Compact DTLS 1.2 implementation and a suggestion for improved DTLS-secured multicast topology

Department of Computer Science
Research Unit Distributed and Embedded Systems
Aalborg University

Ivan Kodzhastoyanov
Supervisor: René Rydhof Hansen

June 6, 2016

Abstract

In this paper I present my work on the implementation of a compact version the DTLS 1.2 protocol as well as my suggestions regarding the addition of multicast functionality to the DTLS standard.

In the last decade home automation systems have began to increasingly adopt concepts from the Internet of Things to shape the current state of Smart Home technologies. One crucial aspect, which needs special attention, is the adoption of state of the art security mechanisms. It is important that latest recommendations be followed to keep our modern homes protected against adversaries. This report begins with an introduction of the targeted environment and the concepts, which form the basis for my work. In the following pages I present my experience in developing a compact implementation of the latest version of DTLS. My work is based on latest recommendations and standards from the authorities, involved in the development and maintenance of network technologies. Furthermore, I suggest an improved topology for introducing multicast support to DTLS. This topology reduces the memory requirements and the complexity of the currently suggested approach, which is desired for group communications in constrained environments.

Acknowledgements

I would like to thank my supervisor, René Rydhof Hansen, for the guidance and the constructive advices he gave me through the last year. I also want to thank Daniel Lux and Morten Pagh Frederiksen for providing me with the opportunity to work on this project with Seluxit. Additionally, I want to express my gratitude towards Stefano Martin for aiding me with expert insight of security algorithms, as well as Andreas Bomholtz for providing me with technical insight and advices through my internship period at Seluxit.

Contents

List of Figures

1. Introduction

1.1 Motivation

The Internet of Things market has seen a recent increase in popularity [1] - a trend, expected to continue in the foreseeable future by industry leaders, such as Cisco [1] and Intel [2]. IoT technologies are used on a global scale for environmental, industrial and urban applications as well as on a more personal scale for applications targeting our homes and bodies. People's interest in Smart Homes, the target area of this report, is continuously rising [3]. Individuals report several potential applications that inspire their interest in Smart Home solutions, the most prominent of which is security [21]. Despite of this user requirement, security is a lagging aspect in IoT solutions [61] [40].

The inner-workings of automated systems, within the Smart Home environment, depend on user-to-machine and machine-to-machine interactions in the form of network communications. The devices in a Smart Home environment interact with each other in a secure private home area network(HAN). They can also be controlled over the Internet, through a gateway. However, this introduces an additional set of vulnerabilities to the HAN and requires an extended security strategy. The use of Virtual Private Network(VPN) [4] protocols allows to extend the communication security property of the HAN to legitimate nodes on the Internet. This is achieved by establishing a secure connection between two communicating parties.

TLS [63] is a widely deployed network security protocol, which provides VPN establishment. It is highly flexible, transparent to the user and provides point-to-point security. These properties make TLS a preferred choice for securing connections on the Internet. However, it relies on the use of a reliable transport channel, such as TCP. Due to their low-power, lossy character, IoT and Smart Home environments often use datagram based transport channels, such as UDP.

A datagram based version of TLS, called DTLS, was created to address applications, which utilize datagram based transport channels. TLS and DTLS profiles [66] have been defined to address limitations in constrained environments. However, these profiles do little more than identifying suitable cipher suites for use in the said environments. Furthermore, DTLS does not support useful features, such as multicast, which can be used for communication optimization in constrained environments. IETF's DICE working group [30] has started work on a strategy to enable multicast support in DTLS. Their current draft on the subject details a minimal modification to the existing DTLS specification that can be used to enable secure multicast communication for CoAP based applications. A topology for communications and key management is also described. However, the proposed DTLS modification and communication topology do not provide a particularly scalable approach.

In this report I document my work on the development of a standard DTLS 1.2 implementation for use in constrained environments and the design of what I believe to be a more scalable scheme for introducing multicast support in DTLS. The implementation was created for a project I am developing in cooperation with the software R&D company Seluxit.

