CoRE Working Group
Internet-Draft
Intended status: Informational
Expires: January 17, 2013

K. Hartke
O. Bergmann
Universitaet Bremen TZI
July 16, 2012

Datagram Transport Layer Security in Constrained Environments
draft-hartke-core-codtls-02

Abstract

This draft considers some obstacles in implementing Datagram Transport Layer Security (DTLS) in constrained environments, and presents some ideas for a constrained version of DTLS that is friendly to constrained nodes and networks.

Table of Contents

1. Introduction

1.1. Background

Nodes that take part in the "Internet of Things" often have strict limitations regarding their computational power, memory size (both ROM and RAM), and power management [I-D.ietf-lwig-guidance]. Network communication, especially wireless, also imposes constraints that need to be considered during protocol design, e.g. low bitrate, variable delay 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 mapping 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: E.g., 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 DTLS has not been designed with constrained nodes/networks in mind, it is thought to be usable in such environments [SOS12]. Still, there are a few challenges when it comes to implement DTLS.

1.2. Overview

The present document considers some obstacles in implementing DTLS in constrained environments, and presents a few ideas to make DTLS 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 MAYs into MUSTs or MUST NOTs, or by requiring or precluding certain extensions or cipher suites. Existing DTLS implementations ought to continue to be used 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/decompression.

Breaking changes: New implementations are required that do not interoperate with implementations of DTLS.

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.)

The term "byte" is used in its now customary sense as a synonym for "octet".

2. Potential Problems and Possible Solutions

2.1. Handshake Message Fragmentation

DTLS records can be large in size for a single 6LoWPAN [RFC4944] payload: [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 doesn't really work well for constrained applications and networks.

DTLS offers fragmentation at the handshake layer and hence can get around IP fragmentation. However, this can add a significant overhead on the number of datagrams and bytes transferred (see Table 1). Packet loss is also still a big problem for the constrained nodes; 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 exchange 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:

- Use IP fragmentation instead of DTLS 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 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. A recipient must still be prepared to receive arbitrarily fragmented handshake messages at the DTLS layer, though.

- Reduce the number of bytes to be transferred, so the overhead of header data becomes smaller when fragmenting for small packet sizes, and fewer packets need to be transmitted that could potentially be lost:

  * Use an out-of-band mechanism to exchange large blobs. For example, the TLS Cached Information Extension [I-D.ietf-tls-cached-info] allows to omit the exchange of fairly static information, such as the server certificate, if this information is already available.

  * Use 6LoWPAN General Header Compression [I-D.bormann-6lowpan-ghc] to compress DTLS messages, as proposed in [DCOSS12].

* Use some DTLS-specific kind of Stateless Header Compression, as shown in Section 3. This can significantly reduce the number of datagrams and bytes transferred, and in particular also the proportion of header data in the number of bytes transferred (see Table 2).

* Use compressed point formats for elliptic curve points.

* Use self-delimiting numeric values [RFC6256] instead of fixed-size numeric values.

* Use a bit field instead of multiple type fields to indicate which handshake messages are present in a datagram.

o Perform the DTLS handshake over another protocol, for example, CoAP [I-D.ietf-core-coap] with its support for block-wise transfers [I-D.ietf-core-block], as shown in Section 4.

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 but using Stateless Header Compression (Section 4)

2.2. Timer Values

DTLS 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 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:

- Adjust the timer value to meet the conditions of constrained nodes and low-power, lossy networks.

- Add some kind of acknowledgment message to DTLS that allows an implementation to confirm the receipt of a message before preparing the next message flight.

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:

- 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 do 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.)

- 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).

