eeDTLS: Energy-Efficient Datagram Transport Layer Security for the Internet of Things

Citation: Baneerjee, Utsav et al. "eeDTLS: Energy-Efficient Datagram Transport Layer Security for the Internet of Things." 2017 IEEE Global Communications Conference, December 2017, Singapore, Singapore, Institute of Electrical and Electronics Engineers (IEEE), January 2018 © 2017 IEEE

As Published: http://dx.doi.org/10.1109/glocom.2017.8255053

Publisher: Institute of Electrical and Electronics Engineers (IEEE)

Persistent URL: https://hdl.handle.net/1721.1/122466

Version: Author's final manuscript: final author's manuscript post peer review, without publisher's formatting or copy editing

eeDTLS: Energy-Efficient Datagram Transport Layer Security for the Internet of Things

Utsav Banerjee*, Chiraag Juvekar*, Samuel H. Fuller† and Anantha P. Chandrakasan*
*Massachusetts Institute of Technology, Cambridge, MA, USA
†Analog Devices Inc., Norwood, MA, USA

Abstract—**In the fast growing world of the Internet of Things (IoT), security has become a major concern. Datagram Transport Layer Security (DTLS) is considered to be one of the most suited protocols for securing the IoT. However, computation and communication overheads make it very expensive to implement DTLS on resource-constrained IoT sensor nodes. In this work, we profile the energy costs of DTLS 1.3, using experimental models for cryptographic computations and radio-frequency (RF) communications. Based on this analysis, we present eeDTLS, a low-energy variant of DTLS, that provides the same security strength as DTLS, but has lower energy requirements. By employing a combination of packet size reduction and optimized handshake computations, eeDTLS can provide up to 45% energy savings in a typical IoT use case. eeDTLS can be implemented in conjunction with any low-energy IoT RF protocol, and the proposed energy models and protocol optimizations can also be used to improve the energy efficiency of custom IoT security architectures.**

I. Introduction

The Internet of Things (IoT) envisions a scenario where a large variety of wireless electronic devices sense data from their surrounding environments, process the data to obtain useful information, and communicate this information to the cloud. Researchers estimate that there will be over 50 billion wireless connected devices by 2020. On one hand, IoT enables fundamentally new and innovative applications, but on the other, these devices are attractive targets for cyber attackers, thus making IoT security a major concern. According to a security survey from 2016 [1], only 10% IoT products have adequate security features. The number of sensors in an IoT network can vary from tens (e.g., smart home and healthcare solutions) to thousands (e.g., agricultural and industrial automation) depending on the application. These devices must operate unattended for extended periods of time, and are either battery-powered or rely on energy harvesting, thus making them severely energy-constrained. Therefore, it is of utmost importance to consider energy efficiency when designing security protocols for such networks.

Since IoT will be integrated with the conventional Internet, most IoT devices are expected to use Internet Protocol (IP) addresses as unique identifiers, and IPv6 will be used to fulfill the large address space requirements. Therefore, IP-based security protocols become the first choice for securing the IoT. Transport Layer Security (TLS) is a cryptographic protocol widely used by the Internet community to provide secure and reliable data communications for applications such as e-mail and financial transactions, and this forms the basis of HTTPS (secure HTTP). TLS has been standardized by the Internet Engineering Task Force (IETF) [2] to secure connection-based Internet services such as Transmission Control Protocol (TCP). TLS-secured TCP-based applications form the backbone of the Internet, but TCP is not suitable for low-power wireless networks, primarily because of protocol overheads. The User Datagram Protocol (UDP) has emerged as the transport layer protocol-of-choice for the IoT. UDP-based services are connection-less and light-weight, hence they require low bandwidth and minimal memory usage on embedded devices. The Datagram Transport Layer Security (DTLS) protocol is based on TLS, and is intended to secure UDP-based communications. Connection-less services are unreliable, and present unique challenges such as packet re-ordering, packet loss and packet fragmentation. DTLS is designed to not only handle these problems seamlessly, but also counter replay and denial-of-service (DoS) attacks. DTLS has also been standardized by the IETF [3], and is considered as one of the most suited protocols for securing the IoT [4].

