Lightweight DTLS Implementation in CoAP-based IoT

Vishwas Lakkundi and Keval Singh

Altiux Innovations Pvt. Ltd., Bangalore, India

{vishwas.lakkundi, keval.singh}@altiux.com

***Abstract*—Security is emerging as a key area of focus in the Internet of Things. Lightweight implementations of the required security features are the need of the hour considering the resource constrained nature of the underlying nodes and networks. At the same time, it is essential to ensure that such implementations are robust, reliable and efficient. This paper addresses this need by providing a framework for implementing a lightweight version of the Datagram Transport Layer Security protocol in the Internet of Things. In addition, a real-world application scenario incorporating this lightweight security approach is included for illustration in this position paper. It also sheds light on the ongoing standardization activities in the security domain and relevant future directions to help practitioners keep abreast of all the related developments.**

Keywords—Internet of Things; M2M; information security; DTLS; CoAP.

I. INTRODUCTION

The nodes in Internet of Things (IoT) often have constraints regarding their resources such as computational power, memory size and power management. Network communication, especially wireless, also imposes additional restrictions such as low bitrates, variable delays and high packet losses. Due to their pervasive nature, sensitive data can be collected and transmitted from different sources for both public and private use [1]. Consequently, security of transmitted data as well as source authentication are crucial. Security can be provided at different layers of the underlying protocol stack. Typically, application layer protocols often delegate security mechanisms to the transport layer, which helps in achieving end-to-end security. And the overhead due to this security mechanism is very relevant to the overall system performance. One such protocol is Datagram Transport Layer Security (DTLS) [2], which additionally has inbuilt binding within Constrained Application Protocol (CoAP) [3], which is a specialized web transfer protocol intended to be used by constrained devices in IoT. Though DTLS was not designed with lossy networks and constrained devices in mind, it has emerged as a key candidate to provide security in IoT. However, it cannot be employed as is since it is considered to be too heavy for resource constrained environments. Instead, lightweight implementations of DTLS are more suitable for use in IoT.

A typical communication scenario between an unconstrained network and a constrained network is shown in Fig. 1. An unconstrained network is typically represented by the Internet, whereas the IoT consisting of a low power wireless personal area network (LoWPAN) represents the constrained domain.

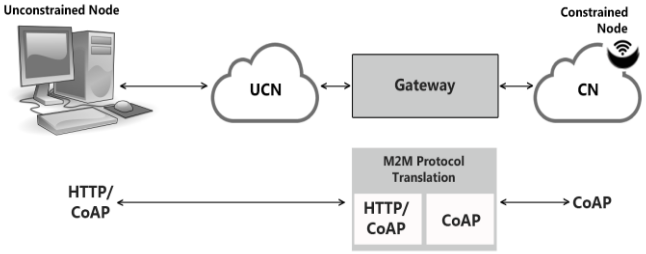

Fig. 1. Networking and communication scenario in IoT.

An IoT gateway placed on the edge between the unconstrained network (UCN) and the constrained network (CN) adapts the communication between these two domains. Its role usually involves the adaptation between different protocol-layer implementations. Also called a border router, it carries out protocol translations vis-à-vis end-to-end IoT security as illustrated in Fig. 2.

Since the gateway is generally an unconstrained device, it can also be used for scaling down the functionalities from the UCN to the CN domain and also for managing security settings in peripheral constrained networks [4]. To maintain the end-to-end approach, the gateway needs to remain invisible to the communicating endpoints. As shown in Fig. 1, a node on the UCN can be either HTTP enabled or only CoAP enabled.

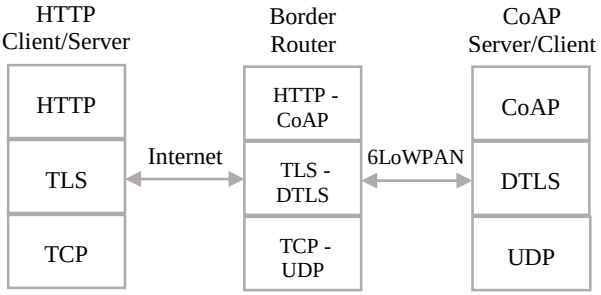

Fig. 2. Architecture for end-to-end security in IoT.