1.2 Attack Example

To showcase the importance of communication security and the implications of communicating over unsecured channels I present an attack that exploits the lack of point-to-point communication security. This attack assumes the following setup:

- A Smart Home environment, that includes an automated security system.
- At-home devices are participants in a secure private HAN.
- There is no direct physical access to the devices in the HAN.
- Communications within the HAN are protected - messages are encrypted and authenticated.
- Devices in the HAN are not connected to the Internet.
- Only the gateway is connected to the Internet.
- Devices can be controlled over the Internet through the gateway.

In the assumed setup, the security system monitors for intruders. Communications within the HAN are protected using a HAN encryption key, as shown in Figure 1.1c. An application enables the user to remotely control devices within the home area network over the Internet. The gateway forwards messages between users on the Internet and devices within the home area network. A message path begins at the user operated application. It is sent to a server over a TLS protected channel. The server then uses a VPN connection to the gateway to send it the same message. Once the message is received by the gateway, it is encrypted with the network key for the given home area network. This message exchange is depicted in Figure 1.1a. Once the target device receives the message it decrypts it and reacts according to the incoming message's content. If the device needs to respond, the response it generates is transmitted along the same path as the message that triggered it. In this communication scenario the raw message(and the response, if any) will be protected on it's way from one node to the next and it will be visible at each node on the path.

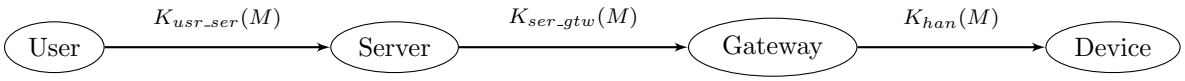
An attacker cannot directly attack devices within the HAN, because they are not physically accessible or directly connected to the Internet and the HAN is encryption protected. However, the gateway serves as a mediator between users and their at-home devices. It is connected to the Internet and susceptible to attacks. An attacker who manages to obtain control over the gateway can discover the HAN encryption key, monitor incoming messages and interpret messages exchanged between nodes within the HAN. This situation has several implications, namely:

- Private information from within the network is visible to the attacker.
- The attacker can repeat selected user messages to control devices.
- The attacker can shut the security system down and freely enter the house.

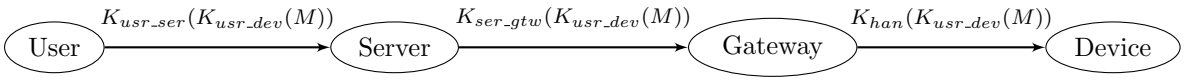
The potential damages range from breach of privacy through robbery to physical harm. A breach of privacy can lead to emotional pain and access to classified information. Financial loss can be caused both by theft an by an attacker manipulating the systems within the house to increase the household's energy consumption and the related bills. Most severely, an attacker can enter the household or cause disasters therein to compromise the resident's well being.

To protect against this threat end-to-end VPN protection can be used instead of the hop-by-hop approach previously presented. The associated message exchange procedure

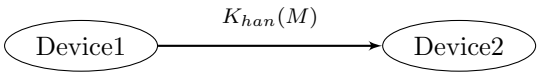
for user-to-device communications is depicted in Figure 1.1b and that for device-to-device communications inside the HAN is depicted in Figure 1.1d. Using this alternative will allow to protect communications between pairs of nodes within the network, which ensures continuous communication security even if the home area network is breached. Furthermore, user-to-device communication will be protected all the way from the origin to the destination point, meaning messages going through the gateway will be encrypted and authenticated, thus protecting against eavesdropping and attempts for message forgery. In both user-to-device and device-to-device connections UDP will be used at least on one end, which means TLS is an incompatible VPN option. DTLS, however, is a good fit for use in the assumed environment.



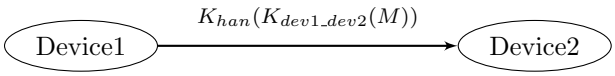
(a) Hop-by-hop Message Protection



(b) End-to-end Message Protection



(c) Home Area Network Message Protection



(d) Home Area Network + DTLS Message Protection

Figure 1.1: Message Protection

1.2.1 Contribution

This thesis documents my work on the implementation of DTLS 1.2 for use on embedded devices in constrained environments. The implementation targets a particular hardware setting, which is described in detail in Chapter 2. My work includes an analysis of the following:

- RAM and ROM usage: both are scarce in embedded devices. One of the goals for this project is to analyze DTLS profiles and implementation opportunities, which would allow to reduce the RAM and ROM requirements for the implementation to a minimum. Possible DTLS profiles will be selected based on the targeted device's specifications as well as compatibility with other DTLS implementations.

