BlinkToSCoAP: An End-to-End Security Framework for the Internet of Things

Giulio Peretti[§], Vishwas Lakkundi[†*] and Michele Zorzi[§]
[§]Department of Information Engineering, University of Padova, Italy
[†]Altiux Innovations, Bangalore, India

***Abstract*—The emergence of Internet of Things and the availability of inexpensive sensor devices and platforms capable of wireless communications enable a wide range of applications such as intelligent home and building automation, mobile healthcare, smart logistics, distributed monitoring, smart grids, energy management, asset tracking to name a few. These devices are expected to employ Constrained Application Protocol for the integration of such applications with the Internet, which includes User Datagram Protocol binding with Datagram Transport Layer Security protocol to provide end-to-end security. This paper presents a framework called BlinkToSCoAP, obtained through the integration of three software libraries implementing lightweight versions of DTLS, CoAP and 6LoWPAN protocols over TinyOS. Furthermore, a detailed experimental campaign is presented that evaluates the performance of DTLS security blocks. The experiments analyze BlinkToSCoAP messages exchanged between two Zolertia Z1 devices, allowing evaluations in terms of memory footprint, energy consumption, latency and packet overhead. The results obtained indicate that securing CoAP with DTLS in Internet of Things is certainly feasible without incurring much overhead.**

***Index Terms*—Internet of Things; M2M; DTLS; CoAP; end-to-end security; 6LoWPAN.**

I. Introduction

The Internet of Things (IoT) encompasses heterogeneous entities such as sensors, actuators and mobile devices operating in a constrained environment consisting of low-power and lossy networks. The IoT allows a wide range of application scenarios and their integration with the Internet, wherein the IoT entities interact with each other, with Internet remote services as well as with human beings carrying an Internet-capable device such as a smartphone or a tablet PC. To facilitate this integration, the core building blocks used for IoT applications include the web services, the IPv6 protocol and its lightweight version obtained through 6LoWPAN.

The Constrained Application Protocol (CoAP) is a specialized web transfer protocol most suited for constrained devices in Machine-to-Machine (M2M) applications. The CoAP protocol provides a client/server interaction model between application endpoints and includes the same key functionalities of the HTTP protocol. Therefore, CoAP is easily interfaced with HTTP, resulting in simplified web integration while also ensuring M2M critical requirements such as low overhead, multicast support, built-in discovery and simplicity. Like HTTP, the CoAP client/server interaction model requires that CoAP requests are sent by clients in order to request an action on an available resource of the server. Upon receiving the request, the server responds to the client with a CoAP response containing an appropriate response code and an optional resource representation [1]. As shown in Fig. 1, CoAP is characterized by a 4-bytes header that contains the *protocol version number*, the *message type* (CON, NON, ACK, RST), the *Token length*, the *code* of the client request or server response and the *message ID* used to match messages as well as to detect message duplicates.

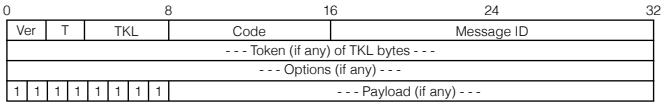


Fig. 1: CoAP message format.

Datagram Transport Layer Security (DTLS) protocol – a modified version of Transport Layer Security (TLS), the most widespread end-to-end web security protocol [2] – resolves the original protocol issues when running over unreliable transport protocols such as UDP. In order to accomplish this requirement, DTLS adds to the TLS headers the following fields: *Handshake Message Number* (to reconstruct the correct order of handshake messages), Record Sequence Number (to reconstruct the correct order of Record messages) and *Epoch* (to distinguish messages encrypted with different cypher suites). DTLS is designed to be as similar to TLS as possible in order to take advantage of pre-existing protocol infrastructure [3]. It provides confidentiality, integrity protection and authenticity.