This paper is organized as follows: the next section introduces DTLS and its security features. Section III provides an overview of CoAP and state-of-the-art on lightweight DTLS implementations. Our guidelines on lightweight DTLS implementation techniques for CoAP-based IoT are elaborated in Section IV. It also includes a real-world application scenario that illustrates the integration of different variants of DTLS with CoAP. Section V provides an overview of ongoing standardization activities in the DTLS domain, while future directions are outlined in Section VI.

II. DTLS Overview

A. Transport Layer Security

Transport Layer Security (TLS) offers communication security at the transport layer to protect HTTP applications running on top of TCP. TLS version 1.2 is defined in RFC5246 [5]. This version of TLS offers more flexibility in the sense that ciphersuites that were hardcoded in the Pseudo-Random Function (PRF) in earlier versions are replaced with cipher-suite-specified PRFs. All TLS versions separate authentication and key exchange, and bulk data protection. The former is more expensive in terms of performance and message size. Particulars of authentication and key exchange using the TLS handshake vary with the ciphersuites selected. Once the TLS handshake is established, the necessary keying parameters are setup for use with the TLS Record Layer (RL) that is responsible for bulk data protection. Ciphersuites used for the TLS RL include AES-128/AES-256 with SHA-1 and RC4 with SHA-1/MD5. TLS may also be used without RL as in Secure Real-time Transport Protocol (SRTP) using DTLS (DTLS-SRTP) [6].

TLS was designed for reliable transport protocols, thus it expects no loss or reordering of messages from the transport layer. If a message is lost or appears out of order, it assumes an attack and thus drops the connection. Hence, it cannot be used with unreliable transport protocols that are invariably lossy in nature. This led to the emergence of the Datagram Transport Layer Security (DTLS).

B. Datagram Transport Layer Security

DTLS is derived from, and inherits some characteristics of TLS. It allows re-use of TLS security functionalities on top of User Datagram Protocol (UDP). With the emergence of CoAP as a specialized web transfer protocol for constrained devices, DTLS is the preferred security protocol in IoT.

Like TLS, DTLS also has a base protocol called Record Layer, and four sub-protocols on top, namely Handshake, ChangeCipherSpec, Alert Protocol and the application data protocol as explained in [2]. The required security features for a specific smart object application in IoT depend on various factors such as the underlying communication architecture and the threats to be mitigated [7].

III. State-of-the-Art

A. Constrained Application Protocol

CoAP is a specialized web transfer protocol intended to be used by constrained devices in IoT/M2M applications. It provides a client/server interaction model between application endpoints and includes the same key functionalities of HTTP. For this reason, CoAP can be 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.

B. Lightweight DTLS

IoT nodes often have constraints regarding their resources such as computational power, memory size and power management. Network communication, especially wireless, also imposes additional restrictions such as low bitrates, variable delays and high packet losses. In addition, since frames at the link layer are much smaller than the IPv6 MTU of 1280 bytes, additional adaptation mechanisms such as 6LoWPAN [8] for IEEE 802.15.4 networks is required, which further limits the network capacity. However, application layer protocols often delegate security mechanisms to the transport layer, which helps in achieving end-to-end security. And the overhead due to this security mechanism is very relevant to the overall system performance. One such protocol is DTLS, which additionally has inbuilt binding within CoAP. Though DTLS was not designed with lossy networks and constrained devices in mind, it has emerged as a key candidate to provide security in IoT. However, it cannot be employed as it is since it is considered to be too heavy for use in constrained environments and networks such as IoT. Thus emerged several lightweight implementations of DTLS for use in IoT, some of which are explained below to give us the required insight into the state-of-the-art techniques in this emerging domain.

