrained device, enabling the constrained device to save scarce memory resources may actually help the overall system, regardless of whether it is acting as a server or a client. For example, a memory-constrained client may be able to maintain several connections sequentially, but not in parallel. Likewise, a CoAP server may take the role of a DTLS server during the initial session establishment, but re-establish the session as a DTLS client due to the asynchronous communication with CoAP Observe. To support these and other scenarios, this document introduces session resumption without client-side state in addition to the session resumption mechanisms defined in [RFC5077] and [RFC5246].

3.1. Protocol

For session resumption without client-side state, the DTLS client and server first agree on this session resumption type with a mandatory session resumption type negotiation in the full handshake. The client then sends its encrypted session state to the server.

Client

Server

```

ClientHello
(ResumptionType extension)
(empty SessionTicket extension) ----->

<----- HelloVerifyRequest

ClientHello
(ResumptionType extension)
(empty SessionTicket extension) ----->

                                ServerHello
                                (ResumptionType extension)
                                (empty SessionTicket extension)
                                ServerKeyExchange*
                                CertificateRequest*
                                <----- ServerHelloDone

Certificate*
ClientKeyExchange
CertificateVerify*
NewSessionTicket
[ChangeCipherSpec]
Finished ----->

                                [ChangeCipherSpec]
                                <----- Finished

Application Data <-----> Application Data

```

Figure 2: Message Flow for Full Handshake Issuing New Session Ticket

In the full DTLS handshake, the ClientHello message contains a ResumptionType extension indicating the willingness of the client to perform session resumption without client-side state. The ClientHello message additionally contains an empty SessionTicket extension. This extension is defined in Section 3.2 of [RFC5077].

If supported and preferred by the server, the server echoes back this type in the ResumptionType extension of the ServerHello reply. The client then sends its encrypted session state to the server in the NewSessionTicket message of the fifth message flight. The ticket contains the necessary information for the client to resume the session at a later point in time. The NewSessionTicket message is defined in Section 3.3 of [RFC5077].

```

Client                               Server
-----                               -----

```

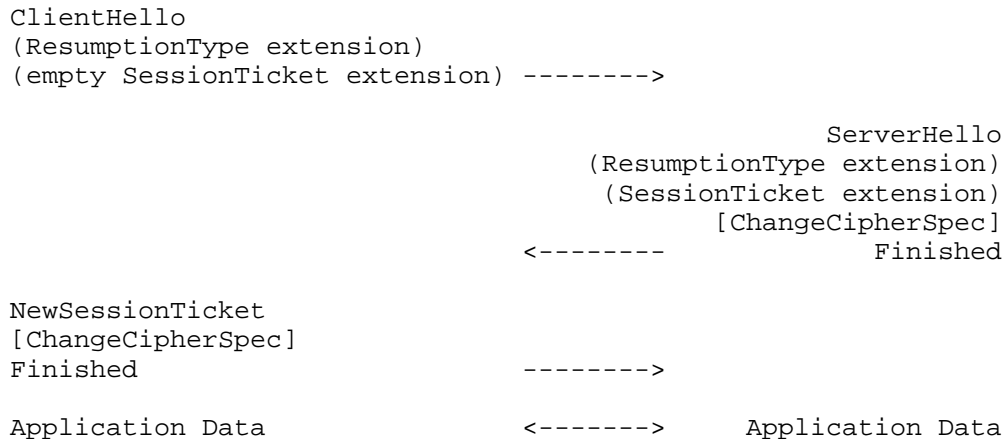


Figure 3: Message Flow for Abbreviated Handshake Using New Session Ticket

When the stateless client subsequently connects to the same server, it is oblivious of the previous full handshake. Hence, the ClientHello message in the abbreviated handshake is equal to the full handshake. On receipt of the ClientHello message, the server tries to re-identify the client (e.g. based on the source IP address or other identifying information) and searches for a matching session ticket. If it finds a matching ticket, it sends the stored session ticket to the client. To this end, the server adds the SessionTicket extension with the corresponding session ticket to its ServerHello reply.

If the client is able to authenticate and to decrypt the SessionTicket received by the server, it resumes the previous session. The client can additionally send its new session state in the NewSessionTicket message for the subsequent handshake.

4. Revised Recommended Ticket Construction

Section 4 of [RFC5077] recommends a ticket construction that may lead to an excessive ticket size for constrained network environments. This recommended ticket construction, for example, includes an entire certificate chain as the client identity in case of certificate-based authentication. The aim of this section is to provide revised recommendations for the ticket construction that take device and network constraints into account.

As defined in [RFC5077], the NewSessionTicket handshake message contains a lifetime value and a session ticket. The lifetime indicates the number of seconds until the ticket expires relative to

the time of ticket issuing. The ticket structure is opaque to the peer storing the ticket while the session is active. Only the ticket issuer needs to access the session ticket information. Hence, the specific structure of the ticket is not subject to interoperability concerns.

The revised session ticket has the following structure:

```
struct {  
    opaque key_name[8];  
    opaque iv[16];  
    opaque encrypted_state<0..2^16-1>;  
    opaque ccm_auth_tag[8];  
} ticket;
```

Regarding the above structure, `key_name` refers to the key used by the ticket issuer to protect the confidentiality and integrity of the offloaded session state information. To allow for early detection of forged session tickets during the session resumption handshake, the `key_name` SHOULD be generated randomly. The ticket issuer MUST take care that it does not use the same `key_name` for different keys.

The session state information of the revised ticket is protected by AES CCM with an 8 byte authentication tag (see [RFC3610]). The integrity protection includes the `key_name` and the `encrypted_state`. The `key_name` and `iv` are transmitted in plain. The shorter authentication tag compared to the recommendation in [RFC5077] denotes a trade-off between a lower ticket expansion and a higher probability of forgery. Moreover, with AES CCM, the stateless peer only needs to maintain a single 128-bit key instead of one 128-bit key for encryption and one 256-bit key for authentication purposes.

The `StatePlaintext` structure describes the unencrypted session state information carried in a session ticket. In this document, we define a new structure for the `peer_identity`, which is called `client_identity` in [RFC5077]. The renaming was deemed necessary due to the fact that a ticket can now be generated by a client as well as a server.

```
struct {  
    ProtocolVersion protocol_version;  
    CipherSuite cipher_suite;  
    CompressionMethod compression_method;  
    opaque master_secret[48];  
    PeerIdentity peer_identity;  
    uint32 timestamp;  
} StatePlaintext;
```

```
enum {
    anonymous(0),
    certificate_based(1),
    psk(2)
} PeerAuthenticationType;

struct {
    PeerAuthenticationType peer_authentication_type;
    select (PeerAuthenticationType) {
        case anonymous: struct {};
        case certificate_based:
            uint32 certificate_lifetime_hint;
        case psk:
            opaque psk_identity<0..2^16-1>;
    };
} PeerIdentity;
```

Here, the `certificate_lifetime_hint` indicates how long the validated certificate chain remains valid. To this end, the `certificate_lifetime_hint` holds the minimum lifetime for all certificates in a chain in seconds. If the time indicated in the lifetime hint is exceeded, a full handshake MUST be performed. Additional information may need to be added to the ticket structure in future revisions of this document in order to enable a state-offloading peer to validate the certificate status via a Certificate Revocation List (CRL) or the Online Certificate Status Protocol (OCSP) during the session resumption handshake.

5. Security Considerations

Session resumption without client-side state as defined in this document is strongly based on [RFC5077]. As such, the security considerations discussed in Section 5 of [RFC5077] apply here as well. Additional security considerations stem from the introduction of the new `ResumptionType` extension.

5.1. Session Resumption Type Negotiation

The `ResumptionType` extension is part of the regular DTLS handshake and thus covered by the hash in the Finished message. Hence, an on-path attacker cannot enforce a particular session resumption type without the peers noticing.

6. IANA Considerations

This document specifies the new ResumptionType extension for DTLS. The corresponding IANA considerations will be addressed in a future version of this document.

7. Acknowledgements

The authors would like to thank Shahid Raza for the discussion and comments regarding the extensions defined in this document. We especially acknowledge the prototyping and implementation efforts of Hossein Shafagh that confirm the feasibility of the proposed extensions in constrained network environments. Finally, the authors appreciate the feedback and suggestions of Sandeep Kumar. This work is funded by the DFG Cluster of Excellence on Ultra High- Speed Mobile Information and Communication (UMIC).

8. Changelog

8.1. Version 1

- Add scenarios where session resumption is beneficial
- Add section on ticket construction
- Minor editorial changes

8.2. Version 0

- Initial version

9. Informative References

[I-D.ietf-core-coap]

Shelby, Z., Hartke, K., and C. Bormann, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-18 (work in progress), June 2013.

[I-D.ietf-core-observe]

Hartke, K., "Observing Resources in CoAP", draft-ietf-core-observe-08 (work in progress), February 2013.

[RFC2119]

Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC3610]

Whiting, D., Housley, R., and N. Ferguson, "Counter with CBC-MAC (CCM)", RFC 3610, September 2003.

- [RFC5077] Salowey, J., Zhou, H., Eronen, P., and H. Tschofenig,
"Transport Layer Security (TLS) Session Resumption without
Server-Side State", RFC 5077, January 2008.
- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security
(TLS) Protocol Version 1.2", RFC 5246, August 2008.
- [RFC6066] Eastlake, D., "Transport Layer Security (TLS) Extensions:
Extension Definitions", RFC 6066, January 2011.