- Message and record sizes: In constrained environments and with lossy networks the amount of the transmitted information is restricted. To assure messages can be delivered I assume the use of a minimal message size and analyze the implications on the construction and fragmentation of actual DTLS records, generated for the selected profiles. Additionally, I provide suggestions for the use of the implementation, which can reduce its memory usage.

- Performance implications of using DTLS: Secure connections introduce a noticeable computation overhead. When the available processing power is low, this overhead has even greater impact. To analyze the consequences of using DTLS I look at the code execution time for both the handshake and application data transmissions.

- Design of schemes for muticast support with emphasis on key exchange.

In addition to the standard DTLS 1.2 implementation, my work includes analysis of existing schemes for including multicast support to DTLS as well as the design of an alternative topology for DTLS secured multicast connections.

My work derives from official recommendations for profiling DTLS for use in IoT applications [66]. It is also guided by suggestions and results from related projects as well as the properties and the limitations of the target device. The most considerable contribution of this project is the suggested use of recently introduced DTLS extensions to further improve the handshake process and the proposed scheme for enabling multicast support in DTLS. This scheme is inspired by the IETF draft on the subject [31]. The shortcomings of the approach in the draft are presented and alternatives are proposed to overcome them.

1.2.2 Related Work

There has been an ongoing research into end-to-end security protocols for the IoT. Multiple articles present schemes for implementing DTLS for that purpose.

In their article: 'DTLS based Security and Two-Way Authentication for the Internet of Things' [22], Th. Kothmayr,C. Schmitt,W. Hu,M. Brunig and G. Carle present what they claim to be the first fully implemented two way authentication security scheme for the Internet of Things. In their paper, the authors present a DTLS based scheme which uses RSA and x.509 certificates to authenticate both communicating parties. It utilizes cipher block chaining (CBC) mode of operation for message encryption and relies on the assistance of a Trusted Platform Module(TPM) with hardware support for the RSA algorithm. The TPM is also said to provide a tamper proof generation and storage of RSA keys. This paper provides exhaustive evaluation of the resource consumption and the message round-trip time. The authors also suggest that the use of Authenticated Encryption with Associated Data block cipher mode of operation can further reduce the size of encrypted messages.

In the paper 'On the feasibility of secure application-layer communications on the Web of Things' [47], the authors J. Granjal, E. Monteiro, and J. Silva, evaluate the energy consumption, the memory footprint and the computational and packet overhead of employing DTLS in various security modes. This paper concludes that small memory space and the absence of Elliptic Curve Cryptography hardware support have impact on the compatibility of IoT networks with existing public-key certification infrastructures. The paper also identifies some viable cypher suites for network applications that allow a level of compromise for security and resource usage.

In the related paper 'On the effectiveness of end-to-end security for Internet-integrated sensing applications' [46], J. Granjal, E. Monteiro, and J. Silva, consider application and network layer end-to-end network security for Internet-integrated sensing applications. In this paper the authors compare the overhead and energy consumption of using various cipher suites. While the use of security strategies impacts the lifetime(how long the battery lasts) expectancy of the applications, some approaches display favorable performance. One of the two most promising solutions seems to be CoAP over DLTS using TLS_PSK_WITH_AES_128_CCM_8 as a cipher suite. The authors' conclusion is that as long as applications are able to accept compromises between security, communication rates and resources usage, end-to-end security is viable at the network and application layers. The authors mention the notable difference that network layer security enables end-to-end security irrespective of the application.

Apart from the official IETF draft [31] on the subject, multicast support in DTLS seems to be a topic in its infancy. In my research on the subject I discovered a handful of papers addressing aspects of multicast DTLS support. All of them suggest some sort of improvements over the IETF suggested approach. In [34], Marco Tiloca proposes a

more efficient way(as opposed to the IETF approach) to respond to multicast messages by reusing the same response key for all listeners in the multicast group. In his paper, the author describes how the use of a group ID allows senders and receivers in the multicast group to connect without previous knowledge of each other. [29] discusses an existing weakness in [34], where the same nonce-response encryption key pair can be reused by different listeners, which compromises the security of using CCM authenticated encryption mode. The author, Kirill Nikitin, describes a way to use a listener's IP address and a part of the related port number to generate distinct listener response keys. In [55], Nikita Martynov compares the use of ECDSA and TESLA for source authentication and the AMIKEY as a group key management protocol.