The DTLS protocol has been tried and tested for over a decade, and provides strong security guarantees. However, computation and communication overheads make DTLS very expensive for energy-constrained IoT devices. In this work, we present a comprehensive study of the energy costs of DTLS version 1.3. We consider the Bluetooth Low Energy (BLE) physical and link layer protocol as a case study, and use energy models for cryptographic computations and radio-frequency (RF) communications to analyze how the energy consumption varies over different IoT use cases. Based on this analysis, we present an optimized energy-efficient variant of DTLS – eeDTLS, that can be used to achieve improved energy-efficiency while providing the same security guarantees of DTLS.

II. Overview of DTLS 1.3

The Network Working Group of IETF is in the process of standardizing the next version of TLS – TLS 1.3. Both TLS 1.3 [5] and DTLS 1.3 [6] currently exist in the form of working drafts. (D)TLS consists of two layers – *record protocol* and *handshake protocol*. The record protocol encrypts application layer payloads, fragments and encapsulates them into structured packets, called *records*, and provides message authentication. The handshake protocol allows the communicating parties (*client* and *server*) to negotiate security settings, perform mutual authentication and establish a secure channel

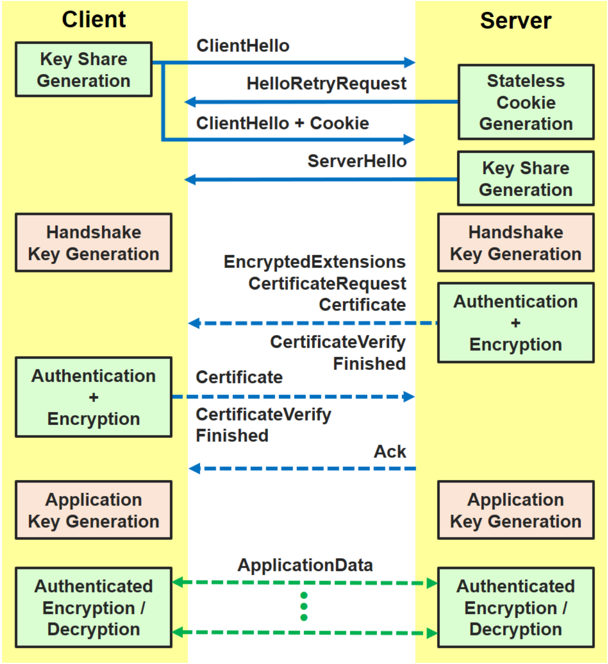

Fig. 1. Overview of DTLS 1.3 handshake protocol with mutual authentication and key exchange (blue arrows represent handshake messages and green arrows represent application data; dashed arrows indicate that the messages are encrypted).

TABLE I
ENERGY CONSUMPTION OF SYMMETRIC CRYPTOGRAPHY ALGORITHMS – EXPERIMENTAL RESULTS ON ARM CORTEX-M0+

Cryptographic Computation	Energy
AES-128-GCM Auth-Encrypt	0.121 μJ/B
AES-128-GCM Auth-Decrypt	0.124 μJ/B
AES-256-GCM Auth-Encrypt	0.141 μJ/B
AES-256-GCM Auth-Decrypt	0.145 μJ/B
SHA-256 Message Digest	0.043 μJ/B
SHA-256 HMAC (64-Byte Key)	0.052 μJ/B
SHA-512 Message Digest	0.089 μJ/B
SHA-512 HMAC (128-Byte Key)	0.122 μJ/B

TABLE II
ENERGY CONSUMPTION OF ELLIPTIC CURVE CRYPTOGRAPHY ALGORITHMS – EXPERIMENTAL RESULTS ON ARM CORTEX-M0+

Cryptographic Computation	Energy
P-256 ECDHE	33.06 mJ/Op
P-256 ECDSA-Sign	12.36 mJ/Op
P-256 ECDSA-Verify	34.02 mJ/Op
P-384 ECDHE	69.26 mJ/Op
P-384 ECDSA-Sign	25.43 mJ/Op
P-384 ECDSA-Verify	70.42 mJ/Op
P-521 ECDHE	143.92 mJ/Op
P-521 ECDSA-Sign	52.08 mJ/Op
P-521 ECDSA-Verify	145.49 mJ/Op

for the exchange of encrypted records. (D)TLS 1.3 proposes a major overhaul of the handshake protocol, removes support for weaker cryptographic primitives and adds stronger security measures than its predecessor.