Authors' Addresses

Rene Hummen (editor)
Chair of Communication and Distributed Systems, RWTH Aachen
Ahornstrasse 55
Aachen 52074
Germany

Email: hummen@comsys.rwth-aachen.de
URI: <http://www.comsys.rwth-aachen.de/team/rene-hummen/>

Johannes Gilger
Research Group IT-Security, RWTH Aachen
Mies-van-der-Rohe Strasse 15
Aachen 52074
Germany

Email: gilger@itsec.rwth-aachen.de
URI: <http://itsec.rwth-aachen.de/people/johannes-gilger/>

Hossein Shafagh
ETH Zurich
Universitaetstrasse 6
Zurich 8092
Switzerland

Email: shafgah@inf.ethz.ch
URI: <http://www.inf.ethz.ch/~mshafagh/>

DICE Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 20, 2014

S. Keoh
University of Glasgow
S.S. Kumar, Ed.
O. Garcia-Morchon
E. Dijk
Philips Research
October 17, 2013

DTLS-based Multicast Security for Low-Power and Lossy Networks (LLNs)
draft-keoh-dice-multicast-security-00

Abstract

Wireless IP-based systems will be increasingly used for building control systems in the future where wireless devices interconnect with each other, forming low-power and lossy networks (LLNs). The CoAP/6LoWPAN standards are emerging as the de-facto protocols in this area for resource-constrained devices. Both multicast and security are key needs in these networks. This draft presents a method for securing multicast communication in LLNs based on the DTLS which is already available in CoAP devices. This draft deals with the adaptation of the DTLS record layer to protect multicast group communication, assuming that all group member devices are already configured with the group security association. The DTLS record layer implementation is used to encrypt and provide authentication to multicast messages using the group keying material before sending the message via IP multicast to the group.

Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

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 <http://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 April 20, 2014.

Copyright Notice

Copyright (c) 2013 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 (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
1.1. Terminology	4
1.2. Outline	4
2. Use Cases and Requirements	4
2.1. Use Cases	4
2.2. Security Requirements	5
3. Overview of DTLS-based Secure Multicast	7
3.1. IP Multicast	8
3.2. Securing Multicast in LLNs	8
4. Multicast Data Security	9
4.1. Sending Secure Multicast Messages	10
4.1.1. One Sender, Multiple Listeners Multicast Group	11
4.1.2. Multiple Senders, Multiple Listeners Multicast Group	12
4.2. Receiving Secure Multicast Messages	13
4.2.1. One Sender, Multiple Listeners Multicast Group	13
4.2.2. Multiple Senders, Multiple Listeners Multicast Group	13
5. IANA Considerations	14
6. Security Considerations	14
7. Acknowledgements	14
8. References	14
8.1. Normative References	14
8.2. Informative References	15
Authors' Addresses	17

1. Introduction

There is an increased use of wireless control networks in city infrastructure, environmental monitoring, industrial automation, and building management systems. This is mainly driven by the fact that the independence from physical control wires allows for freedom of placement, portability and for reducing the cost of installation as less cable placement and drilling are required. Consequently, there is an ever growing number of electronic devices, sensors and actuators that have become Internet connected, thus creating a trend towards Internet of Things (IoT). These connected devices are equipped with communication capability that enables them to interact with each other as well as with Internet services at anytime and anyplace. However, the devices in such wireless control networks are usually battery-operated or powered by scavenged energy, they have limited computational resources (low CPU clock, small RAM and flash storage) and often, the communication bandwidth is limited (e.g., IEEE 802.15.4 radio), and also the transmission is unreliable. Hence, such wireless control networks are also known as Low-power and Lossy Networks (LLNs).

In addition to the usual device-to-device unicast communication that would allow devices to interact with each other, group communication is an important feature in LLNs that can be effectively used to convey messages to a group of devices without requiring the sender to perform time- and energy-consuming multiple unicast transmissions to reach group members. For example, in a building control management system, Heating, Ventilation and Air-Conditioning (HVAC) and lighting devices can be grouped according to the layout of the building, and control commands can be issued to a group of devices. Group communication for LLNs has been made possible using the Constrained Application Protocol (CoAP) [I-D.ietf-core-coap] based on IP-multicast.

Currently, CoAP can be protected using Datagram Transport Layer Security (DTLS) [RFC6347]. However, DTLS is mainly used to secure a connection between two endpoints and it cannot be used to protect multicast group communication. We believe that group communication in LLNs is equally important and should be secured as it is also vulnerable to the usual attacks over the air (eavesdropping, tampering, message forgery, replay, etc). Although there have been a lot of efforts in IETF to standardize mechanisms to secure multicast communication, they are not necessarily suitable for LLNs which have much more limited bandwidth and resources. For example, the MIKEY Architecture [RFC3830] is mainly designed to facilitate multimedia distribution, while TESLA [RFC4082] is proposed as a protocol for broadcast authentication of the source and not for protecting the confidentiality of multicast messages.

This draft describes an approach to use DTLS as mandated in CoAP to support multicast security. It assumes that all devices in the group share a security parameters and keying material, for e.g., it can be distributed by a controller in the network through a DTLS unicast secure channel to each device in the group. This draft focuses only on the use of DTLS record layer to protect multicast messages to be sent to the group, and thus providing integrity, confidentiality and authenticity to the IP multicast messages in the LLN.

1.1. Terminology

This specification defines the following terminology:

Controller: The entity that is responsible for creating a multicast group, adding members, and distributing keying material to members of the group. It is also responsible for renewing/updating the multicast group keying material. It is not necessarily the sender in the multicast group.

Sender: The entity that sends multicast messages to the multicast group.

Listener: The entity that receives multicast messages when listening to a multicast IP address.

1.2. Outline

This draft is structured as follows: Section 2 motivates the proposed solution with multicast use cases in LLNs and derives a set of requirements. Section 3 provides an overview of the DTLS-based multicast security. In Section 4, we describe the use of DTLS record layer to encrypt and integrity protect multicast messages assuming that all devices in the group already have a security parameters and group keying material in possession. Section 5 and Section 6 describe Security and IANA considerations.

2. Use Cases and Requirements

This section defines the use cases for multicast and specifies a set of security requirements for these use cases.

2.1. Use Cases

As stated in the Group Communication for CoAP Internet Draft [I-D.ietf-core-groupcomm] in the IETF CoRE WG, multicast is essential in several application use cases. Consider a building equipped with 6LoWPAN [RFC4944] IP-connected lighting devices, switches, and 6LoWPAN border routers; the devices are organized as groups according

to their location in the building, e.g., lighting devices and switches in a room/floor can be configured as a multicast group, the switches are then used to control the lighting devices in the group by sending on/off/dimming commands to the group. 6LoWPAN border routers that are connected to an IPv6 network backbone (which is also multicast enabled) are used to interconnect 6LoWPANs in the building.

Consequently, this would also enable multicast groups to be formed across different subnets in the entire building. The following lists a few multicast group communication uses cases in a building management system; a detailed description of each use case can be found in Group Communication for CoAP Internet Draft [I-D.ietf-core-groupcomm].

- a. Lighting control: enabling synchronous operation of a group of 6LoWPAN connected lights in a room/floor/building. This ensures that the light preset of a large group of luminaries are changed at the same time, hence providing a visual synchronicity of light effects to the user.
- b. Firmware update: firmware of devices in a building or a campus control application are updated simultaneously, avoiding an excessive load on the LLN due to unicast firmware updates.
- c. Parameter update: settings of devices are updated simultaneously and efficiently.
- d. Commissioning of above systems: information about the devices in the local network and their capabilities can be queried and requested, e.g. by a commissioning device.

2.2. Security Requirements

The Miscellaneous CoAP Group Communication Topics Internet Draft [I-D.dijk-core-groupcomm-misc] has defined a set of security requirements for group communication in LLNs. We re-iterate and further describe those security requirements in this section with respect to the use cases as presented in Section 2.1:

- a. Multicast communication topology: We consider both one-to-many and many-to-many communication topologies in this draft. The one-to-many communication topology is the simplest group communication scenario that would serve the needs of a typical LLN. For example, in the lighting control use case, the switch is the only entity that is responsible for sending control commands to a group of lighting devices. These lighting devices are actuators that do not issue commands to each other. In other use cases, a many-to-many multicast communication topology would be required, in particular multiple sensors and actuators are

part of a multicast group and these sensors will trigger events to the group in order to notify the interested parties. Devices in the group could also send commands in order to trigger some actions on other devices in the group.

- b. Establishment of a Group Security Association (GSA) [RFC3740]: A secure channel must be used to distribute keying material, multicast security policy and security parameters to members of a multicast group. A GSA must be established between the controller (which manages the multicast group and may be a different device than the sender) and the group members. The 6LoWPAN border router, a device in the 6LoWPAN, or a remote server outside the 6LoWPAN could play the role of controller for distributing keying materials. Since the keying material is used to derive subsequent group keys to protect multicast messages, it is important that it is encrypted, integrity protected and authenticated when it is distributed. However, this is out of scope of this draft, and it is anticipated that an activity in IETF dedicated to the design of a generic key management scheme for the LLN will be started in the future.
- c. Multicast security policy: All group members must use the same ciphersuite to protect the authenticity, integrity and confidentiality of multicast messages. The ciphersuite can either be negotiated or set by the controller and then distributed to the group members. It is generally very complex and difficult to require all devices to negotiate and agree with each other on the ciphersuite to be used, it is therefore more effective that the multicast security policy is set by the controller.
- d. Multicast data group authentication: It is essential to ensure that a multicast message is originated from a member of the group. The multicast group key which is known to all group members is used to provide authenticity to the multicast messages (e.g., using a Message Authentication Code, MAC). This assumes that only the sender of the multicast group is sending the message, and that all other group members are trusted not to tamper with the multicast message.
- e. Multicast data source authentication: Source authenticity is optional. It can typically be provided using public-key cryptography in which every multicast message is signed by the sender. This requires much higher computational resources on both the sender and the receivers, thus incurring too much overhead and computational requirements on devices in LLNs. Alternatively, a lightweight broadcast authentication, i.e., TESLA [RFC4082] can be deployed, however it requires devices in

the multicast group to have a trusted clock and have the ability to loosely synchronize their clocks with the sender. Consequently, given that the targeted devices have limited resources, and the need for source authenticity is not critical, it is advocated that source authenticity is made optional.

- f. Multicast data integrity: A group level integrity is required to ensure that messages have not been tampered with by attackers who are not members of the multicast group.
- g. Multicast data confidentiality: Multicast message may be encrypted, as some control commands when sent in the clear could pose privacy risks to the users.
- h. Multicast data replay protection: It must not be possible to replay a multicast message as this would disrupt the operation of the group communication.
- i. Multicast key management: Group keys used to protect the multicast communication must be renewed periodically. When members have left the multicast group, the group keys might be leaked; and when a device is detected to have been compromised, this also implies that the group keys could have been compromised too. In these situations, the controller must perform a re-key protocol to renew the group keys. This work will be addressed as part of the key management for LLN in the future based on [RFC3740] and [RFC4046].