Finally, there is the tinyDTLS project [5]. It is a library that provides a bare minimum implementation of DTLS. It is designed for use with CoAP and supports the recommended cryptographic primitives [31] [66] [58] for that purpose. Its goal is to provide a compact DTLS implementation for both clients and servers. Due to the fact it is an open source project and because it shares the same objectives as my project, I have chosen to use this library as a reference implementation and source of inspiration.

1.2.3 Approach

The topic for my project was set at a meeting I had with the CEO of Seluxit - Daniel Lux. At this meeting we discussed the company's need for security protocol implementation in their stack of solution to provide secure communication between devices in a network. The project aligned well with the requirements for my master thesis work. Thus the topic of my master thesis was selected. I spent last semester researching the subject. Initially I looked into existing standards and protocols that target network communication security as well as the authorities involved into the process of developing and evaluating such standards and protocols. As a result DTLS was identified as an appropriate choice of security protocol. Next, I studied the protocol and the security primitives involved in it. The required implementation was to target embedded devices and constrained environments and to provide support for multicast connections. This meant that the implementation's memory footprint and performance were key criteria. In order to properly design the code I looked into existing open source solutions and read several reports covering the process of implementing DTLS for similar use. I also considered IETF drafts, containing advices for DTLS implementations targeting IoT. This helped me identify potential cipher suites and implementation strategies. Additional details of Seluxit's existing stack enabled me to make my final design decisions.

Through the implementation phase I drew inspiration from the open source library tinyDTLS. I used this library mainly to identify implementation blocks as well as required algorithm support.

1.3 Outline

The structure of this paper is as follows: Chapter 2 introduces the project's scope, requirements and goals as well as a specification of the targeted hardware. Chapter 3 introduces fundamental concepts, used in later chapters. Chapter 4 introduces a detailed description of the DTLS 1.2 protocol including the related data structures and the handshake message flow. It also addresses DTLS's use in constrained environments and the existing suggestions for providing multicast support in DTLS. Chapter 5 presents general details of my standard DTLS 1.2 implementation as well as suggestions for it's use, which can reduce the memory usage. Chapter 6 begins with identifying shortcomings of currently suggested approaches for introducing multicast support in DTLS. Afterwards, the chapter introduces an alternative topology for DTLS secured multicast communications. Chapter

7 introduces the tests I have performed so far. Chapter 8 discusses possibilities for future improvements of the implementation. Security considerations are presented in Chapter 9.

2. Project Description

This paper is based on my research and work for a project I developed in cooperation with Seluxit, a software company that develops solutions for the Internet of Things. More specifically, they concentrate on the development of a protocols stack for IoT applications, based on latest and promising technologies. Furthermore, they have developed a radio module for use on smart devices, which is meant to run the protocol stack. Last, but not least, Seluxit develops a user interface, which enables owners of products, based on the Seluxit IoT platform, to interact with the devices in their surrounding environment.

2.1 Project Scope

The objective for this project was to develop a DTLS implementation for use on embedded devices in constrained environments. The exact target environment is Smart Home applications. As documented by the European Union Agency for Network and Information Security(ENISA) in their publication "Threat Landscape for Smart Home and Media Convergence" [64] Smart Home environments have a wide range of related threats. Those fall in 9 categories, namely: Legal, Physical Attacks, Unintentional(Accidental) Damages, Disasters, Damage/Loss of IT Assets, Failures/Malfunctions, Outages, Eavesdropping/Interception/Hijacking, and Nefarious Activity/Abuse. Threats in the last two categories target and exploit vulnerabilities in data exchange originating from or addressing devices in the Smart Home network. In the related report "Security and Resilience of Smart Home Environments" [56] ENISA advices that the latest versions of TLS and DTLS be used to protect network communications from the related threats. Thus, the project and the related implementation address the Eavesdropping/Interception/Hijacking, and Nefarious Activity/Abuse threat categories. Threats outside this scope are not considered. The entire range of documented threats from the report, grouped in categories, can be seen in A.