C. Lightweight DTLS Implementation

Implementation of DTLS could be based on employing any of the following techniques:

- Pre-shared Key (PSK)
- Raw Public Key
- Certificates

PSK based implementation examples are elaborated in the following subsection. However, DTLS implementations based on certificates and raw public keys are currently out of scope of this paper as they are considered to be very heavy for IoT.

1) TinyDTLS

TinyDTLS is a software library that provides a very simple datagram server with DTLS support. It is designed to support session multiplexing in single-threaded applications and thus specifically targets embedded systems. It is distributed under the MIT License [9]. Its salient features include:

- Basic support for DTLS with PSK only
- No support for public key cryptography
- Supports HMAC-SHA256
- Supports Rijndael (AES)
- Supports clock handling, NetQ, PN number generation

2) Lightweight TinyDTLS

This lightweight DTLS implementation is based on the open-source library TinyDTLS, which in turn is ported to Contiki [10]. It has the following salient features:

- Supports AES-128 and SHA-256
- No support for CCM
- No DTLS message fragmentation
- Limited alert protocol
- Not compliant with DTLS IETF RFC 6347
- Compliant with Class 1 device specifications (<10 KB RAM, <100 KB Flash)

3) CoAP over DTLS - TinyOS Implementation

It includes the integration of three libraries that implement lightweight versions of DTLS and CoAP protocols as well as

the IPv6/6LoWPAN stack. It is implemented using the *nesC* programming language in TinyOS [11]. Its salient features are:

- Provides integration of DTLS with CoAP and 6LowPAN
- Defines the necessary interfaces
- Includes DTLS with PSK only
- No support for public key cryptography support
- Supports HMAC-SHA2
- Supports Rijndael (AES)
- Supports CCM
- Supports clock handling, NetQ, PN number generation
- Not Compliant with Class 1 device specifications

IV. Lightweight DTLS Implementation Techniques

The TinyDTLS implementation library consists of the following core modules. Further details on each of these modules can be found in [9].

- DTLS
  - State machine used to establish a DTLS session
  - Handshake protocol definition
  - Structure of different messages used by the protocol
- Cryptography
  - Encryption operations
  - Decryption operations
- Keyed-Hash Message Authentication Code (HMAC)
  - Algorithm used for message authentication
- Counter with Cipher Block Chaining Message Authentication Code (Counter with CBC-MAC or CCM)
  - Actual implementation of encryption function
  - Actual implementation of decryption function
- Rijndael Cipher
  - Implementation of AES
- Secure Hash Algorithm (SHA-2)
  - Implementation of a set of cryptographic hash functions

A. *Protocol Library*

The TinyDTLS protocol library consists of interconnection of several components with the main DTLS module as shown in Fig. 3. It contains all the logic required to handle secure communications including data sessions, handshake protocol definition and structures of different messages belonging to the security protocol.

The Crypto module handles all the authentication and encrypt/decrypt operations. As a lightweight DTLS implementation, the crypto component supports only DTLS_PSK_WITH_AES_128_CBC_SHA-256, which is composed of the pre-shared key exchange algorithm and the 128 bit AES algorithm in CCM mode.

Message integrity and a second check for message authentication is achieved by the HMAC component, which calculates a MAC through the SHA-256 function in combination with a secret cryptographic key generated from the master secret realized during the handshake phase [12].

The Clock Handling module carries out default implementation of the internal clock and the Random Number Generation module generates random numbers, both of which are employed in the computation of secret keys.