3. Overview of DTLS-based Secure Multicast

The goal of this draft is to secure COAP group communication over 6LoWPAN networks, by extending the use of the DTLS security protocol to allow for the use of DTLS record layer to provide protection to multicast messages. The IETF CoRE WG has selected DTLS [RFC6347] as the default must-implement security protocol for securing CoAP, therefore it is conceivable that DTLS can be extended to facilitate CoAP-based group communication. Reusing DTLS for different purposes while guaranteeing the required security properties can avoid the need to implement multiple security protocols and this is especially beneficial when the target deployment consists of resource-constrained embedded devices. This section first describes group communication based on IP multicast, and subsequently sketches a solution for securing group communication using DTLS.

management for LLN is out of scope of this draft, and we assume that each device in the group has been configured with a GSA using a.

Senders in the group can encrypt and authenticate application messages using the keying material in the DTLS record layer before it is sent using IP multicast. For example, a CoAP message addressed to a multicast group is protected using DTLS record layer and then sent to a multicast group. The listeners when receiving the message, use the multicast IP destination address (i.e., Multicast identifier) to look up the GSA needed for that connection. The received message is decrypted and the authenticity is verified using the keying material for that connection.

4. Multicast Data Security

This section describes in detail the use of DTLS record layer to secure multicast messages. This assumes that group membership has been configured by the controller, and all devices in the group have been configured with the GSA. Since the exact details of the group key management are outside the scope of this draft, we assume that the GSA can be used to derive the same SecurityParameters structure as defined in [RFC5246] for all devices. Additional ciphersuites may need to be defined to convey the bulk cipher algorithm, MAC algorithm and key lengths within the key management protocol. We provide two such examples of ciphersuites that could be defined as part of a future key management mechanism:

```
Ciphersuite MTS_WITH_AES_128_CCM_8 = {TBD1, TBD2}
Ciphersuite MTS_WITH_NULL_SHA256   = {TBD3, TBD4}
```

Ciphersuite MTS_WITH_AES_128_CCM_8 is used to provide confidentiality, integrity and authenticity to the multicast messages where the encryption algorithm is AES [AES], key length is 128-bit, and the authentication function is CCM [RFC6655] with a Message Authentication Code (MAC) length of 8 bytes. Similar to RFC4785 [RFC4785], the ciphersuite MTS_WITH_NULL_SHA is used when confidentiality of multicast messages is not required, it only provides integrity and authenticity protection to the multicast message. When this ciphersuite is used, the message is not encrypted but the MAC must be included in which it is computed using a HMAC [RFC2104] that is based on Secure Hash Function SHA256 [SHA]. Depending on the future needs, other ciphersuites with different cipher algorithms and MAC length may be supported.

The SecurityParameters.ConnectionEnd should be set to "server" for senders and "client" for listeners. The current read and write states can be derived from SecurityParameters by generating the six key material items:

```
client write MAC key
server write MAC key
client write encryption key
server write encryption key
client write IV
server write IV
```

This requires that the `client_random` and `server_random` within the `SecurityParameters` are set same for all devices as part of the key management protocol to derive the same keying material for all devices in the group with the PRF function defined in Section 6.3 of [RFC5246]. Alternatively, the key management protocol could directly provide the above six key material to all group devices as part of the GSA.

The current read and write states are instantiated for all group members based on the keying material; senders use "server write" parameters for the write state and listeners use "server write" parameters for the read state. Additionally each connection state contains the sequence number which is incremented for each record sent; the first record sent has the sequence number 0.

For the optional multicast data source authentication, the sender can sign the message using public key cryptography at the application layer and send it as the multicast message in the DTLS record payload. This option is independent of the DTLS layer and outside the scope of this draft.

4.1. Sending Secure Multicast Messages

All messages addressed to the multicast group must be secured using "server write" parameters. Using the DTLS record layer, multicast messages are encrypted and protected using a Message Authentication Code (MAC) according to the chosen ciphersuite. The authenticated encrypted message is passed down to the lower layer of the IP protocol stack for transmission to the multicast address.

As described in the previous section, the example ciphersuite `MTS_WITH_AES_128_CCM_8` defines that the multicast message must be encrypted using AES with a 128-bit "server write encryption key". Since the CCM mode of operation is used for authenticated encryption, the same key is used to compute the MAC. As for the ciphersuite example `MTS_WITH_NULL_SHA`, the multicast message must not be encrypted, but a MAC must be computed using the "server write MAC key".

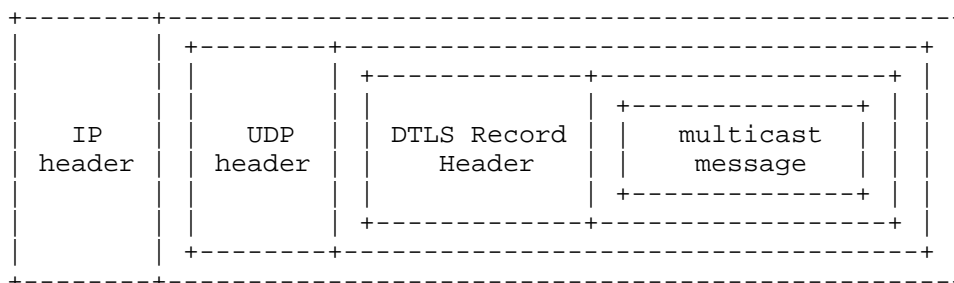


Figure 4.1: Sending a multicast message protected using DTLS Record Layer

4.1.1. One Sender, Multiple Listeners Multicast Group

This section describes the use of DTLS record layer to protect a one-sender, multiple-listeners multicast group communication. In this setting, it is the responsibility of the controller which configures the group membership to ensure that there is only one sender in a multicast group and other devices never send multicast messages to the same group in order to ensure the security properties of the multicast messages. This is especially a concern in AEAD cipher suites if multiple senders reuse the same nonce for encryption as described in Section 5.1.1 in [RFC5116].

The following illustrates the structure of the DTLS record layer header, the epoch and sequence number are used to ensure message freshness and to detect message replays. As there is only one sender in the multicast group, the sender is responsible for maintaining and manipulating the epoch and sequence number when sending multicast messages. The receivers in the group are "trusted" not to tamper with these parameters.

1 Byte	2 Byte	2 Byte	6 Byte	2 Byte		
Content Type	Version Ma Mi	Epoch	Seq Number	Length	Ciphertext (Enc)	MAC

Figure 4.2: The DTLS record layer header and optionally encrypted payload and MAC

The sequence number is initialized to 0, and it is increased by one whenever the sender sends a new multicast record message. This is the standard behavior of the current DTLS in order to detect message replay. The sender or the controller can increase the epoch number by sending a ChangeCipherSpec message whenever the sequence number

has been exhausted, or whenever the ciphersuite has been changed in order to reset the sequence number. Finally, the multicast message is protected (encrypted if needed, and authenticated with a MAC) using the "server write" parameters.

4.1.2. Multiple Senders, Multiple Listeners Multicast Group

There is a need to support multi-senders in group communication. In particular, in a lighting network there are multiple presence sensors that would be assigned the sender role as they are responsible for multicasting the presence information to the luminaries in the group. In this section, we outline an approach to enable all senders in the group to securely send information using a common group key, while preserving the freshness and integrity of the messages.

In addition to configuring each device in the group with the GSA, the controller can assign a unique SenderID (represented as two octets) to each device which has the sender role in the group. The list of SenderIDs are then distributed to all the group members by the controller. This is an additional group setup procedure that should be performed to ensure that each sender in the group can be uniquely identified by the group members. Alternatively, this setup procedure can be eliminated by allowing senders to derive their SenderIDs themselves based on the device's IPv6 or MAC address, or even randomly. The specific method to be used is not defined here, except care should be taken that it would lead to a high probability of unique SenderIDs for all senders within the specific multicast group. To overcome potential clash in SenderIDs, a back-off mechanism is defined in the Security Considerations section.

The existing DTLS record layer header is adapted such that the 6-byte sequence number field is split into a 2-byte SenderID field and a 4-byte "truncated" sequence number field. Each sender in the group uses its own unique SenderID in the DTLS record layer header when sending a multicast message to the group. It also manages its own epoch and "truncated" sequence number in the "server write" connection state, hence they do not need to synchronize them with other senders in the group. Figure 4.3 illustrates the adapted DTLS record layer header.

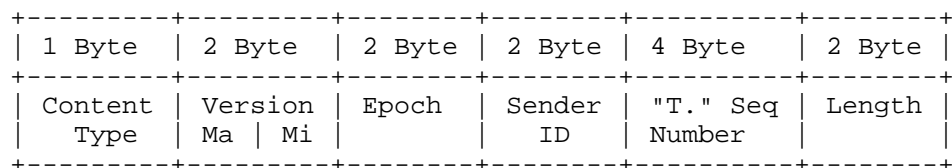


Figure 4.3: The adapted DTLS record layer header

4.2. Receiving Secure Multicast Messages

4.2.1. One Sender, Multiple Listeners Multicast Group

When a listeners receives a protected multicast message from the sender, it looks up the corresponding "client read" connection state based on the multicast IP destination of the packet. This is fundamentally different from standard DTLS logic in that the current "client read" connection state is bound to the source IP address. However, given that this is a one sender- multiple listeners communication topology, it is possible to bind the current "client read" connection state to the source IP address if it is already known to all listeners. Therefore a lookup based on the source IP address is also possible in this case.

The listeners authenticate and decrypt the multicast message using the "server write" keys. The verification of MAC ensures that the payload and the DTLS Record Layer header have not been tampered with. As there is only one sender, and all other group members are "trusted", only the sender is able to manipulate the epoch and the sequence number, hence once the DTLS header has been authenticated, the epoch and the sequence number can be sufficiently trusted to detect any message replay.