Fig. 1 shows the message flow for a full DTLS 1.3 handshake with digital certificate-based mutual authentication and Diffie-Hellman key exchange [7]. The client begins the DTLS handshake by sending a *ClientHello* message containing details about supported cipher suites, public-key parameters and key shares for key exchange. The server then computes a stateless cookie and sends it in the *HelloRetryRequest* message. Next, the client sends another *ClientHello*, but now with the cookie it received from the server, and the server replies with a *ServerHello* containing its key share and selected security parameters. This procedure ensures that attackers cannot mount DoS attacks on the client or the server using forged handshake requests [3]. The remaining part of the handshake is completely encrypted using keys derived from the Diffie-Hellman shared secret. The server continues the handshake with an *EncryptedExtensions* message containing additional protocol settings, and a *CertificateRequest* message to indicate that it requires client authentication. These are followed by the server's *Certificate* and a *CertificateVerify* message that authenticates the server's side of the key exchange. The server ends this flight of messages with a *Finished* message that authenticates the handshake and confirms the security of the encrypted channel. The client replies with its own set of *Certificate*, *CertificateVerify* and *Finished* messages. Since UDP packets may be lost, the DTLS 1.3 server is required to acknowledge the receipt of this final set of messages with an *Ack* message. This ends the handshake, and the two parties can now exchange *ApplicationData* encrypted under a new set of keys derived from the handshake parameters.

III. DTLS OVER BLE - ENERGY MODELS & CASE STUDY

In order to analyze the energy costs of DTLS and make necessary optimizations, it is important to have accurate energy models for the cryptographic computations and RF communications. In this section, we discuss the energy models used to motivate eeDTLS, along with a comprehensive analysis of the energy consumption of DTLS 1.3 handshake and application data for some typical IoT use cases.

A. Software Profiling of Cryptographic Algorithms

DTLS owes its security to cryptographic algorithms of varying complexity, such as symmetric key encryption, hashing, public key authentication and key exchange. However, they also add to the computation cost of DTLS, which is a serious concern for resource-constrained embedded devices that constitute the IoT. To accurately profile the energy requirements of these cryptographic primitives, we implemented them in software on the NXP FRDM-KL25Z evaluation board, which contains an ultra-low-power 90nm ARM Cortex-M0+ micro-processor running at 48 MHz [8]. The software was implemented as bare-metal C code using the open-source cryptographic libraries from ARM mbedTLS [9]. Total energy consumption of the processor core and memory is reported (Tables I and II).

Table I shows our experimental results for symmetric cryptography algorithms – AES (Advanced Encryption Standard) [10] and SHA-2 (Secure Hash Algorithm) [11]. We implemented AES-GCM (Galois Counter Mode) with 12-byte initialization vector (IV) and 13-byte additional authenticated data (AAD). For SHA-2, we implemented both Message Digest (MD) and Hash-Based Message Authentication Code (HMAC). Energy consumption is reported per byte of input message. Table II shows experimental results for ECDHE (Elliptic Curve Diffie-Hellman Key Exchange) and ECDSA (Elliptic Curve Digital Signature Algorithm) [12]. Energy consumption is reported per operation for ECDHE, ECDSA-Sign and ECDSA-Verify. The NIST standard prime curves P-256, P-384 and P-521 were used for this analysis. Windowing methods, with window size $W = 3$, were used for faster elliptic curve operations, along with efficient modular arithmetic owing to the special structure of the NIST primes. Larger window sizes could not be used due to memory constraints of the processor. Clearly, the elliptic curve cryptography (ECC) algorithms are significantly more expensive compared to AES and SHA, and will contribute to majority of DTLS computation costs, as will be discussed later.

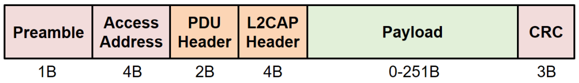

Fig. 2. Bluetooth Low Energy 4.2 data packet structure (all sizes in bytes).