This paper presents an IoT framework that includes the CoAP protocol protected by the DTLS protocol running over the 6LoWPAN stack. It is realized by merging, adapting and optimizing various implementations of these protocols developed in-house at the SIGNET Group, Department of Information Engineering, University of Padova. Furthermore, we present an in-depth analysis of performance variation due to the introduction of the security protocol in terms of additional memory usage, computational time, energy overhead and the packet overhead introduced by the DTLS header. Our results infer that CoAP can be secured with DTLS, thus providing end-to-end security in IoT networks consisting of even Class 1 constrained devices with ∼10KB of RAM and ∼100KB of flash [4].

*The author was with Patavina Technologies Italy while carrying out this work.

This paper is divided into five sections. State-of-the-art on IoT security is included in Section II. Our BlinkToSCoAP security framework is described in Section III, while Section IV presents the results obtained and their performance analysis. Concluding remarks are drawn in Section V.

II. Related Work

Recently, a lot of research into end-to-end security protocols for CoAP-based IoT is being carried out.

Authors of [5] introduce a DTLS security architecture that performs two-way authentication, which includes both client and server authentication, based on RSA. This particular asymmetric encryption algorithm is too heavy for constrained devices and the devices considered in their architecture have to guarantee hardware support for secure application, and RSA key storage, such as Trusted Platform Modules (TPMs).

In [6], the performance impact of several DTLS security modes, proposed by the CoAP standard is analyzed in order to identify the limitations of node platforms and the requirements of IoT applications. The authors have evaluated energy, packet and computational overheads as well as the memory footprint of various security modes, showing that the small memory space and absence of Elliptic Curve Cryptography (ECC) hardware support are a critical aspect for the compatibility of IoT networks with existing public-key certification infrastructures. However, some DTLS security suites were identified as viable if security and resources usage compromises are allowed by the network application.

In [7], an extensive experimental evaluation is presented to identify the most appropriate secure communications mechanism between end-to-end network-layer and application-layer security, compared in terms of energy, computational overheads and memory footprints. The authors have described the impact of end-to-end security on the communication rate of sensing devices as well as on the lifetime of the constrained network. The end-to-end approach provides the benefit of enabling secure communications regardless of the application, while the network-layer security may facilitate the integration with certification infrastructures through the usage of ECC, but at the cost of extra resources.

In [8], it has been shown that DTLS headers can be compressed using 6LoWPAN mechanisms, significantly reducing the number of additional security bits. This leads to an increment of both the network lifetime and the achievable throughput. The same authors, in a more recent work [9], have presented a DTLS secured CoAP implementation that depends entirely on DTLS header compression for enhanced energy performance.

Securing CoAP with IPsec at the network layer is another option that has been explored in [10]. However, IPsec is neither supported by all the embedded IP stacks nor by all PC operating systems and back-end web servers.

We therefore further explore the inherent advantage that DTLS offers over IPsec, i.e. being an application layer protocol it is implemented in the application space instead of in the OS kernel. In addition, thanks to their similarities, DTLS allows the reuse of existing TLS protocol infrastructure at the cost of minimal application overhead [3].

III. BlinkToSCoAP Framework

The IoT security framework presented in this paper is called BlinkToSCoAP and it is coded in NesC, a dialectal form of the C programming language purposely created for the TinyOS embedded operating system and best suited for the hardware limits of sensor networks. TinyOS applications are component-based, which means that each application is composed of one or more software modules wired together by multiple interfaces. BlinkToSCoAP transmits encrypted CoAP messages over 6LoWPAN networks by means of the interconnection of several TinyOS components that implement lightweight versions of DTLS, CoAP and 6LoWPAN presented in detail in [11].

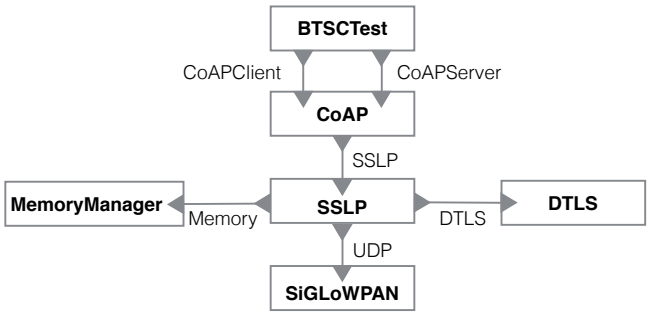


Fig. 2: BlinkToSCoAP architecture.