2.2 Project Requirements and Goals

The implementation is meant to secure communications between CoAP based applications. It is also a requirement that I explore schemes and formalize a strategy for enabling multicast support in DTLS, considering the structure and limitations of Smart Home environments. Based on the target environments and the specifications of the underlying hardware the project has the folowing requirements:

- The implementation should have a small footprint. The radio module that is meant to run the implementation has a 256kB flash storage, most of which is already used by the rest of the system. The exact available storage will depend on the system's configuration, but it will be around 30-40kB. As much of the available memory as possible should be left for the application implementation. In terms of RAM, the radio module is equipped with 32kB. Again, as much of the RAM as possible should be available for the application. This imposes a RAM usage limitation for the implementation of about 5kB.

- The implementation should introduce minimal computational and communication overhead. The radio module is equipped with a 32-bit ARM processor with maximum speed of 80MHz. This imposes noticeable time cost when performing complicated tasks, such as public cipher operations. The module communicates via a

low-current transceiver, providing low-frequency radio transmissions. It is, therefore, preferable that transmitted packets have minimal size.

- The implementation should support only the latest standardized version of DTLS - version 1.2. If properly implemented, DTLS 1.2 libraries should be resilient to weaknesses of older versions of TLS and DTLS.

- The implementation should be compatible with other existing implementations, which support the same protocol version and cipher suites. Seluxit allows for third party software to communicate to their IoT platform. In such cases the user may try to connect to devices using a different DTLS implementation.

Based on these requirements, I have set the following goals for my work on the project:

- Explore details about DTLS 1.2 - what it does, how it works and what it needs to work.

- Explore recommended cipher suites for use in IoT and constrained environments and analyze their impact on performance, code size and RAM usage. Only cipher suites, which use currently supported encryption algorithms(AES and RSA) will be considered.

- Develop a standard unicast DTLS 1.2 implementation.

- Research existing schemes for enabling multicast support in DTLS. Develop a strategy for incorporating multicast functionality in the existing DTLS implementation, based on the characteristics of the target environment - Smart Homes.

2.3 Non-goals

The project targets CoAP applications. DTLS is the must implement protocol for securing CoAP [58]. While other security protocols may be used as well [70], DTLS support is mandatory. Thus, exploring alternatives is an addition to DTLS support, rather than a requirement.

The project concentrates on the implementation of DTLS, rather than the use of existing solutions. Licensed libraries are not considered, due to the added cost of using them. Open source libraries could have been considered, however, only a handful of them target constrained environments. Furthermore, to my knowledge, at the time I am writing this report no DTLS implementation, licensed or otherwise, claims support of multicast. Therefore, the use of existing libraries is not a goal for this project. I did, however, explore some open source libraries, written in C, for inspiration. More precisely, I used tinyDTLS [5] as a reference implementation.

3. Background

3.1 CoAP

The Constrained Application Protocol [58] is a specialized web transfer protocol, designed for machine-to-machine applications. It is intended for use on constrained nodes in constrained(low-power, lossy) networks. CoAP provides a request/response interaction model between endpoints and supports key concepts of the Web, as well as specialized requirements such as multicast support. It is similar in purpose and functionality to HTTP, and is often described as a compressed and optimized adaptation of it. CoAP was designed to easily interface with HTTP, which enables for integration with the Web. It is selected by the IETF as the application protocol for use in IoT.