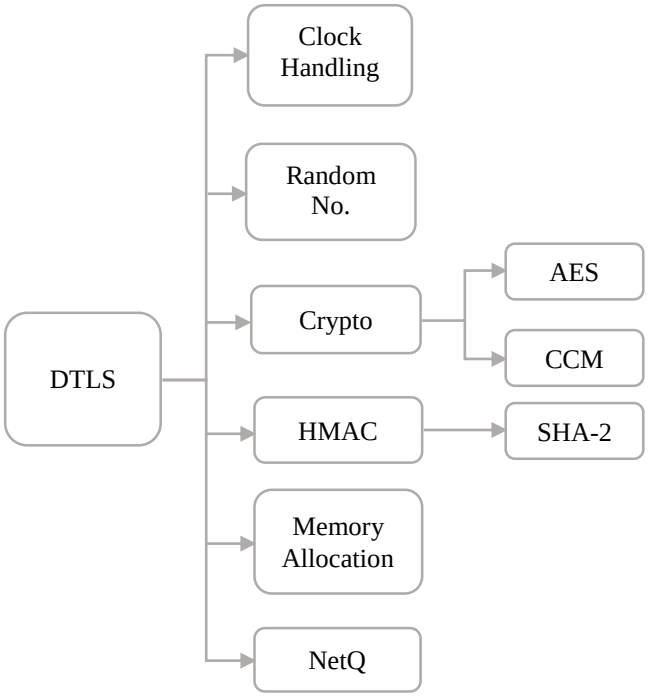

Fig. 3. TinyDTLS protocol library structure.

The Memory Allocation module allocates memory to peers as well as help freeing memory allocated to them. The Network Packet Queue (NetQ) utility functions implement an ordered queue of data packets to send over the network and can also be used to queue received packets from the network.

B. *Interaction Architecture and Interfaces*

The interaction between CoAP and DTLS modules is illustrated in the architecture diagram in Fig. 4. It also depicts the interaction between CoAP, UDP and 6LoWPAN (6LP). In case, no security is desired by the corresponding application, CoAP bypasses its interaction with DTLS and communicates directly with UDP and 6LP instead.

The required interfaces between CoAP and DTLS in both forward and reverse directions of data flow are shown in Fig. 5 and Fig. 6 respectively.

In the forward direction, a CoAP packet is sent to the DTLS module for adding security functionality. This operation needs two interfaces: one for reading normal data packets from CoAP to DTLS and the other for sending encrypted data packets from DTLS. Afterwards, the encrypted data packets are sent across to UDP. These three steps are indicated numerically in Fig. 5.

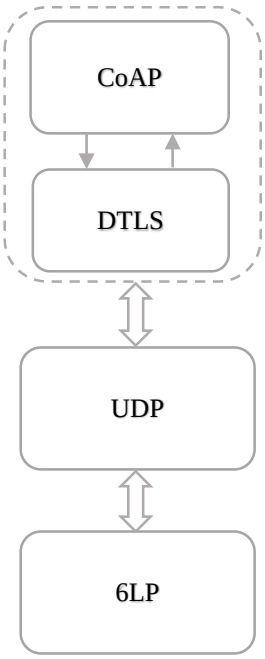

Fig. 4. CoAP-DTLS interaction architecture.

In the reverse direction, secured data packets received from UDP/6LP are sent across to DTLS for decryption. This operation also needs two interfaces: one for reading secured data packets from UDP and the other for sending decrypted data packets from DTLS back to CoAP, which in turn transfers this data to the corresponding application above it.

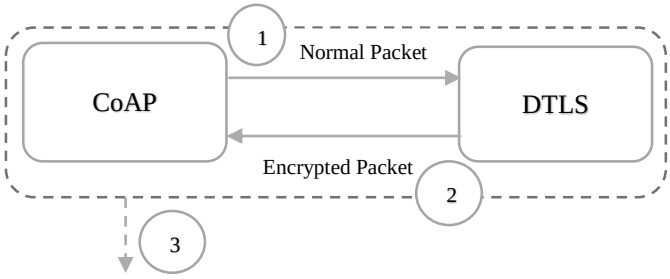

Fig. 5. Encrypting a CoAP packet using DTLS.