B. Energy Model for BLE

BLE is the low-energy version of Bluetooth [13] that operates at 1 Mbps data rate, and employs adaptive frequency hopping spread-spectrum to communicate over the unlicensed 2.4 GHz ISM band. Energy consumption of BLE is much lower than other RF protocols like IEEE 802.15.4 [14], thus making it the popular choice for IoT applications. Fig. 2 shows a standard BLE 4.2 data packet. The 4 byte access address is the physical address of the slave device. The PDU (Physical Data Unit) header contains control flags and the payload size (in bytes), while the L2CAP (Logical Link Control and Adaptation Protocol) header contains information about packet fragmentation. The payload can be up to 251 bytes long, and its integrity is protected by a 3-byte CRC (Cyclic Redundancy Check).

BLE networks, called *piconets*, are comprised of multiple *slave* devices connected to a *master* device which coordinates all the communications, that is, a piconet is inherently in star topology. Slave devices are in sleep for most of the time, except for periodic connection events when the slave wakes up to communicate with the master. Connection events always start with a packet being sent by the master, and the slave has to wait for 150 μs, called Inter-Frame Space (IFS), before transmitting data. Table III shows the energy consumed by the TI CC2540 BLE transceiver [15], [14] during different phases of a connection event, when it has to transmit data to the master. The energy spent by a BLE 4.2 slave device during transmission (E_T) and reception (E_R) of data can be modeled using the following equations [14]:

$$\begin{aligned} E_T &= E_{WUP} + nl_{HDR}E_{RX} + (2n-1)E_{IFS} + \\ &\quad (nl_{HDR} + l_P)E_{TX} + E_{SLP} \\ E_R &= E_{WUP} + (nl_{HDR} + l_P)E_{RX} + \\ &\quad (2n-1)E_{IFS} + nl_{HDR}E_{TX} + E_{SLP} \end{aligned}$$

where l_{HDR} (= 14 bytes) is the total size of BLE header and trailer structures, l_P is the total payload being transmitted / received, and n is the number of fragments the payload gets divided into.

TABLE III
ENERGY CONSUMPTION OF BLE TRANSCEIVER DURING DIFFERENT PHASES OF CONNECTION EVENT [14]

Phase	Energy
Wake-Up and Pre-Processing	E_{WUP} = 15 μJ
Receive (RX)	E_{RX} = 0.528 μJ/B
Inter-Frame Space (IFS)	E_{IFS} = 6.75 μJ
Transmit (TX)	E_{TX} = 0.672 μJ/B
Post-Processing and Sleep	E_{SLP} = 33.6 μJ

C. DTLS over BLE

Using these models, we can now accurately analyze the energy consumption of a duty-cycled BLE sensor node communicating with a cloud server using a DTLS-protected secure channel. Although absolute values of RF energy consumption may vary among different commercial transceivers, this analysis is sufficient to predict the energy trends is typical IoT applications. As case study, we consider a DTLS 1.3 connection with the following parameters:

- The negotiated cipher suite is TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256, and the elliptic curve used for ECDHE and ECDSA is P-256. These primitives guarantee a security level of 128 bits.
- Only end-point certificates, signed using P-384 ECDSA by a trusted certification authority (CA), are exchanged during handshake (assuming the CA public key is known to both parties).

Fig. 3 shows the structure of a DTLS-protected BLE packet with encrypted application data. Since the total BLE 4.2 payload size is restricted to 251 bytes, maximum size of the encrypted data in a single packet is 174 bytes. Larger

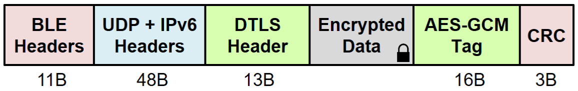

Fig. 3. Structure of DTLS-protected BLE packet with additional headers contributed by IPv6, UDP and DTLS (all sizes in bytes). The encrypted application data can be up to 174 bytes long, and message authentication is provided by the AES-GCM tag.