The BlinkToSCoAP architecture, depicted in Fig. 2, is conditioned by the pre-existent wiring structures of the underlying protocol components. At the top of the BlinkToSCoAP stack there is the BTSCTest component that acts as a CoAP Server or Client and initializes the necessary components and circuitry of the node. BTSCTest interprets the CoAP client or server role by making use of the CoAP functionalities provided by the two interfaces of the CoAP component, namely *CoAPClient* and *CoAPServer*.

The BlinkToSCoAP client periodically sends a CoAP *Blink* request to the BlinkToSCoAP server, which is waiting for incoming CoAP requests. Every time the server correctly receives a Blink message, an internal variable counter is incremented and sent to the client through the servers CoAP response. Upon every *Blink* receipt event, both server and client devices turn on three LEDs of the nodes in a configuration that represents the binary value of the shared counter.

Under the CoAP module, the SecureSiGLoWPAN (SSLP) component is designed to handle the exchange of data between CoAP, DTLS and 6LoWPAN protocols. Our optimized 6LoWPAN protocol is named as SiGLoWPAN. SSLP component is essential because the DTLS component is designed to interact with other components only through this interface in order to both provide encryption functionalities as well as access the transmission channel. Furthermore, SSLP intercepts CoAP transmission requests and responses, encrypts them by means of the DTLS component, and successively redirects the encrypted data to the UDP interface of the IPv6/SiGLoWPAN components in order to transmit it through the wireless channel. Vice-versa, when a new UDP datagram is signaled by the IPv6/SiGLoWPAN component, SSLP redirects its payload to the DTLS module to deliver the decrypted data to the upper CoAP component. Therefore, SSLP module is directly wired

with CoAP, SiGLoWPAN and DTLS components, acting like a gateway for the data flowing between them. SSLP also makes use of the MemoryManager component that belongs to the IPv6/SiGLoWPAN library, in order to be able to correctly interact with the SiGLoWPAN library itself.

In order to optimize the memory footprint of the proposed framework, some of its configuration parameters have been modified as follows: in the CoAP library, the maximum contemporaneous CoAP active transactions has been reduced from 5 to 2; in the SiGLoWPAN library, the maximum queue dimension for IPv6 packets has been reduced from 5 to 3, and the maximum number of IPv6 addresses a node can have has been reduced to 1; the amount of RAM dedicated to the dynamic management of the MemoryManager component has been reduced from 2500 to 500 bytes. Furthermore, the DTLS module has been optimized by eliminating redundant code and by replacing the most RAM expensive variable types with a more efficient variable type, wherever suitable.

The only cipher suite supported by our DTLS component is DTLS_PSK_WITH_AES_128_CBC_SHA-256, composed of Pre Shared Key (PSK) exchange algorithm involving the 128 bit Advanced Encryption Standard (AES) algorithm in Counter CBC-MAC (CCM) mode and a keyed Hash Message Authentication Code (HMAC) implementing 256-bit Secure Hash Algorithm (SHA-256).

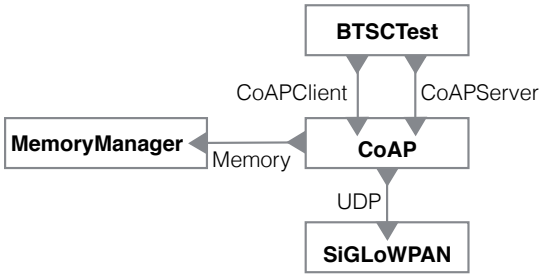


Fig. 3: BlinkToCoAP Architecture.