4.2.2. Multiple Senders, Multiple Listeners Multicast Group

Listener devices in a multi-senders multicast group, need to store multiple "client read" connection states for the different senders linked to the SenderIDs. The keying material is same for all senders however the epoch and the "truncated" sequence number of the last received packets needs to be kept different for different senders. The listeners first perform a "server write" keys lookup by using the multicast IP destination address of the packet. By knowing the keys, the listeners decrypt and check the MAC of the message. This guarantees that no one has spoofed the SenderID, as it is protected by the MAC. Subsequently, by authenticating the SenderID field, the listeners retrieve the "client read" connection state which contains the last stored epoch and "truncated" sequence number of the sender, which is used to check the freshness of the message received. The listeners must ensure that the epoch is the same and "truncated" sequence number in the message received is higher than the stored value, otherwise the message is discarded. As each sender manages its own epoch and sequence number, receivers are confident that these values are reliable. Once the authenticity and freshness of the message have been checked, the listeners can pass the message to the higher layer protocols. The epoch and the sequence number in the corresponding "client read" connection state are updated as well.

5. IANA Considerations

tbd

Note to RFC Editor: this section may be removed on publication as an RFC.

6. Security Considerations

This document discusses various design aspects for multicast security in LLNs. As such this document, in entirety, concerns security.

Section 4.1.2 with multiple senders require that SenderIDs are unique to maintain the security properties of the DTLS record layer messages. However in the event that two or more senders are configured with the same SenderID, a mechanism needs to be present to avoid a security weakness and recover from the situation. One such mechanism is that all senders of the mutlicast group are also listeners. This allows a sender which receives a packet from a different device with its own SenderID in the DTLS header to be aware of a clash in SenderID. Once aware, the sender can inform the controller on a secure channel about the clash along with the source IP address. The controller can then provide a different SenderID to either device or both.

Section 4.1.2 additionally truncates the sequence number from 6 octets to 4 octets. This reduction of the sequence number space should be taken into account to ensure that epoch is incremented before the "truncated" sequence number wraps over. This should be done with an appropriate key management mechanism which is not defined in this draft.

7. Acknowledgements

The authors greatly acknowledge discussion, comments and feedback from Dee Denteneer, Peter van der Stok and Zach Shelby. Additionally thank David McGrew for suggesting options for recovering from a SenderID clash. We also appreciate prototyping and implementation efforts by Pedro Moreno Sanchez who worked as an intern at Philips Research.

8. References

8.1. Normative References

[AES] National Institute of Standards and Technology, ,
 "Specification for the Advanced Encryption Statndard
 (AES)", FIPS 197, Nov 2001.

- [SHA] National Institute of Standards and Technology, , "Secure Hash Standard", FIPS 180-2, Aug 2002.
- [RFC2104] Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, February 1997.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC3830] Arkko, J., Carrara, E., Lindholm, F., Naslund, M., and K. Norrman, "MIKEY: Multimedia Internet KEYing", RFC 3830, August 2004.
- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.
- [RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, January 2012.
- [RFC6655] McGrew, D. and D. Bailey, "AES-CCM Cipher Suites for Transport Layer Security (TLS)", RFC 6655, July 2012.
- [RFC5116] McGrew, D., "An Interface and Algorithms for Authenticated Encryption", RFC 5116, January 2008.

8.2. Informative References

- [I-D.dijk-core-groupcomm-misc]
Dijk, E. and A. Rahman, "Miscellaneous CoAP Group Communication Topics", draft-dijk-core-groupcomm-misc-04 (work in progress), June 2013.
- [I-D.ietf-core-coap]
Shelby, Z., Hartke, K., and C. Bormann, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-18 (work in progress), June 2013.
- [I-D.ietf-core-groupcomm]
Rahman, A. and E. Dijk, "Group Communication for CoAP", draft-ietf-core-groupcomm-16 (work in progress), October 2013.
- [I-D.ietf-tls-oob-pubkey]
Wouters, P., Tschofenig, H., Gilmore, J., Weiler, S., and T. Kivinen, "Out-of-Band Public Key Validation for

Transport Layer Security (TLS)", draft-ietf-tls-oob-pubkey-09 (work in progress), July 2013.

- [I-D.ietf-core-resource-directory]
Shelby, Z., Krco, S., and C. Bormann, "CoRE Resource Directory", draft-ietf-core-resource-directory-00 (work in progress), June 2013.
- [I-D.vanderstok-core-dna]
Stok, P., Lynn, K., and A. Brandt, "CoRE Discovery, Naming, and Addressing", draft-vanderstok-core-dna-02 (work in progress), July 2012.
- [RFC4082] Perrig, A., Song, D., Canetti, R., Tygar, J., and B. Briscoe, "Timed Efficient Stream Loss-Tolerant Authentication (TESLA): Multicast Source Authentication Transform Introduction", RFC 4082, June 2005.
- [RFC4785] Blumenthal, U. and P. Goel, "Pre-Shared Key (PSK) Ciphersuites with NULL Encryption for Transport Layer Security (TLS)", RFC 4785, January 2007.
- [RFC4944] Montenegro, G., Kushalnagar, N., Hui, J., and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks", RFC 4944, September 2007.
- [RFC6763] Cheshire, S. and M. Krochmal, "DNS-Based Service Discovery", RFC 6763, February 2013.
- [RFC3740] Hardjono, T. and B. Weis, "The Multicast Group Security Architecture", RFC 3740, March 2004.
- [RFC5374] Weis, B., Gross, G., and D. Ignjatic, "Multicast Extensions to the Security Architecture for the Internet Protocol", RFC 5374, November 2008.
- [RFC4046] Baugher, M., Canetti, R., Dondeti, L., and F. Lindholm, "Multicast Security (MSEC) Group Key Management Architecture", RFC 4046, April 2005.

Authors' Addresses

Sye Loong Keoh
University of Glasgow Singapore
Republic PolyTechnic, 9 Woodlands Ave 9
Singapore 838964
SG

Email: SyeLoong.Keoh@glasgow.ac.uk

Sandeep S. Kumar
Philips Research
High Tech Campus 34
Eindhoven 5656 AE
NL

Email: sandeep.kumar@philips.com

Oscar Garcia-Morchon
Philips Research
High Tech Campus 34
Eindhoven 5656 AE
NL

Email: oscar.garcia@philips.com

Esko Dijk
Philips Research
High Tech Campus 34
Eindhoven 5656 AE
NL

Email: esko.dijk@philips.com

Internet Draft
Intended Status: Informational
Expires: December 14, 2013

S. Keoh
S. Kumar
Philips Research
Z. Shelby
Sensinode
June 12, 2013

Profiling of DTLS for CoAP-based IoT Applications
draft-keoh-dtls-profile-iot-00

Abstract

This document collects various implementation challenges of DTLS on embedded systems, and proposes a profile of DTLS for CoAP-based Internet of Things (IoT) applications. Specifically, this document investigates the application features and functionality of DTLS protocol, the fragmentation issue of DTLS Handshake protocol, and the complexity of the DTLS Handshake state machine. A RESTful DTLS Handshake which relies on CoAP Block-wise Transfer is proposed to address the fragmentation issue. The next step is to define a DTLS profile for embedded systems.

Status of this Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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."

The list of current Internet-Drafts can be accessed at
<http://www.ietf.org/lid-abstracts.html>

The list of Internet-Draft Shadow Directories can be accessed at
<http://www.ietf.org/shadow.html>

Copyright and License Notice

Copyright (c) 2013 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 (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1	Introduction	3
1.1	Terminology	3
2	Implementation Issues of DTLS on Embedded Systems	3
2.1	Some Considerations of DTLS Protocol	3
2.2	Complexity of the Handshake Protocol State Machine	5
2.3	Code size	6
2.4	Handshake Message Fragmentation	6
3	Possible Mitigation Strategy	7
3.1	Profiling of DTLS	7
3.1.1	Cipher suites	7
3.1.2	DTLS Extensions	8
3.1.3	Fine-tuning DTLS functionality	8
3.2	Using CoAP Block-wise Transfers	8
4	Next Steps	12
3	Security Considerations	13
4	IANA Considerations	13
5	References	13
5.1	Normative References	13
5.2	Informative References	13
	Authors' Addresses	15

1 Introduction

With the completion of the Constrained Application Protocol (CoAP) [I-D.ietf-core-coap] specification, it is expected that there will be million of devices deployed in various application domains in the future. These applications range from smart energy, smart grid, building control, intelligent lighting control, industrial control systems, asset tracking, to environmental monitoring. CoAP would become the de-facto standard protocol to enable interaction between devices and to support Internet-of-Thing (IoT) applications. Security is important to protect the communication between devices, and Datagram Transport Layer Security (DTLS) [RFC6347] has been chosen as the mandatory to implement protocol for this purpose.

Over the past few years, there have been many efforts to implement DTLS on embedded systems [TINYDTLS, CONTIKI-DTLS] in order to support Internet of Things (IoT) applications. In fact, the Transport Layer Security (TLS) [RFC6347] and its datagram variant were both invented for use in the Internet-based web applications, and implementers face many challenges to deploy (D)TLS on IoT devices that are limited in memory resources (RAM, Flash), CPU and power. This Internet Draft aims to document the immediate problems that hinder the deployment of DTLS on embedded systems and proposes a DTLS profile for CoAP-based IoT applications. In particular, a approach to use the CoAP Blockwise Transfer [I-D.ietf-core-block] to perform DTLS Handshake is presented.