TABLE IV
ENERGY COSTS OF DTLS 1.3 – COMPUTATIONS AND COMMUNICATIONS

Protocol Phase	**Payload (Bytes)**	**Energy (μJ)**		**Cryptographic Computation Details**
		Compute	**RF**	
ClientHello (T)	180	16528	305.4	0.5 × P-256 ECDHE
HelloRetryRequest (R)	50	-	130.8	-
ClientHello + Cookie (T)	210	-	325.6	-
ServerHello (R)	130	-	173.0	-
Handshake Traffic Key Generation	-	16565.8	-	0.5 × P-256 ECDHE + 1 × SHA-256-MD (570 Bytes) + 8 × SHA-256-HMAC (32 Bytes each)
EncryptedExtensions + CertificateRequest (R)	50	6.2	139.2	1 × AES-128-GCM-Auth-Decrypt (50 Bytes)
Server Certificate (R)	600	70497.4	642.5	1 × AES-128-GCM-Auth-Decrypt (600 Bytes) + 1 × P-384-ECDSA-Verify
Server CertificateVerify + Server Finished (R)	130	34093.4	181.5	1 × AES-128-GCM-Auth-Decrypt (130 Bytes) + 1 × P-256-ECDSA-Verify + 1 × SHA-256-MD (1300 Bytes) + 2 × SHA-256-HMAC (32 Bytes each)
Client Certificate (T)	600	72.6	773.2	1 × AES-128-GCM-Auth-Encrypt (600 Bytes)
Client CertificateVerify + Client Finished (T)	130	12465.4	211.2	1 × AES-128-GCM-Auth-Encrypt (130 Bytes) + 1 × P-256-ECDSA-Sign + 1 × SHA-256-MD (2030 Bytes) + 2 × SHA-256-HMAC (32 Bytes each)
Server Ack (R)	20	2.48	123.4	1 × AES-128-GCM-Auth-Decrypt (20 Bytes)
Application Traffic Key Generation	-	69.7	-	1 × SHA-256-MD (1350 Bytes) + 7 × SHA-256-HMAC (32 Bytes each)
Total Handshake Energy (μJ):		150.3×10^3	3.00×10^3	
ApplicationData (T)	32	3.9	145.4	1 × AES-128-GCM-Auth-Encrypt (32 Bytes)

application data get fragmented into $n = \lceil l_P/174 \rceil$ packets, where l_P is the total number of bytes to be transmitted. An un-encrypted packet has very similar structure, except for the absence of the 16-byte AES-GCM tag, that is, $n = \lceil l_P/190 \rceil$.

Table IV provides a detailed analysis of the energy spent by a DTLS 1.3 client device in cryptographic computations and RF communications, both during the handshake and the application data phases. Typical sizes of DTLS handshake messages are provided (rounded to the nearest ten bytes), along with the computations required to generate them [5]. (T) and (R) indicate whether packets are transmitted or received respectively. We have assumed that the client periodically transmits 32 bytes of data in the *ApplicationData* phase, that is, after the handshake is completed. Average energy consumption of DTLS handshake computations, on the FRDM-KL25Z running the mbedTLS stack [9], was measured to be 157 mJ, which is close to our estimate in Table IV.

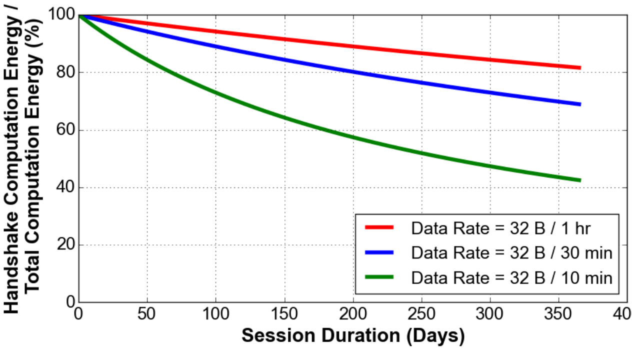

Fig. 4. Percentage of total computation energy spent in DTLS handshake, for application data rates of 32 bytes per hour, per 30 minutes, and per 10 minutes. Session durations vary from 1 day to 365 days.

As conjectured earlier, the handshake computations are largely due to the infrequent, but expensive, ECC operations. Fig. 4 shows the fraction of total computation energy that is spent in performing the handshake. With a typical data rate of 32 bytes per hour, the handshake accounts for 82% of the total computation energy in case of year-long sessions, and 99% for week-long sessions. This percentage becomes lower for faster data rates, e.g., 32 bytes per 10 minutes, which may apply to applications with real-time data requirements.