In order to highlight the performance variation due to the introduction of the DTLS module, the BlinkToSCoAP application has been modified into an equivalent application called BlinkToCoAP, which excludes DTLS. BlinkToCoAP is designed to be as close as possible to BlinkToSCoAP except that its CoAP module is interfaced directly to the UDP protocol to access the channel. The architecture of the unsecured application framework is depicted in Fig. 3.

IV. Implementation and Performance Analysis

An experimental evaluation campaign is performed using the proposed BlinkToSCoAP and BlinkToCoAP applications over a 6LoWPAN network to evaluate the performance variation due to the introduction of DTLS security modules. The network setup consists of two Zolertia motes that communicate directly in the 2.4 GHz ISM band by means of the IEEE 802.15.4 Medium Access Control (MAC) protocol with no Radio Duty Cycling (RDC) enabled. The performance is evaluated in terms of parameters such as memory footprint, packet overhead and energy consumption, whose experiments are described in the following sections together with their results.

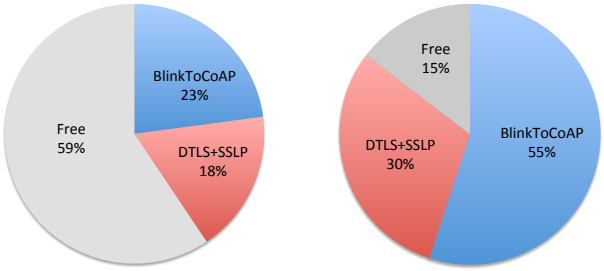
A. Memory Footprint

The memory footprint of the BlinkToSCoAP and BlinkToCoAP applications is provided by the GCC MSP430 Toolchain that displays the total ROM and RAM bytes written during the devices programming phase.

Table I: Memory footprint.

Application	RAM [bytes]	ROM [bytes]
BlinkToSCoAP	6832	37360
BlinkToCoAP	4404	21062

The two applications differ only by the presence of the DTLS protocol, meaning that the difference in their memory footprint is equal to the memory space dimension of SSLP and DTLS components. Memory footprints are shown in Table I. SSLP and DTLS modules together require 16298 extra bytes of ROM and 2428 extra bytes of RAM, which correspond to about 44% and 36% of the total BlinkToSCoAP ROM and RAM memory usage respectively. Fig. 4b and Fig. 4a show graphical representations of these results along with Zolertia Z1 memory still left available for use out of its total 92 KB flash memory and 8 KB RAM.



(a) Code size (ROM) footprint. (b) Data size (RAM) footprint.

Fig. 4: Memory footprints.

B. Packet Overhead

To calculate the packet overheads our experiment involved the use of a wireless packet sniffer board interfaced to a Linux system running the Wireshark packet analyzer software. This device is used to capture packets such as handshake messages, CoAP secured transactions and CoAP unsecured transactions exchanged over the wireless channel between the two nodes under test running BlinkToSCoAP or BlinkToCoAP setup. Wireshark is capable of parsing captured data in order to distinguish the various protocols headers, providing access to all their fields as well as their size and in particular, the effective DTLS header dimension. Moreover, this experiment also provides related time measurements. Packet transmissions are in fact events that occur either before or after CPU-intensive periods of time. Table II shows frame and UDP payload dimensions for secured and unsecured CoAP transactions, whereas Table III shows the length details of each of the handshake messages. CoAP protocol adds a total overhead of 17 bytes per frame, while DTLS protocol adds 29 more bytes. This overhead drastically reduces the maximum frame size available (102 octets) at the MAC layer, without link-layer security, as reported in [12].

TABLE II: COAP TRANSMISSION LENGTHS.

	Frame [bytes]	UDP payload [bytes]
Unsecured Request	30	13
Unsecured Response	24	7
Secured Request	59	42
Secured Response	53	36

TABLE III: HANDSHAKE MESSAGE LENGTHS.