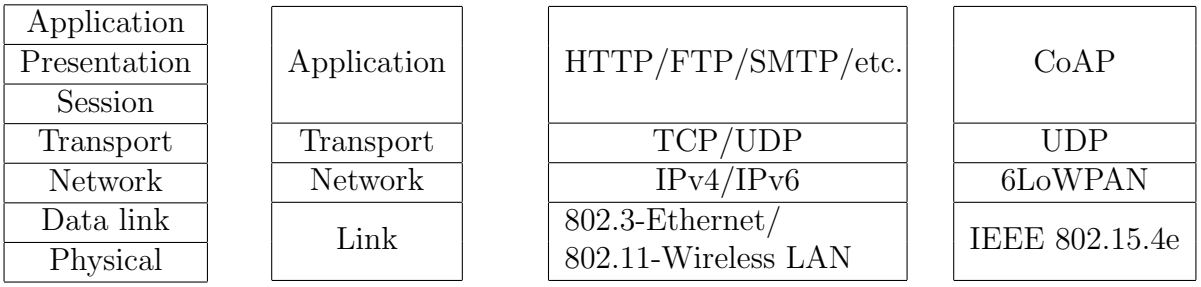
3.2 UDP

The User Datagram Protocol [48] was defined to provide datagram based communication in interconnected computer network environments. The protocol is transaction oriented and provides an unreliable communication channel, e.g. it does not guarantee delivery and duplication protection. It is a good fit for applications where real time communication is of higher priority than reliability of data transfer or where the overhead of maintaining a reliable connection is undesirable. An exemplary application environment for UDP is normal and video telephony, where recovering lost packets introduces delays and breaks the desired real-timeliness property of the communication. Another example of an application field for UDP, more relevant for this report, are home automation systems. Such systems are also designed to react to real time system state(sensor readings, etc.). They often comprise of embedded devices in a constrained environment, where maintaining a reliable transport channel introduces a communication overhead and may impact device lifetime(due to repeated transmissions and prolonged awake periods for battery driven devices).

3.3 IoT

The Internet of Things, as presented in Chapter 1 in [69], is an extension to the existing Internet infrastructure, which targets machine-to-machine communications. Its current state and ongoing development in the field revolves around the characteristics of embedded smart devices as participants in a network as well as continuous, autonomous communications between such devices. As stated in [72], sensors and actuators, communication protocols, and people and processes together drive the development of a digital nervous system, that is the IoT. Like a living organism, this nervous system can sense and react to the surrounding environment. IoT technologies are expected to find application in several areas [6], such as personal health, our homes and cities, the industry and the environment. Cost, efficiency and performance [7] [8] [73] [27] are major factors for the adoption of IoT solutions. To meet the cost requirements, automated systems are composed of specialized, energy efficient and relatively inexpensive hardware [73]. Special protocols have been developed to increase the efficiency and performance in such systems. Figure 3.1 visually depicts the logical layering of functions involved in network communications as well as actual protocol hierarchies used today. 3.1a shows the open system interconnection(OSI) model [20], which conceptually separates the functionality involved in data

exchange/communication in top-to-bottom manner. 3.1b presents the layering used for the Internet [9], where the Application layer is responsible for the functionality in the first 3 layers in the OSI model and the Link layer is comprised of functions related to the last two layers in the OSI model. 3.1c depicts commonly used protocols in standard Internet applications [10]. Finally, 3.1d presents the protocols, selected for the Internet of Things [10] [41]. CoAP provides application layer functionality, similar to HTTP, but in a more compact and efficient way. UDP provides datagram based transport, which is suitable for use in IoT, due to characteristics of related networks(low power lossy networks, composed of memory and processing power constrained nodes). The IEEE 802.15.4 standard allows for low-cost, low-speed, limited range communications, while 6LoWPAN provides compression and optimization mechanisms that allow IPv6 packets to be sent over low power networks.



(a) OSI layers (b) TCP/IP layers (c) Internet protocol suite (d) IoT protocol suite

Figure 3.1: Logical Network Communication Layers and Protocol Suites

IETF has selected DTLS as the must implement security protocol for protecting communications in constrained environments, utilizing the IoT protocol stack [66] [58]. If needed, CoAP can also be secured with IPsec [57] [70]. However, in such cases both DTLS and IPsec support should be present in the system, which will expand the code size. Code footprint is critical for embedded devices. A preferred strategy would be to use a single network communication security protocol. This is why there is an ongoing effort to profile DTLS for use in constrained environments and to provide support of useful CoAP functionality, such as multicast, in DTLS [31].

