Internet Draft
Intended Status: Informational
Expires: May 9, 2014

S. Keoh
University of Glasgow
S.S. Kumar
Philips Research
Z. Shelby
Sensinode

November 5, 2013

Profiling of DTLS for CoAP-based IoT Applications
draft-keoh-dice-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. Specially, this document investigate 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.

Table of Contents

1 Introduction 3
1.1 Terminology 3
2 Implementation Issues of DTLS for IoT Applications 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.

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 for IoT Applications

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 enables 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" do not need to be implemented and supported. This is especially useful for Raw Public-key and/or Certificates mode of operations.

b. The RESUME protocol is useful for IoT applications. At 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 should 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.mcgrew-tls-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 transitions 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 to 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 he 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 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.

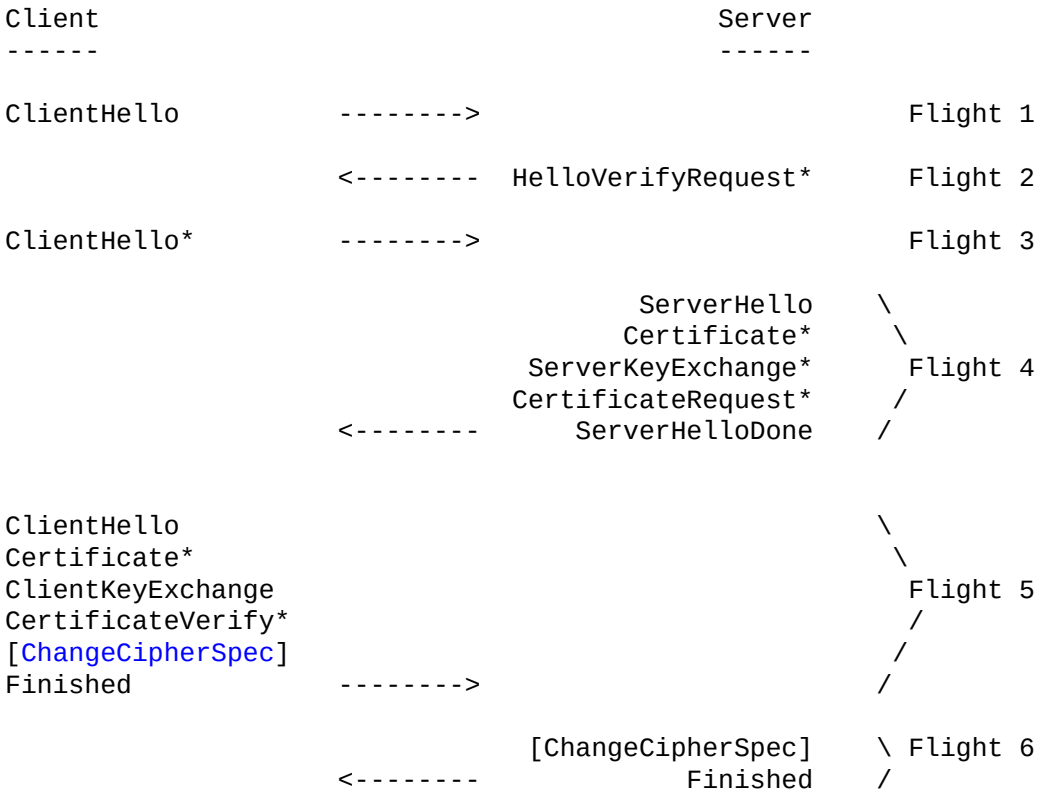
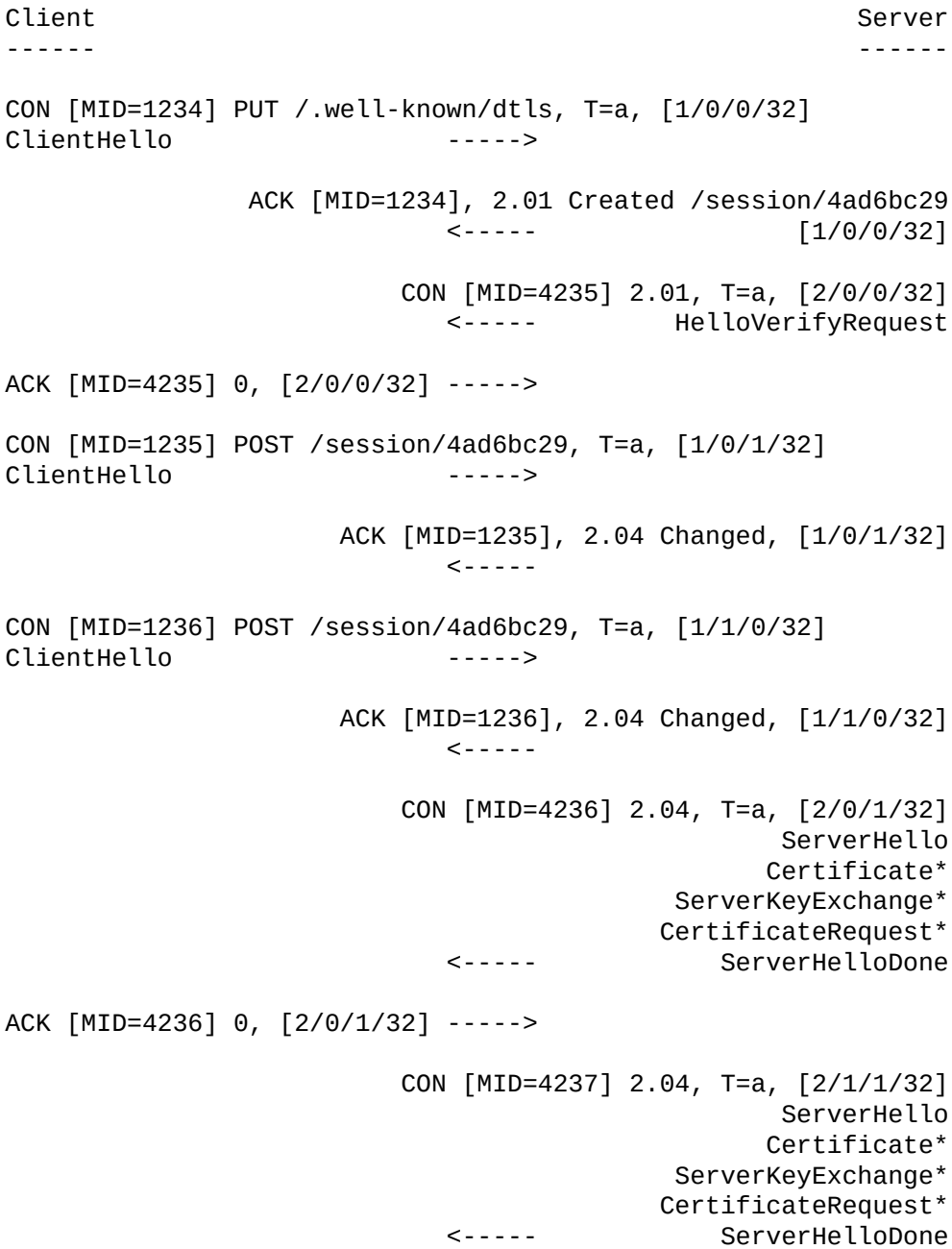