In order to analyze how much the RF transceiver contributes to the total energy consumption, we consider two prototypical scenarios – (a) session duration = 1 year (365 days), and (b) session duration = 1 week (7 days), both at the same data rate of 32 bytes per hour, so that 8760 packets are sent in (a), and 168 packets in (b). Session durations are typically determined by how often the DTLS authentication handshake is performed, which in turn depends on the security requirements of the IoT application. Fig. 5 shows the energy breakdown for these two use cases. "Application Data RF" accounts for 87% of the total energy in (a), while "Handshake Compute" accounts for 84% of the total energy in (b). For both use cases, “Application Data Compute” and “Handshake RF” consume relatively negligible energy. Therefore, we need to reduce the energy costs of both “Application Data RF” and “Handshake Compute” in order to minimize overall energy consumption.

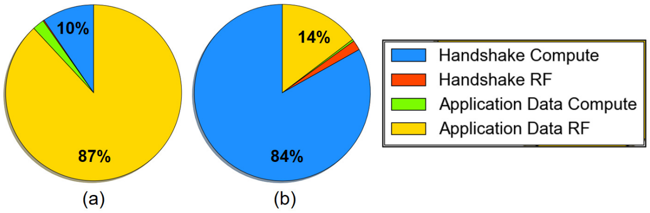

Fig. 5. Energy breakdown of DTLS session computations and communications, for session durations of (a) 1 year and (b) 1 week, with data rate of 32 bytes per hour.

IV. Energy-Efficient DTLS

From the results discussed in the previous section, we realize that different IoT applications will require different optimizations to achieve energy-efficiency. We follow a two-step approach to optimize the protocol, which we call “eeDTLS” – packet optimizations to reduce “Application Data RF” energy, and handshake optimizations to reduce “Handshake Compute” energy.

A. Packet Optimizations

The only way to reduce energy consumption of the RF transceiver, without modifying its circuitry or physical layer protocols, is to have smaller packets. We propose to optimize the following components of the packet structure:

- 48-byte UDP and IPv6 headers
- 13-byte DTLS header
- 16-byte AES-GCM tag

BLE headers are left untouched because we want eeDTLS to be easily portable over different physical and link layer protocols.

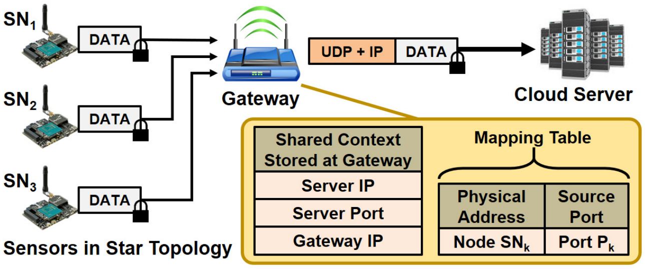

Fig. 6. IoT network architecture, with sensor nodes (SN) and a gateway, for UDP/IPv6 packet optimizations. The gateway maintains a mapping table and stored connection context, which it uses to fill in UDP/IPv6 headers before sending data to the cloud.

Header compression schemes have been proposed in [16], which can reduce the sizes of UDP and IPv6 headers. In this work, we exploit some properties of the network architecture to completely eliminate these headers. We assume that all sensor nodes are connected to a *gateway* device in star topology, which is the default architecture for BLE (Fig. 6). The gateway maintains an address translation table that maps BLE physical addresses of the client nodes into corresponding UDP ports. This table is used to translate BLE packets into valid UDP/IPv6 packets that can be sent through existing IP-based cloud infrastructure, and vice-versa.

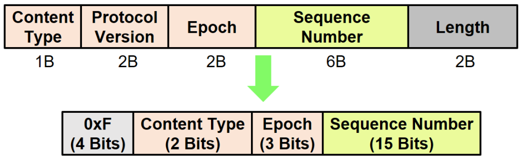

Fig. 7. (top) Standard 13-byte DTLS header (all sizes in bytes), and (bottom) Optimized 3-byte eeDTLS header.