	Frame [bytes]	UDP payload [bytes]
ClientHello	84	67
ClientHelloVerify	61	44
ClientHello (with cookie)	100	83
ServerHello + ServerHelloDone	105	88
ClientKeyExchange	59	42
ChangeCipherSpec	31	14
ClientFinished	70	53
ChangeCipherSpec	31	14
ServerFinished	70	53

C. Energy Consumption

Energy consumption of the employed hardware platform is obtained through experimental measurements of voltage across a current sensing resistor of 32.8 Ω placed in series with the Zolertia Z1 board and the USB cable used as power supply, as illustrated in Fig. 5. The voltage measurements are carried out by means of the UTD2102CEL Digital Storage Oscilloscope.

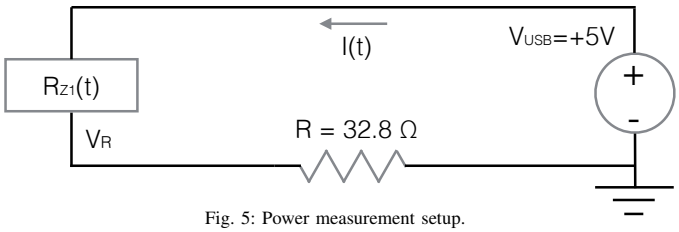


Fig. 5: Power measurement setup.

The Z1 platform is assumed to be a time variable resistor $R_{Z1}(t)$, while $R = 32.8\ \Omega$ is the sensing resistor, I(t) and $V_{USB} = 5V$ are the current and voltage supplied by the USB cable. Applying the Kirchhoff and Ohm laws, we have:

$$V_{USB} = (R + R_{Z1}(t))I(t)$$
$$\Rightarrow I(t) = \frac{V_{USB}}{R + R_{Z1}(t)}$$

The time variable platform equivalent resistance $R_{Z1}(t)$ can be estimated from the voltage measurements of the sensing resistor as follows:

$$R_{Z1}(t) = R\left(\frac{V_{USB}}{V_R(t)} - 1\right)$$

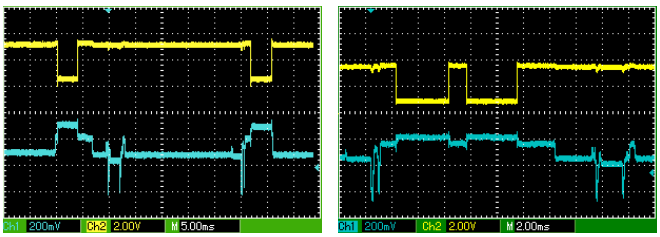
Furthermore, the power provided by the USB power source P_{USB} and the power drained by the Zolertia platform P_{Z1} are estimated as follows:

$$P_{USB} = V_{USB}I(t) = \frac{V_{USB}V_R(t)}{R}$$
$$P_{Z1} = V_{Z1}I(t) = \frac{V_R(t)}{R}\left(V_{USB} - V_R(t)\right)$$

By default, the adopted platform is intended to run at 3 Volts, as reported in [13], therefore the 5 Volts provided by the USB cable are reduced to the correct value by means of an *Automatic Voltage Regulator* (AVR) integrated on the motes CP2102 chip (USB-to-UART bridge), which introduces a performance loss in terms of power consumption. However, since performance evaluation is focused on the variation of energy consumption due to DTLS operations and not on their absolute values, the values get normalized across the board and this issue is neglected. The experiment involves different communication modes between two Zolertia platforms, namely CoAP secured, and unsecured transmissions and DTLS handshake, which are elaborated below.

1) CoAP Secured and Unsecured Transmissions: In this experiment, the two nodes have already established a secure channel by means of the DTLS Handshake Protocol. The expected time behavior of the resistor voltage, shown by the oscilloscope for the client device, is a series of incremental step caused by the amount of power drained by the Zolertia platform due to the request handling and transmission, followed by a period of inactivity when the mote is awaiting the server response and another step that signals the request receipt and further processing. In a complementary manner, the server nodes activity should display a unique step due to the request reception, and its processing and the subsequent response computation and transmission.