1.1 Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

2 Implementation Issues of DTLS on Embedded Systems

2.1 Some Considerations of DTLS Protocol

The Datagram Transport Layer Security (DTLS) protocol currently consists of Handshake protocol, Alert Protocol, ChangeCipherSpec protocol and Application protocol. The Handshake protocol is performed between a client and a server in order to authenticate each other and subsequently establishes a common key to protect the communication channel. This protocol is the essence of DTLS to provide security to both communicating parties. There are various authentication and key establishment ciphersuites that use public-key certificates, as they involve expensive cryptographic operations, DTLS provides a mechanism in the handshake protocol to re-establish previously setup connection, called RESUME. The session ID can be

used by the client and the server to retrieve the cryptographic parameters previously negotiated, and reuse the secure channels, hence avoiding the expensive certificate verification operations. Most of the implementations for embedded systems available today only implement the basic Handshake protocol, and they do not support session resume. Implementations such as [I-D.keoh-lwig-dtls-iot, TINYDTLS, CONTIKI-DTLS] only support DTLS with Pre-Shared Key (PSK) scheme, and it is not entirely clear whether raw public-key [I-D.ietf-tls-oob-pubkey] and certificate can be supported in the future. However, in order to support better interoperability between devices manufactured by different manufacturers, raw public-keys and public-key certificates should be used. However, the following considerations need to be further studied:

- a. The current DTLS Handshake protocol has the option of not performing client authentication. If mutual authentication is always needed for IoT applications, then the option of "server authentication only" may not need to be supported. This is especially useful for Raw Public-key and/or Certificates mode of operations.
- b. The RESUME protocol is useful for IoT applications. On one hand, it simplifies the Handshake protocol if devices need to re-use the previous security parameters, on the other hand it increases the complexity of the DTLS Handshake state machine. It is very likely that the DTLS sessions in IoT application are meant to be longlive, and re-handshake or resumption of previous session could be avoided if possible.
- c. DTLS supports a wide variety of ciphersuites, the IoT community advocates that this can be reduced to a selected few, i.e., ciphersuites that are based on PSK, ECC, and AES CCM [I-D.mcgrewtls-aes-ccm].

The Alert protocol is used to signal warnings and fatal errors while performing the DTLS handshake as well as during the DTLS session. Alert messages can be sent at any time during the handshake and up to the closure of the session. The Alert protocol is optional to implement and most of the DTLS implementations for embedded systems do not implement this.

The ChangeCipherSpec protocol is used to signal transition in ciphering strategies, and the ChangeCipherSpec message must be sent by both the client and the server. This protocol is primarily used during the Handshake protocol when the security parameters have been agreed upon.

- d. Since the DTLS session in IoT application is longlive, the

possibility of re-negotiating a new ciphering strategy is quite low. Additionally, given that the number of supported ciphersuites are limited, IoT devices typically do not change their agreed ciphersuites so frequently.

Application protocol uses the negotiated security parameters to protect the application data. The application data is encrypted and its authenticity is guaranteed with a Message Authentication Code (MAC).

- e. The AEAD cipher that uses CCM mode in AES is effective in the sense that it uses the same key for both encryption and authentication. If the encryption is not needed, the CBC-MAC can be used to provide message authentication, therefore eliminating the need to implement HMAC on embedded devices. (this does not eliminate the SHA function used in PRF, and HMAC is a simple algorithm that is based on SHA.)

2.2 Complexity of the Handshake Protocol State Machine

As DTLS relies on UDP transport, delivery of handshake messages cannot be guaranteed in which handshake messages might be lost during transmission and potentially arrive out-of-order. This means that lost messages must be re-transmitted to ensure the success of the Handshake protocol. Although DTLS proposed to use "flight" to group messages and hence reducing the need to re-order messages, it suffers from message fragmentation issue, which will be discussed in Section 2.4.

A cookie mechanism is used in DTLS to thwart Denial-of-Service (DoS) attacks. This not only increases the number of messages that need to be exchanged between the client and server, it also increases the complexity of the state machine. Although the cookie mechanism is optional in DTLS handshake protocol, we believe that DoS protection is especially important in the IoT applications. As most of the applications expect the devices deployed in the field to play the DTLS Server role, they will be vulnerable to DoS attacks.

There are many ways of performing authentication using DTLS. The PSK mode of operation is pretty straightforward, but when raw public-key and certificates are used, it is possible to either use the keying materials in the certificate or the ephemeral keys to perform authenticated key exchange. The certificate verification process is also very complex in that it requires the verification of the exchanged certificate up to the trust anchor, and checking the time validity of the certificate. Devices which do not have a trusted time/clock will not be able to check the expiration time of a certificate.

- a. Would it be sufficient to just verify the signature of the certificate and/or certificate chain against a trust anchor, and do not care about time validity?
- b. Is there a need to support ephemeral key exchange in IoT applications?

2.3 Code size

From various DTLS implementation experiences [I-D.keoh-lwig-dtls-iot, I-D.aks-crypto-sensors], the codesize may require further reduction in order to allow for other functionality to be incorporated such as routing, 6LoWPAN networking stack, CoAP, and application-layer codes. Most of the implementations only support the PSK mode, and this already consumes approximately 16KB of Flash and 4KB of RAM [I-D.keoh-lwig-dtls-iot]. If Raw Public-key and Certificate schemes are to be implemented, the codesize of the DTLS implementation would definitely increase dramatically. This has severe implication on constrained devices especially the Class 1 devices [I-D.ietf-lwig-guidance, I-D.ietf-lwig-terminology].

2.4 Handshake Message Fragmentation

A study conducted in [I-D.hartke-core-codtls] revealed that it is highly likely that DTLS Handshake protocol suffers from fragmentation problems when the DTLS record is too large to fit into a single 6LoWPAN payload. As the physical layer MTU of [IEEE.802-15-4] only has 127 bytes, when adding the MAC layer, the 6LoWPAN adaptation layer headers, and 25 Bytes of DTLS headers, there are only 60-70 bytes left for the DTLS handshake messages. The DTLS-PSK Handshake protocol can barely fit every message using a single 6LoWPAN payload (except the stateless cookie exchange messages). However, when Raw Public-key and Certificate are used, it would not be possible to do so and fragmentation support will be needed.

- a. Both 6LoWPAN and DTLS offer fragmentation support. 6LoWPAN supports fragmentation of IPv6 packets into small link-layer frames, but it might not work well for constrained applications and networks. DTLS offers fragmentation at the handshake layer, however this can add a significant overhead due to packet loss and the need for a buffer to enable message re-ordering.
- b. According to [I-D.hartke-core-codtls] fragmented handshake messages can arrive at a constrained node in any order, the receiver must provide a message buffer that is large enough to hold multiple fragments. When several handshake messages forming a single flight are sent out in parallel, it is likely that the receiver's resources are too limited to order fragments from

distinct handshake messages. Furthermore, since handshake messages can be fragmented arbitrarily and with overlaps, the receiver must, in addition to the message buffer, keep track of the fragments received so far.

Possible retransmission require even more buffer space as replay-protection requires encryption of every single packet that is to be transmitted. In particular, this renders destructive in-place encryption impossible as the source data must be preserved.

- c. Some implementers suggest that the 6LoWPAN General Header Compression (GHC) [I-D.bormann-6lowpan-ghc] can be used to reduce the number of bytes to be transferred. If the headers can be compressed, this reduces the number of packets that need to be transmitted.

3 Possible Mitigation Strategy

3.1 Profiling of DTLS

DTLS can be successfully used in constrained environments if smart choices can be made from the multiple options that exist in DTLS that was designed for the web applications world. This compact "IoT profile" should allow for a compact implementation (in terms of code size and RAM) and simplified handshake between IoT devices.

3.1.1 Cipher suites

Most constrained IoT devices will not be able to support multiple cipher implementations due to code space requirements. It can be beneficial to choose a few cipher suite profiles that could cover the security requirements for most IoT applications. In choosing these cipher suite profiles, reuse of the same crypto primitives to achieve different security functionality can reduce implementation costs.

Symmetric cipher based confidentiality and authentication functionality can be achieved by using the AES in CCM mode of operation. Further, the AES-CCM operation is built-in on many 802.15.4 hardware chips further reducing the need in code and also accelerate the computation. [I-D.mcgrew-tls-aes-ccm] indicates different ciphersuites based on AES-CCM for TLS.

For public key based handshake, ECC is very suitable for constrained devices. However there are multiple options in terms of field types and curves that can be chosen for a cipher suite [RFC4492]. Additionally for certificate based cipher suites, choosing the certificate signing algorithm to be also ECC based avoids the need for an additional crypto primitive implementations on the constrained

devices. Selecting a field, curve and algorithm that would ensure security of IoT applications as a public key based IoT cipher suite can substantially reduce the negotiation required in the handshake phase.

3.1.2 DTLS Extensions

Further improvements to DTLS in constrained environments can be made by choosing some of the TLS extensions [RFC6066] that are always supported by the end-points. Some of these extensions have been designed for constrained networks which can be used to define the DTLS IoT profile.

The "Maximum Fragment Length Negotiation" extension enables a smaller fragment sizes that would reduce the amount of fragmentation at the lower layers. "Client Certificate URLs" extension reduces the need for sending the certificates in the handshake message reducing bandwidth requirements and fragmentation due to large certificates. Other extensions that may be useful are the "Trusted CA Indication", "Truncated HMAC" and "Certificate Status Request".

By choosing a mandatory set of extensions as part of the DTLS IoT profile will make DTLS more efficient in constrained environments.

3.1.3 Fine-tuning DTLS functionality

Savings can be done also by choosing not to implement certain DTLS functional logic that is not expected to be used in most IoT applications. Some of these have been suggested in the previous section like not requiring the RESUME protocol or reducing the number of error handling logic as part of the Alert protocol. These reduced functionality should not in any way affect the security of the DTLS but only reduce the flexibility that was designed into DTLS as a web protocol but may not be required in IoT applications.

Additionally, timer values for retransmission can be adjusted to prevent unnecessary congestion due to the underlying lossy network which can be aggravated due to large flight messages being resent at short intervals.

Thus the DTLS IoT profile can be a combination of cipher suites, DTLS extensions and fine-tuning functionality that makes it suitable for constrained devices and networks.

3.2 Using CoAP Block-wise Transfers

CoAP has defined block-wise transfers to allow for the delivery of

large payload between CoAP client and server. Therefore, instead of relying on 6LoWPAN fragmentation and DTLS fragmentation which do not work well, the DTLS Handshake protocol can be done in combination with CoAP to better manage the fragmentation issues. Using this approach, the DTLS message flight mechanism can be used, thus reducing the number of messages to be exchanged between the client and server to six as illustrated in Figure 1.