Next, we propose to reduce the 13-byte DTLS record header to a fixed size of 3 bytes (Fig. 7), unlike the variable-size header compression proposed in [17]. “Protocol Version” is constant, and “Length” can be inferred from the physical layer (PDU) header, so these fields are omitted. We use 2 bits for “Content Type”, which has three possible values; and 3 bits for “Epoch”, which can vary from 0 to 5 (when non-forward-secret key updates are not allowed). The number of bits assigned to the sequence number is dictated by our final optimization – *truncated AES-GCM tags*. According to [18], AES-GCM tags can be truncated to 32 bits, provided the same encryption key is not used for more than 2^{15} packets, each up to 256 bytes in length. This sets the upper limit for the DTLS sequence number to $2^{15} - 1$, that is, 15 bits. Since the header size must be a multiple of 8 bits, the upper 4 bits are set to 0xF, which indicates that eeDTLS is being used. The complete 13-byte record header is used as AAD for AES-GCM, therefore security is preserved. Using truncated tags mandates performing a handshake after every 2^{15} transmitted packets, but this number is large enough for typical IoT applications.

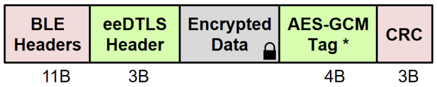

Fig. 8. Optimized eeDTLS packet over BLE, with 91% reduction in protocol overheads (all sizes in bytes).

Fig. 8 shows the optimized eeDTLS packet. Protocol overheads have been reduced by 91%, from 77 bytes to 7 bytes. By pushing the protocol overheads to their lower limit, we have also increased the maximum encrypted data size to 244 bytes, which further improves energy efficiency by allowing the client node to buffer sensor data, and reduces fragmentation of packets ($n = \lceil l_P/244 \rceil$ for eeDTLS). “Application Data RF” energy is reduced by 33%, which is smaller than the reduction in number of bytes. This is due to the energy consumed by the duty-cycled RF transceiver to wake up and power down.

B. Handshake Optimizations

ECDSA certificates constitute a bulk of the handshake energy consumption. In energy-constrained IoT applications, it is fair to assume that the client and the server can cache each other's public keys to authenticate the key exchange, for which we use two seldom-used TLS extensions – "Client Certificate URL" [19] and "Cached Information" [20]. This not only reduces the *Certificate* messages to few tens of bytes, but also eliminates the need to verify CA signatures in the certificates, which is particularly helpful because CA signatures use higher security levels. In our case study, the optimized handshake has 47% lower energy consumption.

C. Analysis of eeDTLS

Fig. 9 summarizes the energy benefits of eeDTLS for our two test scenarios. For case (a) with 1 year session, packet optimizations provide 28% energy reduction, and handshake optimizations provide an additional 7%. For case (b) with 1 week session, packet optimizations provide only 4% energy reduction, while handshake optimizations provide 42%. Therefore, packet optimizations provide energy benefits for use cases with less frequent handshakes (a), while handshake optimizations help with applications where handshakes need to be performed more frequently (b). Overall, eeDTLS provides 33% and 45% energy reduction respectively for (a) and (b).

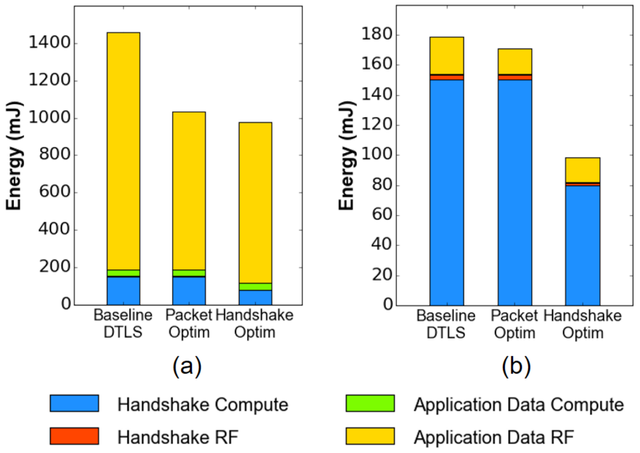

Fig. 9. Energy benefits of eeDTLS, for session durations of (a) 1 year and (b) 1 week, with data rate of 32 bytes per hour.

V. Conclusion and Future Work

In this work, we have described energy models for cryptographic computations and RF communications in a DTLS-secured IoT device, and used them to analyze the energy costs of DTLS 1.3 over BLE 4.2. Based on this analysis, we have presented eeDTLS, an optimized low-energy variant of the DTLS protocol that retains its strong security and authentication properties. Our optimizations can provide up to 33% reduction in energy for long DTLS sessions, and up to 45% reduction for short sessions. Although BLE has been used for our case study, these optimizations can be seamlessly ported to any other RF protocol, e.g., IEEE 802.15.4, etc. Also, our energy models can be used to analyze custom IoT security protocols, and determine appropriate optimizations. Since "Handshake Compute" is a significant portion of the energy consumption, this also motivates the design of dedicated energy-efficient hardware for the cryptographic operations in DTLS, especially public-key algorithms like ECC.