To highlight DTLS operations, the SSLP component is set to turn on a LED every time the DTLS module is called to handle a message, involving operations such as encryption, decryption, message computation and message parsing. When the security operations end, the LED is turned off. Furthermore, while one probe of the oscilloscope senses the voltage of the sensing resistor, a second one measures the activity of the toggling diode. The oscilloscope behavior for a secured transmission while employing a BlinkToSCoAP client is shown in Fig. 6a. The yellow line indicates LED activity. Its behavior corresponds to the CPU time used by the DTLS library for its security operations. Moreover, request transmission and response receipt are clearly visible on the blue line that indicates the voltage across the sensing resistor. The resistor voltage also shows radio operations, recognized as two consecutive peaks clearly visible after the request operation and in a less clear form just before the response operation. Between the CoAP request and its transmission, it is also possible to recognize the fraction of time when the mote is sensing the wireless channel in order to avoid packet collisions. The behavior of the same experiment applied to the BlinkToSCoAP server mote is shown in Fig. 6b.



(a) Client. (b) Server.

Fig. 6: Secured transmission and LED activity.

Furthermore, the experiment is repeated without LED activity in order to acquire reliable measurements of the sensing resistor voltage during CPU-intensive and inactivity phases. The same experiment is also repeated for BlinkToCoAP application in order to highlight the variation of performance due to security operations. Results of these experiments are shown in Fig. 7.

(a) Client, unsecured. (b) Client, secured.

(c) Server, unsecured. (d) Server, unsecured.

Fig. 7: Secured and unsecured transmission measurements.

The mean values of the sensing resistor voltage during the CPU-active, used to get the estimation of energy consumption, and CPU-inactive phases are given below:

$$V_{R,CPU-active} = 0.764[V]$$
$$V_{R,CPU-inactive} = 0.645[V]$$

The BlinkToSCoAP and BlinkToCoAP CPU time requirements for computation of CoAP requests, CoAP responses, handshake flights and radio transmissions are obtained from the voltage measurements. The energy consumption of different tasks performed by the Z1 motes can be calculated from these values as described earlier. CPU ticks, average energy consumption and processing time values are presented together in Table IV for CoAP secured and unsecured transactions.

The experiment involving LED indicators for DTLS CPU usage time provides precise time values of DTLS security operations, inclusive of message parsing, creation, encryption and decryption. These results are reported in Table IV together with estimated energy consumption and CPU ticks.

The time required by security operations can be evaluated considering the number of bytes that the transceiver could transmit in the same amount of time. In particular, the transceiver of the Z1 mote is able to send 55 bytes in 3.84 milliseconds (time required for creation and encryption of a CoAP request) and 57 bytes in 4 milliseconds (time required for parsing and decrypting a CoAP request). A relevant performance indicator of the DTLS energy consumption is E_{DTLS} defined as the ratio of the total energy spent by both the client and the server for a secured CoAP transaction to the total energy spent for an unsecured CoAP transaction.

TABLE IV: PERFORMANCE PARAMETERS FOR COAP TRANSACTIONS AND OVERALL DTLS OPERATIONS.

	Processing time $[ms]$	Energy consumption $[\mu J]$	Ticks
Unsec Client request	2.70	266	43.2 K
Unsec Client response	0.91	88	14.4 K
Unsec Client transaction	3.81	355	57.6 K
Unsec Server transaction	4.42	436	70.72 K
Unsec CoAP Transaction	/	791	128.32 K
Sec Client request	7.08	698	113.28 K
Sec Client response	5.22	515	83.52 K
Sec Server request	5.82	574	93.12 K
Sec Server response	7.34	722	117.12 K
Sec Client transaction	12.30	1213	196.8 K
Sec Server transaction	13.16	1296	210.24 K
Sec CoAP Transaction	/	2509	407.04 K
Msg creation & encryption	3.84	379	61.44 K
Msg parsing & decryption	4	395	64 K
Overall DTLS	7.84	774	125.44 K