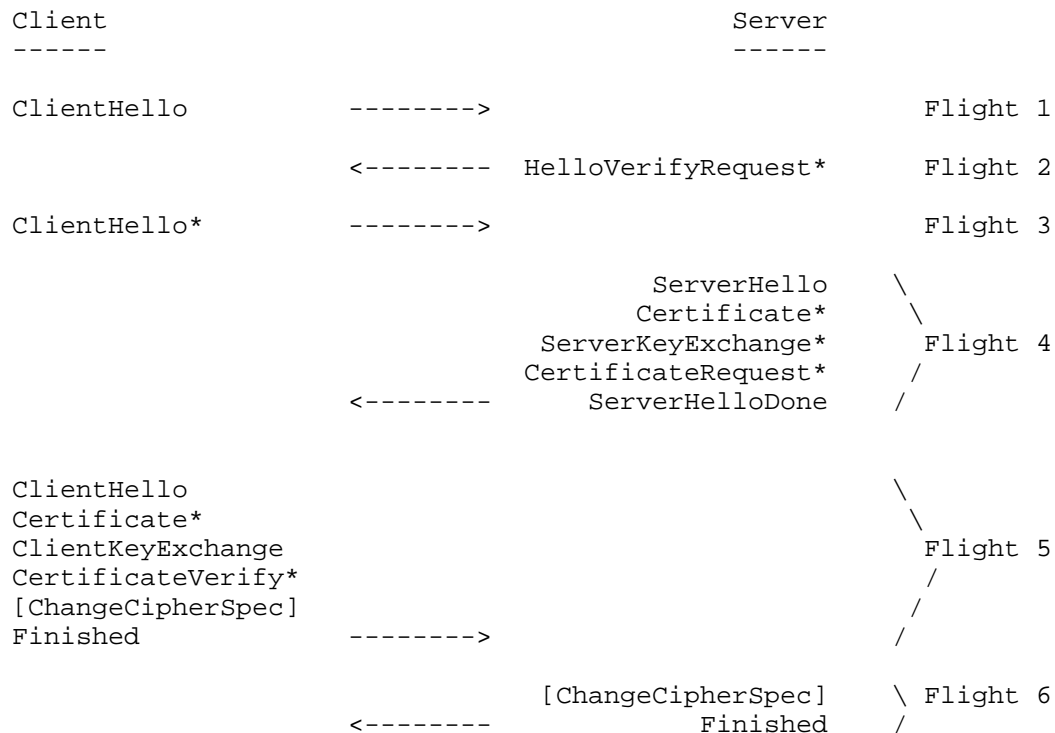


Figure 1: DTLS Handshake Protocol

[I-D.hartke-core-codtls] suggested to perform a RESTful DTLS handshake, which relies on CoAP Block-wise transfer [I-D.ietf-core-block]. A DTLS connection can be modeled as a CoAP resource. A DTLS resource is created when the client initiate a CoAP POST to a CoAP well-known URL. The state of the DTLS connection can be updated using a POST command. Figure 2 shows an example shows the use of CoAP Block-wise transfer to perform the DTLS handshake protocol. The client denotes the CoAP/DTLS client, and the server denotes the CoAP/DTLS server.

Using the example notation from [I-D.ietf-core-block], a Block option is shown in a decomposed way separating the kind of Block option (1

or 2) followed by a colon, and then the block number (NUM), more bit (M), and block size exponent ($2^{*(SZX+4)}$) separated by slashes. E.g., a Block2 Option value of 33 would be shown as 2:2/0/32, or a Block1 Option value of 59 would be shown as 1:3/1/128. In this example, a session id (/session/4ad6bc29) is used to identify the DTLS handshake session. Flight 2, 4 and 5 show the use of CoAP Block, where Flight 2 requires two blocks of data transfer, while Flight 4 and Flight 5 require eight blocks of data transfer each.

```

Client                                             Server
-----
CON [MID=1234] POST /.well-known/dtls
Payload:ClientHello                               <----->
                                                ACK [MID=1234], 2.01 Created
                                                <-----
CON [MID=4235], 2.01 Created, /session/4ad6bc29
                                                <----- Payload:HelloVerifyRequest
ACK [MID=4235], 0                               <----->
CON [MID=1235] POST /session/4ad6bc29, 1:0/1/64
Payload:ClientHello                               <----->
                                                ACK [MID=1235], 2.04 Changed, 1:0/1/64
                                                <-----
CON [MID=1236] POST /session/4ad6bc29, 1:1/0/64
Payload:ClientHello                               <----->
CON [MID=1236], 2.04 Changed, 2:0/1/64, 1:1/0/64
                                                <-----
CON [MID=4236] 2.04 Changed, 2:1/1/64
Payload:(ServerHello
Certificate*
ServerKeyExchange*
CertificateRequest*
                                                <-----
ServerHelloDone)
ACK [MID=4236], 0                               <----->
CON [MID=4237] 2.04 Changed, 2:2/1/64
Payload:(ServerHello
Certificate*
```

```

ServerKeyExchange*
CertificateRequest*
<-----
ServerHelloDone)
.
.
.
CON [MID=4243] 2.04 Changed, 2:7/0/64
Payload:(ServerHello
Certificate*
ServerKeyExchange*
CertificateRequest*
<-----
ServerHelloDone)

ACK [MID=4243], 0 ----->

CON [MID=1237] POST /session/4ad6bc29, 1:0/0/64
Payload:(Certificate*
ClientKeyExchange
CertificateVerify*
[ChangeCipherSpec]
Finished) ----->

ACK [MID=1237], 2.04 Changed, 1:0/0/64
<-----

CON [MID=1238] POST /session/4ad6bc29, 1:1/1/64
Payload:(Certificate*
ClientKeyExchange
CertificateVerify*
[ChangeCipherSpec]
Finished) ----->
.
.
.

CON [MID=1245] POST /session/4ad6bc29, 1:7/0/64
Payload:(Certificate*
ClientKeyExchange
CertificateVerify*
[ChangeCipherSpec]
Finished) ----->

ACK [MID=1245], 2.04 Changed, 1:7/0/64
<-----

CON [MID=4244] 2.04 Changed
Payload:([ChangeCipherSpec]
```

```

                                     <----- Finished)
ACK [MID=4244], 0                   ----->
```

Figure 2: DTLS Handshake Protocol using CoAP Block-wise Transfer

In this example, CoAP CON messages are used, hence the transfer of each block is acknowledged. Re-transmission is initiated if the ACK is not received. As CoAP protocol has already provided the functionality to acknowledge receipt of a message, the DTLS handshake protocol can exploit this reliability mechanism, so that similar level of message delivery guarantee with TLS can be achieved.

4 Next Steps

tbd

3 Security Considerations

tbd

4 IANA Considerations

tbd

5 References

5.1 Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.
- [RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, January 2012.

5.2 Informative References

- [I-D.ietf-core-coap]
Shelby, Z., Hartke, K., Bormann, C., and B. Frank,
"Constrained Application Protocol (CoAP)",
draft-ietf-core-coap-17 (work in progress), June 2013.
- [I-D.ietf-core-block]
Bormann, C. and Z. Shelby, "Blockwise transfers in CoAP",
draft-ietf-core-block-08 (work in progress),
February 2012.
- [I-D.ietf-lwig-guidance]
Bormann, C., "Guidance for Light-Weight Implementations
of the Internet Protocol Suite",
draft-ietf-lwig-guidance-01 (work in progress),
July 2012.
- [I-D.ietf-lwig-terminology]
Bormann, C., Ersue, M., and A. Keranen, "Terminology for
Constrained Node Networks",
draft-ietf-lwig-terminology-04 (work in progress),
April 2013.

- [I-D.ietf-tls-oob-pubkey]
Wouters, P., Gilmore, J., Weiler, S., Kivinen, T., and H. Tschofenig, "TLS Out-of-Band Public Key Validation", draft-ietf-tls-oob-pubkey-03 (work in progress), April 2012.
- [I-D.mcgrew-tls-aes-ccm]
McGrew, D. and D. Bailey, "AES-CCM Cipher Suites for TLS", draft-mcgrew-tls-aes-ccm-03 (work in progress), February 2012.
- [I-D.bormann-6lowpan-ghc]
Bormann, C., "6LoWPAN Generic Compression of Headers and Header-like Payloads", draft-bormann-6lowpan-ghc-04 (work in progress), March 2012.
- [I-D.keoh-lwig-dtls-iot]
Keoh, S., Kumar, S., and O. Garcia-Morchon, "Securing IP-based Internet of Things with DTLS", draft-keoh-lwig-dtls-iot-01 (work in progress), February 2013.
- [I-D.hartke-core-codtls]
Hartke, K., and O. Bergmann, "Datagram Transport Layer Security in Constrained Environment", draft-hartke-core-codtls-02 (work in progress), July 2012.
- [I-D.aks-crypto-sensors]
Sethi, M., Arkko J., Keranen A., and H. Rissanen, "Practical Considerations and Implementation Experiences in Securing Smart Object Networks", draft-aks-crypto-sensors-02 (work-in-progress), March 2012.
- [TINYDTLS] O. Bergmann, "TinyDTLS - A Basic DTLS Server Template", Accessed March 2013.
- [CONTIKI-DTLS]
K. Dominik Korte, "DTLS for Contiki", Jacobs University of Bremen, May 2010.
- [RFC5513] Farrel, A., "IANA Considerations for Three Letter Acronyms", RFC 5513, April 1 2009.
- [RFC5514] Vyncke, E., "IPv6 over Social Networks", RFC 5514, April 1

2009.

[RFC4492] Blake-Wilson, S., Bolyard, N., Gupta, V., Hawk, C., and B. Moeller, "Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS)", RFC 4492, May 2006.

[RFC6066] Eastlake 3rd, D., "Transport Layer Security (TLS) Extensions: Extension Definitions", RFC 6066, January 2011.

Authors' Addresses

Sye Loong Keoh
Philips Research Europe
High Tech Campus 34,
5656 AE, Eindhoven,
The Netherlands

Email: sye.loong.keoh@philips.com

Sandeep S. Kumar
Philips Research Europe
High Tech Campus 34,
5656 AE, Eindhoven,
The Netherlands

Email: sandeep.kumar@philips.com

Zach Shelby
Sensinode
Kidekuja 2
Vuokatti 88600
Finland

Email: zach@sensinode.com

DICE Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 24, 2014

S. Kumar
Philips Research
S. Keoh
University of Glasgow Singapore
O. Garcia-Morchon
Philips Research
October 21, 2013

DTLS Relay for Constrained Environments
draft-kumar-dice-dtls-relay-00

Abstract

The 6LoWPAN and CoAP standards defined for resource-constrained devices are fast emerging as the de-facto protocols for enabling the Internet-of-Things (IoTs). Security is an important concern in IoTs and the DTLS protocol has been chosen as the preferred method for securing CoAP messages. DTLS is a point-to-point protocol relying on the IP routing to deliver messages between the client and the server. However in some low-power lossy networks (LLNs) with multi-hop, a new "joining" device may not be initially IP routable. This prevents DTLS from being directly useful as an authentication and confidentiality protocol during this stage, requiring other security protocols to be implemented on the device. These devices being resource-constrained often cannot accommodate more than one security protocol in their code memory. To overcome this problem and reuse DTLS, we present a DTLS Relay solution for the non-IP routable "joining" device to establish a secure DTLS connection with a DTLS server. Further we present a stateful and stateless mode of operation for the DTLS Relay.