Acknowledgment

The authors would like to thank Joshua Nekl from Analog Devices Inc. for useful discussions. The authors acknowledge financial support from the Irwin and Joan Jacobs MIT Presidential Fellowship, the Qualcomm Innovation Fellowship and Analog Devices Inc.

References

[1] IOActive Press Release, "Less Than 10% of Internet of Things (IoT) Products Have Adequate Security According to Practitioner Survey," June 2016. [Online]. Available: http://www.ioactive.com/news-events/iot-products-have-inadequate-security-according-to-practitioner-survey.html

[2] T. Dierks and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2," *IETF RFC*, vol. 5246, August 2008.

[3] E. Rescorla and N. Modadugu, "Datagram Transport Layer Security Version 1.2," *IETF RFC*, vol. 6347, January 2012.

[4] S. L. Keoh, S. S. Kumar and H. Tschofenig, "Securing the Internet of Things: A Standardization Perspective," in *IEEE Internet of Things Journal*, vol. 1, no. 3, pp. 265-275, June 2014.

[5] E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.3," *IETF Internet-Draft*, March 2017. [Online]. Available: https://tlswg.github.io/tls13-spec/

[6] E. Rescorla and H. Tschofenig, "The Datagram Transport Layer Security (DTLS) Protocol Version 1.3," *IETF Internet-Draft*, October 2016. [Online]. Available: https://tools.ietf.org/html/draft-rescorla-tls-dtls13-00

[7] W. Diffie and M. Hellman, "New Directions in Cryptography," *IEEE Transactions on Information Theory*, vol. 22, no. 6, pp. 644-654, November 1976.

[8] NXP Semiconductors, "Kinetis KL25 Sub-Family: 48 MHz Cortex-M0+ Based Microcontroller with USB," *Data Sheet*, Rev. 5, August 2014.

[9] ARM Holdings, *ARM mbedTLS*. [Online]. Available: https://tls.mbed.org

[10] NIST, "Advanced Encryption Standard (AES)," *NIST Technical Report*, FIPS PUB 197, November 2001.

[11] NIST, "Secure Hash Standard (SHS)," *NIST Technical Report*, FIPS PUB 180-4, March 2012.

[12] Certicom Research, "SEC 1: Elliptic Curve Cryptography," *Standards for Efficient Cryptography*, Version 2.0, May 2009.

[13] Bluetooth SIG, "Bluetooth Specification 4.2." [Online]. Available: https://www.bluetooth.com/specifications/bluetooth-core-specification

[14] M. Siekkinen, M. Hiienkari, J. K. Nurminen and J. Nieminen, "How Low Energy is Bluetooth Low Energy? Comparative Measurements with ZigBee/802.15.4," *IEEE Wireless Communications and Networking Conference Workshops (WCNCW)*, pp. 232-237, April 2012.

[15] Texas Instruments Inc., "CC2540: 2.4GHz Bluetooth Low Energy System-on-Chip," *Data Sheet*, June 2013.

[16] J. Nieminen, T. Savolainen, M. Isomaki, B. Patil, Z. Shelby and C. Gomez, "IPv6 over BLUETOOTH (R) Low Energy," *IETF RFC*, vol. 7668, October 2015.

[17] S. Raza, D. Trabalza and T. Voigt, "6LoWPAN Compressed DTLS for CoAP," *IEEE International Conference on Distributed Computing in Sensor Systems*, pp. 287-289, May 2012.

[18] NIST, "Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC," *NIST Special Publication*, vol. 800-38D, November 2007.

[19] D. Eastlake, "Transport Layer Security (TLS) Extensions: Extension Definitions," *IETF RFC*, vol. 6066, January 2011.

[20] S. Santesson and H. Tschofenig, "Transport Layer Security (TLS) Cached Information Extension," *IETF RFC*, vol. 7924, July 2016.