This is calculated as:

$$E_{DTLS} = \frac{E_{sec}^{trans}}{E_{unsec}^{trans}} \cong 3.17$$

Moreover, considering the energy consumed only by the DTLS operations, the estimation of energy consumption related to the SSLP component for a secured CoAP transaction can be calculated as:

$$E_{sec}^{trans}(SSLP) = E_{sec}^{trans} - E_{unsec}^{trans} - 2E_{sec}^{trans}(DTLS)$$
$$\cong 172\ [\mu J]$$

A graphical comparison of the energy consumed (by both client and server devices) is depicted in Fig. 8.

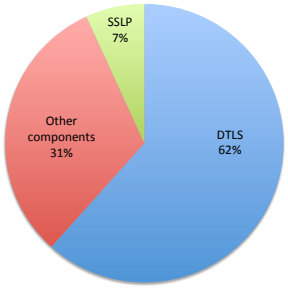
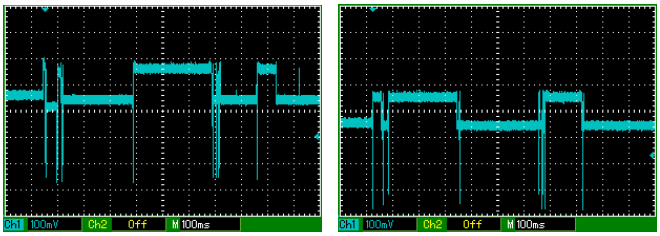


Fig. 8: Energy consumed by different secured CoAP operations.

2) DTLS Handshake: The handshake phase of DTLS protocol consumes considerable amount of energy and we evaluate it in this section. Every time two nodes interact for the first time, or after a long period of time, both of them must spend a certain amount of time and energy to establish or recover a DTLS session. Since the lightweight DTLS implementation considered in this paper does not include mechanisms for expired session recovery, experiments and analysis are applied only to the handshake establishment phase. Fig. 9 shows the voltage values of the sensing resistor for both the client and server devices exchanging handshake messages, while Table V shows the average time, energy consumption and number of ticks required by client and server to handle a full DTLS handshake as defined in [2].



(a) Client. (b) Server.

Fig. 9: Handshake measurements.

TABLE V: HANDSHAKE MEASUREMENTS.

Device	Flights involved	Processing time $[ms]$	Energy consumption $[\mu J]$	Ticks
Client	1	7.8	769	124.8 K
Client	2, 3	14.2	1400	22.72 K
Client	4, 5	302.3	29822	4833.6 K
Client	6	3.2	316	51.2 K
Client	7	72.4	7142	1158.4 K
Client	All	400	39450	6398.4 K
Server	1,2	31.7	3127	507.2 K
Server	3,4	259.9	25639	4158.4 K
Server	5	2.9	286	46.4 K
Server	6,7	142.8	14087	2284.8 K
Server	All	437.3	43140	6996.8 K

The total time required for two nodes to perform a full handshake is a channel-dependent parameter, but the time durations required for frame transmissions are negligible compared to the duration of their overall operations, which are the main source of latency. The average delays caused by the handshake phase during client and server mode of operations are $D_C = 880$ ms and $D_S = 800$ ms respectively. The time and energy required to establish a secure channel are a major source of performance degradation. In fact, the operations of the handshake phase drain an amount of energy that approximately corresponds to the energy consumed by a device exchanging 33 CoAP secured transactions, while for the same amount of time required to establish a secure channel the device could transmit approximately 200 unsecured transactions. Further details on the BlinkToSCoAP framework and its performance analysis can be found in [14].

V. CONCLUDING REMARKS

This paper introduces the BlinkToSCoAP security framework for the Internet of Things and provides an overview of the underlying protocol implementations and TinyOS components and also outlines the relationships that ensure their effective collaboration. In addition, the extensive experiments carried out during the course of this work and their results show that despite the semi-optimized state of protocol implementations and the large number of functionalities involved, all the necessary code and data required to implement the proposed end-to-end security framework using lightweight DTLS protocol within CoAP and 6LoWPAN based IoT applications do not overshoot the amount of ROM and RAM available in the Zolertia Z1 platform.