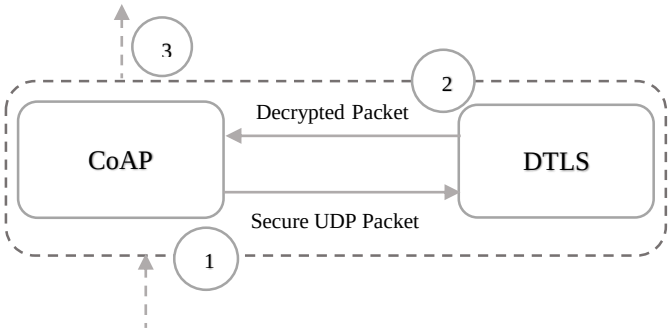

Fig. 6. Sending a DTLS decrypted packet to CoAP.

The various interfaces defined earlier that are employed during the forward and reverse data flow directions are further elaborated below. These interfaces are essentially defined by the TinyDTLS implementation library available at [9]. The minimum configuration required for any useful communication to take place include the creation of the DTLS context, read and write call backs and registration of the key management function.

The pseudo-code for Read Callback interface that is invoked once the DTLS session is established, and the application data has been received, is given in Fig. 7.

```
int read_from_peer(dtls_context, session, data, length)
{
        return dtls_write(context, session, data, length);
}
```

Fig. 7. TinyDTLS read callback interface.

In the pseudo-code for Write Callback interface given in Fig. 8, the callback function *send_to_peer()* is called whenever data needs to be sent over the network. Here, the *sendto()* system call is used to transmit data within the given session.

```
int send_to_peer(dtls_context, session, data, size)
{
  int fn = dtls_get_app_data(dtls_context);
  return sendto(fn, data, size, session->addr, session->size);
}
```

Fig. 8. TinyDTLS write (send) callback interface.

Here, *dtls_context* refers to a specific instance of the DTLS library corresponding to a particular application, *session* refers an active secure connection that has been established, *data* refers to the message to be secured, and *length* refers to the length of the above message.

C. Lightweight DTLS Application Scenario

Our IoT home automation application scenario to demonstrate light control, temperature and humidity sensing on a reference hardware board on the right through a web browser on the left is shown in Fig. 9. It also involves a web layer in the middle, which includes a web gateway service that performs HTTP to CoAP translation. This application involves secure CoAP, meaning CoAP packets are secured using DTLS. For securing the CoAP link between the web and device layers, we employ lightweight TinyDTLS implementation on the device layer and the Scandium DTLS implementation on the web layer.

This implementation is unique since different forms of DTLS are successfully integrated to provide secure CoAP functionality. For total end-to-end security between the browser and the end device, TLS has to be employed between the browser and web layers in addition to the DTLS security mentioned above.

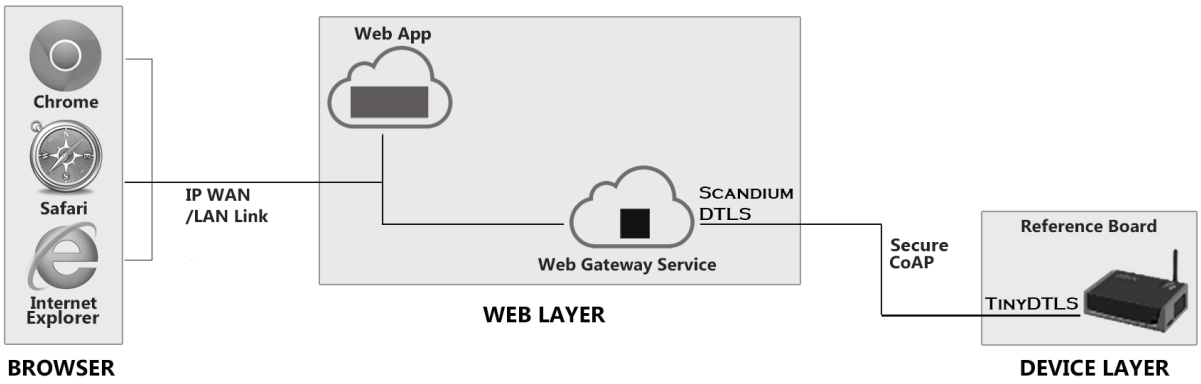

Fig. 9. Application scenario involving CoAP over DTLS.