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 <http://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 April 24, 2014.

Copyright Notice

Copyright (c) 2013 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 (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
2. Use Case	3
3. DTLS relay	5
3.1. Relay in Stateful mode	5
3.2. Relay in Stateless mode	7
3.3. Comparison between the two modes	9
4. IANA Considerations	9
5. Security Considerations	9
6. Acknowledgements	10
7. References	10
7.1. Normative References	10
7.2. Informative References	10
Authors' Addresses	11

1. Introduction

The Internet-of-Things (IoTs) vision is more closer to reality with the IPv6 over Low-Power Wireless Personal Area Networks (6LoWPAN) [RFC4944] and Constrained Application Protocol (CoAP) [I-D.ietf-core-coap] standards. The 6LoWPAN adaptation layer allows for transmission of IPv6 Packets over IEEE 802.15.4 networks [ieee802.15.4] and thereby enabling end-to-end IPv6 connectivity of "Things". CoAP is a web protocol based on REST architecture designed to work under the special requirements of the constrained environment. It supports binding to UDP [RFC0768] which is preferred over TCP [RFC0793] in low-power lossy networks (LLNs) such as IEEE 802.15.4.

Security is important concern in such a wireless multi-hop network that could be used in various application domains such as smart energy and building automation. However security protocols are often

heavy-weight both in terms of code and network processing. Due to the constrained nature of most of these devices, multiple security protocols for different purposes and at different networking layers are hard to envision. It is more efficient to use a single security protocol to fulfill multiple security requirements in such constrained environments.

CoAP has chosen Datagram Transport Layer Security (DTLS) [RFC6347] as the preferred security protocol for authenticity and confidentiality of the messages. It is based on Transport Layer Security (TLS) [RFC5246] with modifications to run over UDP. DTLS makes use of additional reliability mechanisms in its handshake due to the lack of TCP reliable transmission mechanisms that are available to TLS.

DTLS is a point-to-point protocol relying on the underlying IP layer to perform the routing between the DTLS client and the DTLS server. However in some LLNs with multi-hop, a new "joining" device may not be initially IP routable. A new "joining" device can only initially use a link-local IPv6 address to communicate with a neighbour node using neighbour discovery [RFC6775] until it receives the necessary network configuration parameters. Before the device can receive these configuration parameters, it may need to authenticate itself or wish to authenticate the network to which it connects. DTLS although a suitable protocol for such authentication and secure transfer of configuration parameters, would not work due to the lack of IP routability of its messages to the intended recipients.

We present a DTLS Relay solution to overcome this problem for the "joining" device to establish a secure DTLS connection with a DTLS server. This draft is inspired by the Protocol for carrying Authentication for Network Access (PANA) Relay Element [RFC6345] which is intended to solve a similar problem when PANA [RFC5191] is used for network access. Further we present a stateful and stateless mode of operation for the DTLS Relay.

This draft is an early description of the solutions and does not provide the complete details yet. This draft is structured as follows: we present a use-case for the DTLS Relay in Section 2, then present the DTLS Relay solution in Section 3 for stateful and stateless mode of operation. Further we present some security considerations in Section 5.

2. Use Case

We present here a target usecase based on [I-D.jennings-core-transitive-trust-enrollment] describing a rendezvous protocol that allows a constrained IoT device to securely connect into a system or network. The main idea is that the joining

Device has a pre-established trust relationship with a "Transfer Agent" entity, for e.g. Pre-Shared Keys provisioned during manufacturing. This "Transfer Agent" provides the needed trust credentials to the Device and/or a Controller in the system to establish a secured connection to perform further authentication and transfer of system/network configuration parameters. This step is enabled by an "Introducer" entity which informs the "Transfer Agent" about the details of Controller to which the joining Device should connect, and provide to the Controller the identity including one-time credentials for enable secure connection to the Device. The transitive trust establishment procedure is explained in detail in [I-D.jennings-core-transitive-trust-enrollment] and we focus here on how to enable this using DTLS.

As depicted in the Figure 1, the joining Device (D) is multi-hop away from the Controller (C) and not yet authenticated into the network. At this stage, it can only communicate one-hop to its nearest neighbour (N) using their link-local IPv6 addresses. However, the Device needs to communicate with end-to-end security with a Transfer Agent (T) or to Controller (C) to authenticate and get the relevant system/network parameters. If the Device (D), initiates a DTLS connection to the Transfer Agent that has been pre-configured, then the packets are dropped at the neighbour (N) since the Device (D) is not yet admitted to the network or there is no IP-routability to Device (D) for any returned messages.

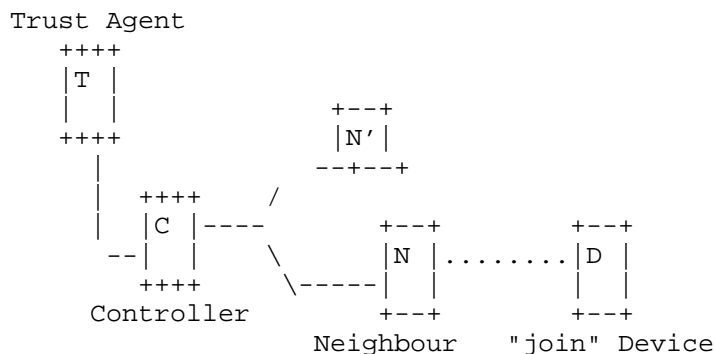


Figure 1: Use case depiction in a multi-hop network

Further the Device (D) may wish to establish a secure connection to the Controller (C) in the network assuming credentials are exchanged out-of-band, for e.g. a hash of the Device (D)'s raw public key could be provided to the Controller (C). However, the Device (D) is

unaware of the IP address of the Controller (C) to initiate a DTLS connection and perform authentication.

To overcome these problems with non-routability of DTLS packets and/or discovery of the destination address of the DTLS server to contact, we define a DTLS Relay solution. This DTLS Relay ability is configured into all authenticated devices in the network which may act as the Neighbour (N) device for newly joining nodes. The DTLS Relay allows for relaying of the packets from the Neighbour (N) using IP-routing to the intended DTLS server. Further, if the DTLS server address is not known to the joining Device (D), then messages are delivered to a pre-configured DTLS server address (mostly the Controller (C)) known to the DTLS Relay.

3. DTLS relay

In this section, we describe how the DTLS Relay functionality can be achieved. When a joining device as a client attempts a DTLS connection (for example to a "Transfer Agent"), it uses its link-local IP address as its IP source address. This message is transmitted one-hop to a neighbour node. Under normal circumstances, this message would be dropped at the neighbour node since the joining device is not yet IP routable, or it is not yet authenticated to send messages through the network. However, if the neighbour device has the DTLS Relay functionality enabled, it forwards DTLS messages to specific servers. Additional security mechanisms need to exist to prevent this forwarding functionality to be used by rogue nodes to bypass any network authentication procedures and are discussed in Section 5.

The DTLS Relay can operate in two different modes: stateful and stateless. We present here both the methods, however for interoperability, only one of the modes should be mandated. Within each mode, the DTLS Relay can further forward packets based on the client defined DTLS server address or a DTLS server address that has been configured into the Relay.

3.1. Relay in Stateful mode

The neighbour node on receiving a DTLS message from a joining device enters into DTLS Relay mode. In this mode, the neighbour node has the additional functionality to send DTLS messages further to the end-point DTLS server the joining device wishes to contact. In the stateful mode of operation, the message is transmitted to the end-point as originating from the DTLS Relay by replacing the IP address and port to DTLS Relay's own IP address and a randomly chosen port. The DTLS message itself is not modified.

Additionally, the DTLS Relay must track the ongoing DTLS connections based on the following 4-tuple stored locally:

- o Client source IP address (IP_C)
- o Client source port (p_C)
- o DTLS Server IP address (IP_S)
- o Relay source port (p_R)

The DTLS server communicates to the Relay as if it were communicating to the end-point Client with no modification required to the DTLS messages. The Relay on receiving this message, looks up its locally stored 4-tuple array to identify to which joining device (if multiple exists) the message belongs. The DTLS message's destination address is replaced with that of the link-local address and port of the joining device from the lookup array and forwarded to it. The Relay does not modify the DTLS packets and therefore the normal processing and security of DTLS is unaffected.