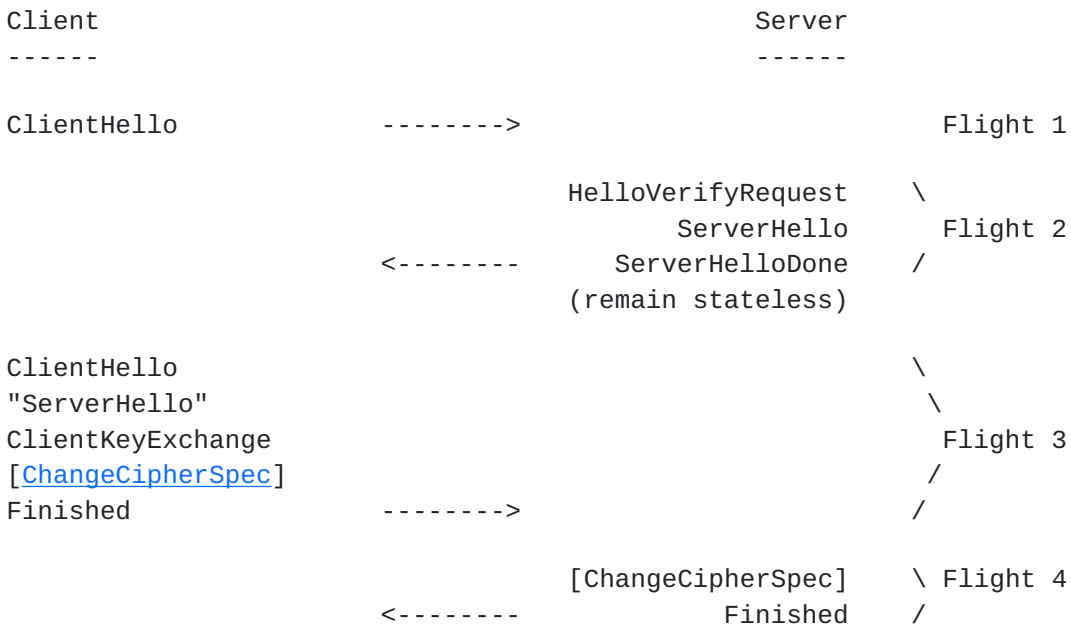


Figure 1: Artist's impression of a four-flight DTLS handshake with Pre-Shared Key

- 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 a less constrained device.

2.4. Connection Closure

Although a connection needs considerably less memory after a handshake has finished, it still requires, e.g., around 80 bytes with AES-128-CCM [I-D.mcgrew-tls-aes-ccm] for the keys, sequence numbers and anti-replay window. More memory is needed if session resumption is supported, to keep 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:

- 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.

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

2.5. 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 within a single IP packet.

DTLS has a per-record overhead of 13 bytes for the record header. AEAD ciphers such as AES-CCM [I-D.mcgrew-tls-aes-ccm] 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).

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:

- 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.

- 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.

- 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 3 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).

2.6. 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.

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:

- 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.

- Favour cryptographic algorithms that use less memory, possibly resulting in a slower performance.

- Add some kind of acknowledgment message to DTLS that allows an receiver to confirm the receipt of a message, and let the sender wait for the acknowledgment before it sends the next part of the flight.

2.7. 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:

- 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.

- Use using pre-composed messages instead of writing code, e.g., for encoding or decoding ASN.1 structures, as shown in Appendix A.

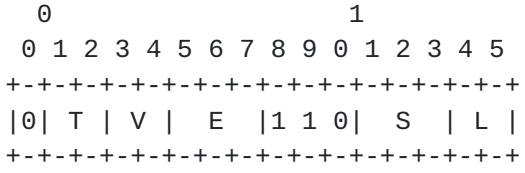
3. Stateless Header Compression

Stateless Header Compression compresses the headers of 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.

3.1. 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:



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.

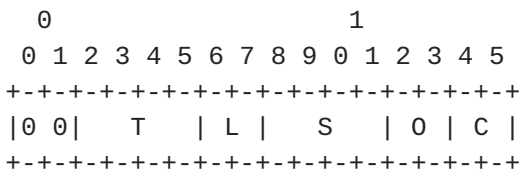
0 - Sequence number 0
1 - 8-bit sequence_number field
2 - 16-bit sequence_number field
3 - 24-bit sequence_number field
4 - 32-bit sequence_number field
5 - 40-bit sequence_number field
6 - 48-bit sequence_number field
7 - Implicit -- number of previous record in the datagram + 1

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

3.2. 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:



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

4. RESTful DTLS Handshake

Where DTLS is used in conjunct with the Constrained Application Protocol (CoAP) [I-D.ietf-core-coap], it might be beneficial to use CoAP with its support for block-wise transfers [I-D.ietf-core-block] instead of DTLS's convoluted handshake protocol to transport DTLS handshake messages.

CoAP, like HTTP, is designed for applications following the REST architectural style [REST]. So the DTLS connection is modeled as a CoAP resource which gets created when a client wants to initiate a connection, and gets updated to modify the state and parameters of the connection. A well-known URI path [RFC5785] is used to identify a collection resource that models the set of active connections and allows new connections to be created.

Figure 2: Message Flights for Full Handshake

Figure 3: Message Flights for Session-Resuming Handshake

There are the following possible operations:

- POST to well-known URI: requests the server to create a new session resource.
- PATCH session resource: requests the server to change session parameters, or to resume a session.
- GET session resource: returns a representation of the session.
- DELETE session resource: requests the server to delete the session resource and free all resources related to the session.