On the other hand, security operations of DTLS implementation consume considerable amounts of energy and introduce considerable delay every time two nodes begin a new communication or need to recover an expired session. Once the security session has been established, security operations increase the energy consumption of CoAP transmissions by a factor of about 3.2 and affect the responsiveness of the nodes, which have to work for additional 8 milliseconds per CoAP transaction in order to deal with obfuscated data and to transmit the additional DTLS header. These factors need to be considered during the IoT system design while introducing end-to-end security. The context considered here to assess the quality of the proposed framework includes communications between only two USB powered devices. A more realistic situation would involve an operating environment with multiple nodes running on battery supply. In conclusion, this work not only provides a sound framework for implementing end-to-end security in IoT but also provides an extensive performance analysis in terms of memory footprint, packet overhead and energy efficiency, which indicate that introducing lightweight DTLS-based security in IoT is certainly feasible even in Class 1 constrained devices.

VI. ACKNOWLEDGMENT

The authors would like to thank Moreno Dissegna, Mario Emilio Cecconato, Matteo Fiorindo and Giulio Marin for their assistance during the course of this work.

REFERENCES

[1] Z. Shelby, K. Hartke, and C. Bormann, *The Constrained Application Protocol (CoAP)*, IETF RFC 7252, June 2014.
[2] T. Dierks and E. Rescorla, *The Transport Layer Security (TLS) Protocol Version 1.2*, IETF RFC 5246, August 2008.
[3] E. Rescorla and N. Modadugu, *Datagram Transport Layer Security Version 1.2*, IETF RFC 6347, January 2012.
[4] C. Bormann, M. Ersue and A. Keranen. *Terminology for Constrained-Node Networks*, IETF RFC 7228, May 2014.
[5] T. Kothmayr, C. Schmitt, W. Hu, M. Brunig, and G. Carle. *A DTLS based end-to-end security architecture for the Internet of Things with two-way authentication*, in Proc. of IEEE LCN 2012, Clearwater, USA.
[6] J. Granjal, E. Monteiro, and J. Sa Silva. *On the feasibility of secure application-layer communications on the Web of Things*, in Proc. of IEEE LCN 2012, Clearwater, USA.
[7] J. Granjal, E. Monteiro, and J. Silva. *On the Effectiveness of End-to- End Security for Internet-Integrated Sensing Applications*, in Proc. of IEEE GreenCom 2012, online conference.
[8] S. Raza, D. Trabalza, and T. Voigt. *6LoWPAN Compressed DTLS for CoAP*, in Proc. of IEEE DCOSS 2012, Hangzhou, China.
[9] S. Raza, H. Shafagh, K. Hewage, R. Hummen, and T. Voigt. *Lithe: Lightweight Secure CoAP for the Internet of Things*, IEEE Sensors Journal, vol. 13, issue 10, pp. 3711-3720, August 2013.
[10] C. Bormann, *Using CoAP with IPsec*, Tech. rep. CoRE Working Group, 2012.
[11] A. Castellani, G. Ministeri, M. Rotoloni, L. Vangelista, and M. Zorzi. Interoperable and globally interconnected Smart Grid using IPv6 and 6LoWPAN, in Proc. of IEEE ICC 2012, Ottawa, Canada.
[12] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler, Transmission of IPv6 Packets over IEEE 802.15.4 Networks, RFC 4944, Sep. 2007.
[13] *Zolertia Z1 WSN Module Datasheet 2010*, available online at http://zolertia.com/sites/default/files/Zolertia-Z1-Datasheet.pdf.
[14] G. Peretti, *CoAP over DTLS TinyOS Implementation and Performance Analysis*, MS Thesis, University of Padova, Italy, December 2013.