The following message flow diagram indicates the various steps of the process where the DTLS server address is known to the joining device:

DTLS Client (C)	DTLS Relay (R)	DTLS Server (S)	Message	
			Src_IP:port	Dst_IP:port
--ClientHello-->			IP_C:p_C	IP_S:5684
	--ClientHello-->		IP_R:p_R	IP_S:5684
		<--ServerHello--	IP_S:5684	IP_R:p_R
		:		
	<--ServerHello--		IP_S:5684	IP_C:p_C
	:			
	::		:	:
	::		:	:
--Finished-->			IP_C:p_C	IP_S:5684
	--Finished-->		IP_R:p_R	IP_S:5684
		<--Finished--	IP_S:5684	IP_R:p_R
	<--Finished--		IP_S:5684	IP_C:p_C
	::		:	:

IP_C:p_C = IP (non-routable) and port of Client

IP_S:5684 = IP and coaps port of Server

IP_R:p_R = IP and port of Relay

Figure 2: Message flow in Stateful mode with DTLS Server defined by Client

In the situation where the joining device is unaware of the IP address of DTLS server it needs to contact, for e.g. the Controller of the network, the DTLS Relay can be configured with IP destination of the default DTLS server that a joining device needs to contact. The joining device initiates its DTLS request as if the DTLS Relay is the intended end-point DTLS server. The DTLS relay translates the DTLS message as in the previous case by modifying both the source and destination IP address to forward the message to the intended DTLS server. The Relay keeps a similar 4-tuple array to enable translation of the DTLS messages received from the server and forward it to the DTLS Client. The following message flow indicates this process:

DTLS Client (C)	DTLS Relay (R)	DTLS Server (S)	Message	
			Src_IP:port	Dst_IP:port
--ClientHello-->			IP_C:p_C	IP_Ra:5684
	--ClientHello-->		IP_Rb:p_Rb	IP_S:5684
		<--ServerHello--	IP_S:5684	IP_Rb:p_Rb
		:		
	<--ServerHello--		IP_Ra:5684	IP_C:p_C
	:			
	::		:	:
	::		:	:
--Finished-->			IP_C:p_C	IP_Ra:5684
	--Finished-->		IP_Rb:p_Rb	IP_S:5684
		<--Finished--	IP_S:5684	IP_Rb:p_Rb
	<--Finished--		IP_Ra:5684	IP_C:p_C
	::		:	:

IP_C:p_C = IP (non-routable) and port of Client

IP_S:5684 = IP and coaps port of Server

IP_Ra:5684 = IP and coaps port of Relay

IP_Rb:p_Rb = IP (can be same as IP_Ra) and the port of Relay

Figure 3: Message flow in Stateful mode with DTLS Server defined by Relay

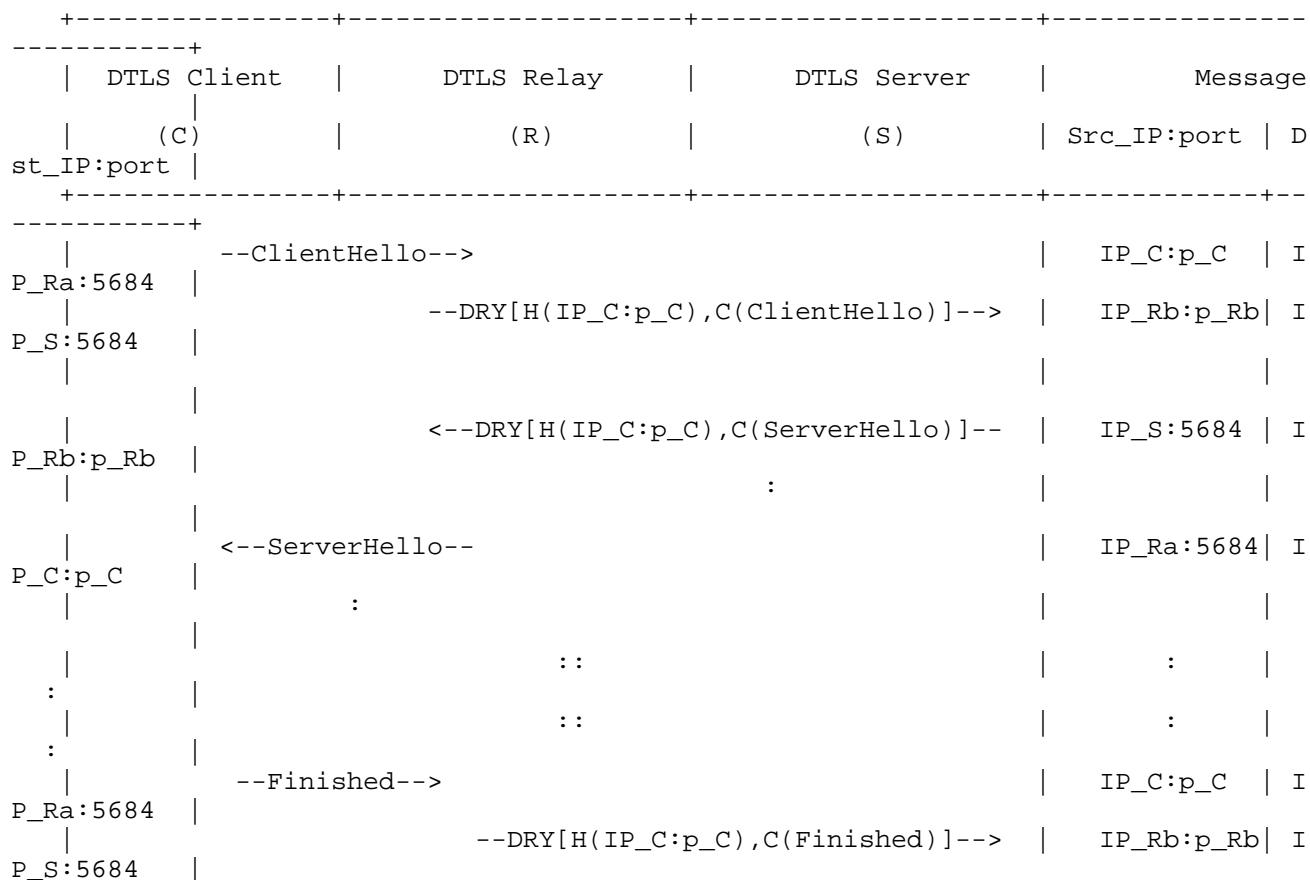
3.2. Relay in Stateless mode

In the alternative mode of operation for the DTLS Relay, a stateless approach is applied where the Relay does not need to store a local 4-tuple array. Just as in the previous case, if a DTLS client with only link local addressing wants to contact a trusted end-point DTLS server, it sends the DTLS message to the Relay. The Relay instead of translating, encapsulates this message into a new type of message called DTLS Relay (DRY) message. The DRY consists of two parts:

- o Header (H) field: consisting of the source link-local address and port of the DTLS Client device, and
- o Contents (C) field: containing the original DTLS message.

The DTLS end server on receiving the DRY message, decapsulates it to retrieve the two parts. It then uses the Header field information to associate the new state created on the server for the DTLS connection to the DTLS client's address and port. The DTLS server then performs the normal DTLS operations on the DTLS message contents. However when the DTLS server replies, it also encapsulates its message in a DRY message back to the Relay with the Header containing the original source link-local address and port of the DTLS Client. The Relay can decapsulate the DRY message, retrieves the Header information to forward this message to the right DTLS Client device.

The following figure depicts the message flow diagram when the DTLS server end-point address is known only to the Relay:



P_Rb:p_Rb		<--DRY[H(IP_C:p_C),C(Finished)]--		IP_S:5684 I
P_C:p_C		<--Finished--		IP_Ra:5684 I
:		::		:
+	-----+		+	-----+
-----+				

IP_C:p_C = IP (non-routable) and port of Client
IP_S:5684 = IP and coaps port of Server
IP_Ra:5684 = IP and coaps port of Relay
IP_Rb:p_Rb = IP (can be same as IP_Ra) and the port of Relay

DRY[H()],C()] = DTLS Relay message with header H and content C

Figure 4: Message flow in Stateless mode with DTLS Server defined by Relay

The message flow for the case in which the DTLS Client is aware of the end-point DTLS server's address is similar and not described further. It can be derived based on Figure 2 and Figure 4.

3.3. Comparison between the two modes

A comparison between the two modes will be done in an updated version of this draft.

4. IANA Considerations

tbd

Note to RFC Editor: this section may be removed on publication as an RFC.

5. Security Considerations

Additional security considerations need to be taken into account about forwarding of messages from devices through a network to which it has not yet been admitted. This can lead to denial-of-service attacks or misuse of network resources without proper authentication. One way to overcome any large scale misuse of the network is to have a management message from the Controller that initiates already authenticated devices in the network to enter into a DTLS Relay mode. The devices can stay such a Relay mode for a fixed period of time or until the Controller sends a new management message blocking the DTLS Relay mode in all devices in the network. This is often possible since the administrator of the network can be aware when new devices join the network either because of the "Introduction" phase or commissioning phase.

Other mechanisms based on IP destination filtering can be applied by the controller to all Relay nodes to avoid misuse of the network resources.

6. Acknowledgements

The authors would like to thank Sahil Sharma, Ernest Ma, Dee Denteneer and Peter Lenoir for the valuable discussions and suggestions,

7. References

7.1. Normative References

- [RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, January 2012.

7.2. Informative References

- [I-D.ietf-core-coap]
Shelby, Z., Hartke, K., and C. Bormann, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-18 (work in progress), June 2013.
- [I-D.jennings-core-transitive-trust-enrollment]
Jennings, C., "Transitive Trust Enrollment for Constrained Devices", draft-jennings-core-transitive-trust-enrollment-01 (work in progress), October 2012.
- [RFC0768] Postel, J., "User Datagram Protocol", STD 6, RFC 768, August 1980.
- [RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981.
- [RFC4944] Montenegro, G., Kushalnagar, N., Hui, J., and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks", RFC 4944, September 2007.
- [RFC5191] Forsberg, D., Ohba, Y., Patil, B., Tschofenig, H., and A. Yegin, "Protocol for Carrying Authentication for Network Access (PANA)", RFC 5191, May 2008.
- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.
- [RFC6345] Duffy, P., Chakrabarti, S., Cragie, R., Ohba, Y., and A. Yegin, "Protocol for Carrying Authentication for Network Access (PANA) Relay Element", RFC 6345, August 2011.
- [RFC6775] Shelby, Z., Chakrabarti, S., Nordmark, E., and C. Bormann, "Neighbor Discovery Optimization for IPv6 over Low-Power

Wireless Personal Area Networks (6LoWPANs)", RFC 6775,
November 2012.

[ieee802.15.4]
IEEE Computer Society, ., "IEEE Std. 802.15.4-2003",
October 2003.

Authors' Addresses

Sandeep S. Kumar
Philips Research
High Tech Campus 34
Eindhoven 5656 AE
NL

Email: sandeep.kumar@philips.com

Sye Loong Keoh
University of Glasgow Singapore
Republic PolyTechnic, 9 Woodlands Ave 9
Singapore 838964
SG

Email: SyeLoong.Keoh@glasgow.ac.uk

Oscar Garcia-Morchon
Philips Research
High Tech Campus 34
Eindhoven 5656 AE
NL

Email: oscar.garcia@philips.com