The following protocols and URI schemes are used:

- CoAP [I-D.ietf-core-coap]
- coap+codtls://

The following well-known URIs are used:

- /.well-known/dtls

The following media types are used:

- application/dtls

(The exact definition of these items is TBD.)

5. Security Considerations

Beyond stateless header compression and profiling, 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

Thanks to Angelo P. Castellani, Stefan Jucker and Shahid Raza 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-04 (work in progress), March 2012.

[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-core-coap]
Shelby, Z., Hartke, K., Bormann, C., and B. Frank, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-10 (work in progress), June 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-tls-cached-info]
Santesson, S. and H. Tschofenig, "Transport Layer Security (TLS) Cached Information Extension", draft-ietf-tls-cached-info-11 (work in progress), December 2011.

[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.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-02 (work in progress), October 2011.

[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>.

[REST] Fielding, R., "Architectural Styles and the Design of Network-based Software Architectures", Ph.D. Dissertation, University of California, Irvine, 2000, <http://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf>.

[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.

[RFC4944] Montenegro, G., Kushalnagar, N., Hui, J., and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks", RFC 4944, September 2007.

[RFC5785] Nottingham, M. and E. Hammer-Lahav, "Defining Well-Known Uniform Resource Identifiers (URIs)", RFC 5785, April 2010.

[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.

[SOS12] Arkko, J. and H. Tschofenig, "Conclusions from the Workshop on Smart Object Security", Workshop on Smart Object Security, March 2012, <http://www.lix.polytechnique.fr/hipercom/SmartObjectSecurity/slides/sos-conclusions.ppt>.

[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 bodies of the Certificate, ServerKeyExchange and ClientKeyExchange messages used in a DTLS handshake with raw public key certificates [I-D.ietf-tls-oob-pubkey] and the ECDHE_ECDSA key exchange [RFC4492].

The templates are given for the named curves secp256r1, secp384r1 and secp521r1; these curves are equivalent to the NIST P-256, P-384, and P-521 curves. They are required in [I-D.mcgrew-tls-aes-ccm-ecc]. The same curve is used in each case for both the raw public key certificate and the ephemeral keys. Points are represented in uncompressed point format.

Note: The templates have not been independently verified yet.

A.1. secp256r1

Raw Public Key Certificate:

```
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 __ __ __ __ __
__ __ __ __ __ __ __ __  __ __ __ __ __ __ __ __
__ __ __ __ __ __ __ __  __ __ __ __ __ __ __ __
__ __ __ __ __ __ __ __  __ __ __ __ __ __ __ __
__ __ __ __ __ __ __ __  __ __ __
```

ECDSA-capable public key (x, y)

Server Key Exchange:

```
03 00 17 41 04 __ __ __  __ __ __ __ __ __ __ __
__ __ __ __ __ __ __ __  __ __ __ __ __ __ __ __
__ __ __ __ __ __ __ __  __ __ __ __ __ __ __ __
__ __ __ __ __ __ __ __  __ __ __ __ __ __ __ __
__ __ __ __ __ 00 46 30  44 02 20 __ __ __ __ __
__ __ __ __ __ __ __ __  __ __ __ __ __ __ __ __
__ __ __ __ __ __ __ __  __ __ __ 02 20 __ __ __
__ __ __ __ __ __ __ __  __ __ __ __ __ __ __ __
__ __ __ __ __ __ __ __  __ __ __ __ __
```

ephemeral ECDH public key (x, y) and ECDSA signature (r, s)

Client Key Exchange:

```
41 04 __ __ __ __ __ __  __ __ __ __ __ __ __ __
__ __ __ __ __ __ __ __  __ __ __ __ __ __ __ __
__ __ __ __ __ __ __ __  __ __ __ __ __ __ __ __
__ __ __ __ __ __ __ __  __ __ __ __ __ __ __ __
__ __
```

ephemeral ECDH public key (x, y)

A.2. secp384r1

Raw Public Key Certificate:

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

ECDSA-capable public key (x, y)

Server Key Exchange:

```
03 00 18 61 04 __ __ __  __ __ __ __ __ __ __ __
__ __ __ __ __ __ __ __  __ __ __ __ __ __ __ __
__ __ __ __ __ __ __ __  __ __ __ __ __ __ __ __
__ __ __ __ __ __ __ __  __ __ __ __ __ __ __ __
__ __ __ __ __ __ __ __  __ __ __ __ __ __ __ __
__ __ __ __ __ __ __ __  __ __ __ __ __ __ __ __
__ __ __ __ __ 00 66 30  64 02 30 __ __ __ __ __
__ __ __ __ __ __ __ __  __ __ __ __ __ __ __ __
__ __ __ __ __ __ __ __  __ __ __ __ __ __ __ __
__ __ __ __ __ __ __ __  __ __ __ 02 30 __ __ __
__ __ __ __ __ __ __ __  __ __ __ __ __ __ __ __
__ __ __ __ __ __ __ __  __ __ __ __ __ __ __ __
__ __ __ __ __ __ __ __  __ __ __ __ __
```

ephemeral ECDH public key (x, y) and ECDSA signature (r, s)