V. STANDARDIZATION ACTIVITIES

The recently formed IETF DTLS In Constrained Environments (DICE) [13] working group is leading the activities on supporting the use of DTLS in constrained environments with the following tasks:

- To define a DTLS profile that is both suitable for IoT applications and can be reasonably implemented on many constrained devices.

- To define how DTLS record layer can be used to transmit multicast messages securely.

- To investigate practical issues around the DTLS handshake procedure in constrained environments. Many current systems end up fragmenting messages, and the re-transmission and re-ordering of handshake messages results in significant complexity and reliability problems. Additional reliability mechanisms for transporting DTLS handshake messages are required as they will ensure that handling of re-ordered messages needs to be done only once within the stack.

However, DICE does not intend to modify the DTLS state machine. Moreover, key management and multicast session setup are out the scope for the initial work.

There are several ongoing IoT/M2M standardization activities such as OneM2M, ETSI M2M, GISFI, AllSeen Alliance, OIC, ITU GSI, Thread and so on, which are beyond the scope of this paper.

VI. CONCLUSIONS AND FUTURE DIRECTIONS

The requirements for providing end-to-end security in constrained IoT environments such as those based on CoAP and DTLS are quite stringent. Nevertheless, lightweight security implementations still make it possible as illustrated in the previous sections. Taking it a step further, we intend to make available in the near future the detailed performance metrics for our application that employs the lightweight security mechanism described earlier.

To improve the overall performance further, the actual implementation of these security techniques on the underlying hardware plays a key role. In addition, availability of in-built crypto functions such as AES in hardware makes a lot of difference to the memory footprint and energy efficiency, both of which are of great interest to IoT.

Furthermore, it is essential that these implementations are carried out taking into consideration the necessity of making room in the near future for modifications arising out of the ongoing standardization activities such as IETF DICE. Additionally, interoperability capabilities can be looked into through participation in events such as ETSI plug-tests.

Considering the overwhelming importance, relevance and necessity of end-to-end security in IoT, it is considered essential to make provision for session-based security functionalities at different and multiple layers of the protocol stack so that it is both robust and flexible to meet the application requirements.

REFERENCES

[1] D. Altolini, V. Lakkundi, N. Bui, C. Tapaarello, and M. Rossi, "Low Power Link Layer Security for IoT: Implementation and Performance Analysis," in Proc. of IEEE IWCMC 2013, Cagliari, Italy, July 2013.

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

[3] Z. Shelby, K. Hartke, and C. Bormann, The Constrained Application Protocol (CoAP), IETF RFC 7252, June 2014.

[4] R. Bonetto, N. Bui, V. Lakkundi, A. Olivereau, A. Serbanati, and M. Rossi, "Secure Communication for Smart IoT Objects: Protocol Stacks, Use Cases and Practical Examples," in Proc. of IEEE WoWMoM 2012, San Francisco, CA, US, June 2012.

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

[6] J. Fischl, H. Tschofenig, and E. Rescorla, Framework for Establishing a Secure Real-time Transport Protocol (SRTP) Security Context Using Datagram Transport Layer Security (DTLS), IETF RFC 5763, May 2010.

[7] S. Kumar, S. Keoh, and H. Tschofenig, A Hitchhiker's Guide to the (Datagram) Transport Layer Security Protocol for Smart Objects and Constrained Node Networks, IETF Draft (work in progress), March 2014.

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

[9] O. Bergmann, TinyDTLS Software Library Implementation, available at http://tinydtls.sourceforge.net/.

[10] O. Bergmann, S. Gerdes, and C. Bormann, "Simple Keys for Simple Smart Objects," in Proc. of Workshop on Smart Object Security, Paris, France, March 2012.

[11] G. Peretti, CoAP over DTLS TinyOS Implementation and Performance Analysis, MS Thesis, University of Padova, Italy, December 2013.

[12] A. Gifford, HMAC-SHA Implementations, available at http://www.aarongifford.com.

[13] IETF DICE Working Group, http://datatracker.ietf.org/wg/dice/charter.