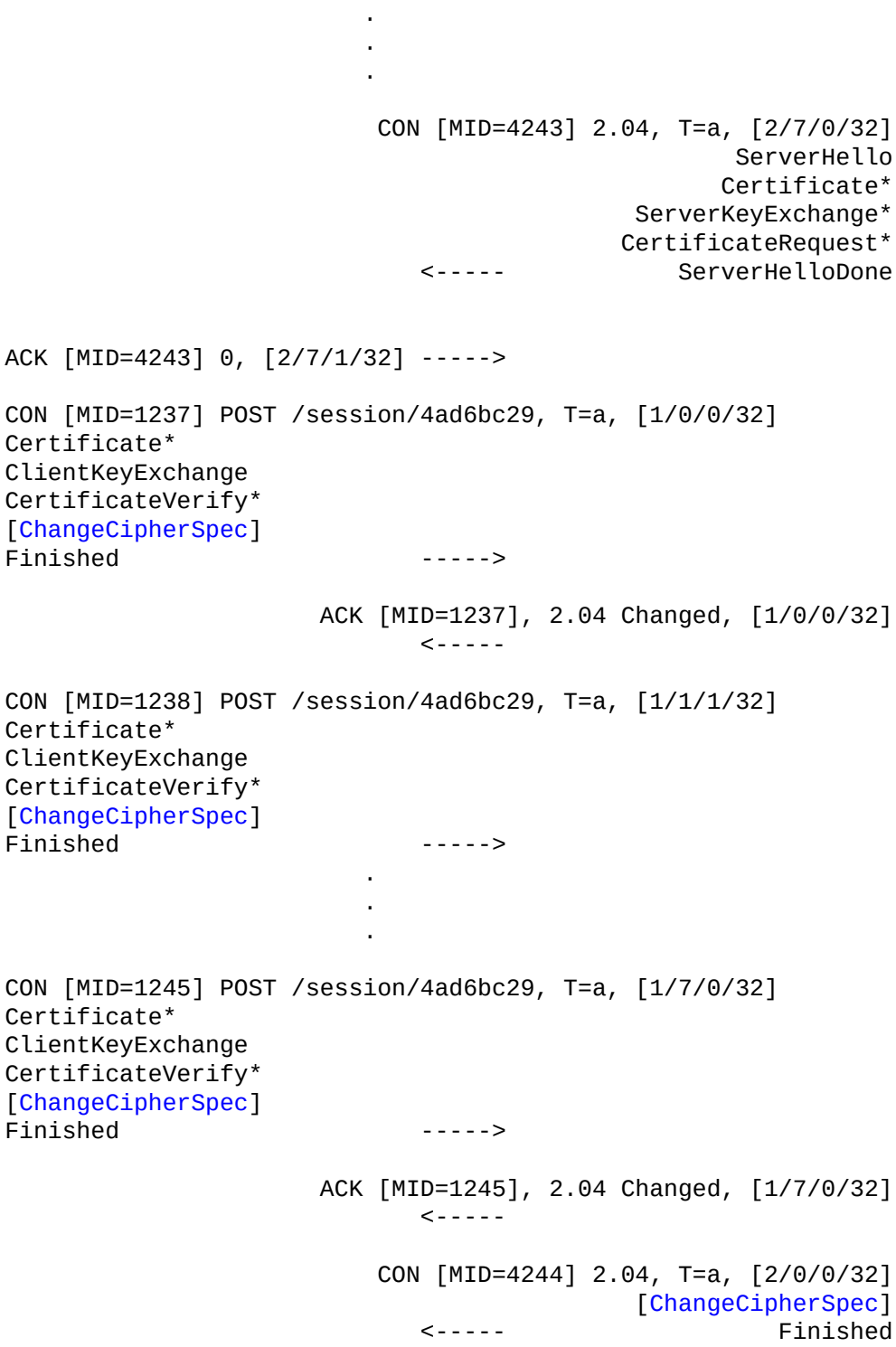
Figure 1: DTLS Handshake Protocol

[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 PUT or POST. Figure 2 shows an example shows the use of CoAP Block-wise transfer to perform the DTLS handshake protocol.

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), block number (NUM), more bit (M), and block size exponent (2**(SZX+4) 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 token option (e.g., T=a) and a session id (/session/4ad6bc29) are 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.




.
.
.
CON [MID=4243] 2.04, T=a, [2/7/0/32]
ServerHello
Certificate*
ServerKeyExchange*
CertificateRequest*
<----- ServerHelloDone
ACK [MID=4243] 0, [2/7/1/32] ----->
CON [MID=1237] POST /session/4ad6bc29, T=a, [1/0/0/32]
Certificate*
ClientKeyExchange
CertificateVerify*
[ChangeCipherSpec]
Finished ----->
ACK [MID=1237], 2.04 Changed, [1/0/0/32]
<-----
CON [MID=1238] POST /session/4ad6bc29, T=a, [1/1/1/32]
Certificate*
ClientKeyExchange
CertificateVerify*
[ChangeCipherSpec]
Finished ----->
.
.
.
CON [MID=1245] POST /session/4ad6bc29, T=a, [1/7/0/32]
Certificate*
ClientKeyExchange
CertificateVerify*
[ChangeCipherSpec]
Finished ----->
ACK [MID=1245], 2.04 Changed, [1/7/0/32]
<-----
CON [MID=4244] 2.04, T=a, [2/0/0/32]
[ChangeCipherSpec]
<----- Finished

ACK [MID=4244] 0, [2/0/0/32] ----->

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

3 Security Considerations

<Security considerations text>

4 IANA Considerations

<IANA considerations text>

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 Thigns 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 Consdierations 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
University of Glasgow Singapore
Republic PolyTechnic, 9 Woodlands Ave 9
Singapore 838964
SG

Email: SyeLoong.Keoh@glasgow.ac.uk

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