Client Key Exchange:

```
61 04 __ __ __ __ __ __  __ __ __ __ __ __ __ __
__ __ __ __ __ __ __ __  __ __ __ __ __ __ __ __
__ __ __ __ __ __ __ __  __ __ __ __ __ __ __ __
__ __ __ __ __ __ __ __  __ __ __ __ __ __ __ __
__ __ __ __ __ __ __ __  __ __ __ __ __ __ __ __
__ __ __ __ __ __ __ __  __ __ __ __ __ __ __ __
__ __
```

ephemeral ECDH public key (x, y)

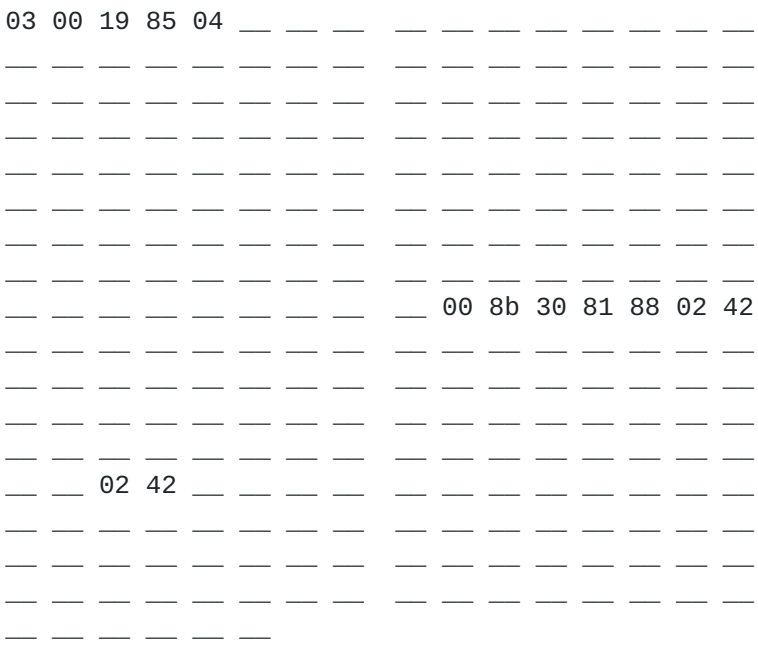
A.3. secp521r1

Raw Public Key Certificate:

```
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 __ __ __ __ __ __
__ __ __ __ __ __ __ __  __ __ __ __ __ __ __ __
__ __ __ __ __ __ __ __  __ __ __ __ __ __ __ __
__ __ __ __ __ __ __ __  __ __ __ __ __ __ __ __
__ __ __ __ __ __ __ __  __ __ __ __ __ __ __ __
__ __ __ __ __ __ __ __  __ __ __ __ __ __ __ __
__ __ __ __ __ __ __ __  __ __ __ __ __ __ __ __
__ __ __ __ __ __ __ __  __ __ __ __ __ __
```

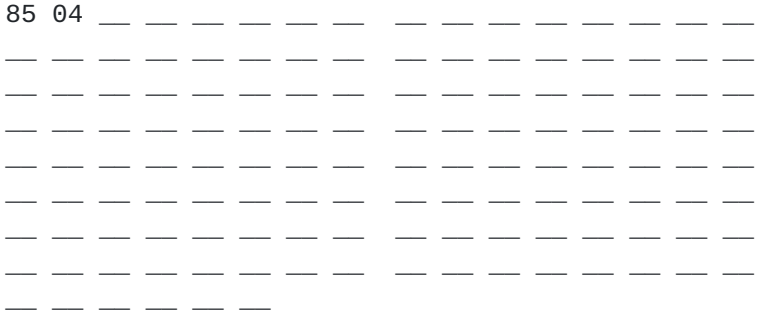
ECDSA-capable public key (x, y)

Server Key Exchange:



ephemeral ECDH public key (x, y) and ECDSA signature (r, s)

Client Key Exchange:



ephemeral ECDH public key (x, y)

Authors' Addresses

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

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

Olaf Bergmann
Universitaet Bremen TZI
Postfach 330440
Bremen D-28359
Germany

Phone: +49-421-218-63904
Email: bergmann@tzi.org