Classes of IoT devices

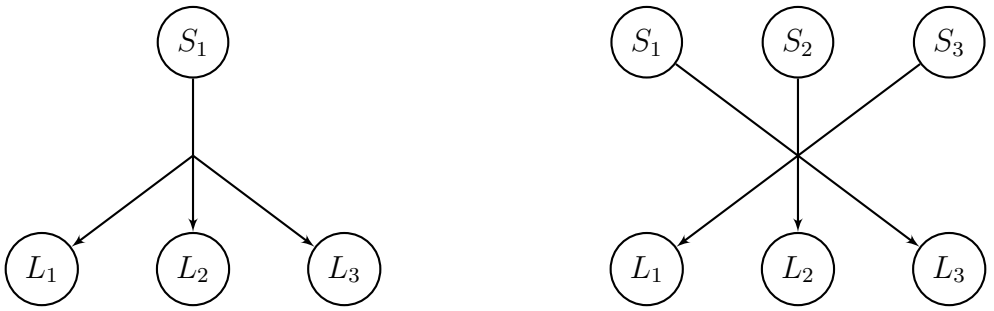
Depending on the exact application environment, IoT devices can be categorized by their roles or by the functionality that they provide. For the purposes of this report, it makes more sense to categorize devices based on their hardware capability. In Section 3 in [60] and Section 2.1.2 in [56] 3 classes of constrained devices have been identified. Figure 3.2 depicts the 3 classes along with their RAM and ROM capabilities and examples of devices in each class. Section 2.1.2 in [56] highlights the implication of the devices' hardware capabilities in reference to security. It states that only class 2 devices have the capacity to implement standard security protocols.

Class	RAM Capacity	ROM Capacity	Exemplary Device
class 0	$<$ 10KiB	$<$ 100KIB	Low-end sensors
class 1	$\approx$ 10KiB	$\approx$ 100KIB	Smart bulbs Smart locks
class 2	$\approx$ 50KiB	$\approx$ 250KIB	Smart appliances, high-end smart sensors

Figure 3.2: Classes of IoT Devices

3.4 IP Multicast

IP multicast [see section 3.1 in 31] [70] is a type of group communication where data is sent to a group of network nodes simultaneously. In such communications there are two roles - sender(s) and listeners and data is exchanged in a request-response fashion. When a sender wants to transmit a request, they encapsulate it in a packet and set the packet's destination to an IP address, allocated for multicast communications. Listeners, on the other hand, check for incoming packets on the port associated with the multicast IP address. Multicast communications may have one-to-many or many-to-many(multiple senders) distributions. Figure 3.3 depicts both distributions, where S_i stands for an instance of a sender, L_j stands for an instance of a listener and $i, j \in \mathbb{N}$. 3.3a depicts a multicast setup with one sender and three listeners and 3.3b depicts a multicast setup with three senders and three listeners. In principle, any node can become a sender or a listener. Nodes can register with a network routing device and join a multicast group, effectively becoming a listener. Senders are not notified of the addition of listeners in the group. If a multicast communication is to be secured, several requirements should be met



(a) One-to-many Distribution

(b) Many-to-many Distribution

Figure 3.3: Multicast Distributions

[see section 2.2 in 31]. First and foremost, the exchanged messages should be encrypted and authenticated to prevent outsiders from interpreting or modifying them. Second, new members of the multicast group should be authenticated and should not be able to interpret old messages. Similarly, members, who leave the group should not be able to interpret future messages, exchanged within the group. Third, individual responses should only be interpretable by the sender, who requested them.

Multicast communications can provide efficiency when a sender has to send the same request to a group of similar devices. It also provides better synchronization, because sending multiple individual requests introduces greater delay between the response of the first and the last receiver. [31] mentions three exemplary use cases for multicast, namely - lighting automation, parameter update and device and service discovery.

For lighting automation the use of multicast can provide a better synchronized response to the action of turning a group of lights on and off or changing their dimness level.

For parameter updates, the use of multicast allows to avoid redundant packets when updating the settings of similar devices.

For device and service discovery, multicast can be used to request information about the devices in the local network and their availability.

Multicast improves efficiency by both reducing the amount of transfered information as well as the repetition of related operations. This can be especially important when requests are encrypted, because in a multicast group with N receivers encryption will be done once instead of N times.

3.5 Security Essentials

The goal of network communication security is to enable two communicating parties to exchange information in a way that no third party can interpret or modify that information. There are three underlaying objectives to achieving that goal, namely - peer authentication, data confidentiality and data integrity.

Data Confidentiality: Data confidentiality is achieved by encryption via the use of symmetric ciphers, such the Advanced Encryption Standard(AES) [23]. Symmetric ciphers are stream or block based. Both types require encryption keys, but while stream ciphers usually expand the key to match the plain text size, block ciphers use a fixed size keys and encrypt the plain text iteratively, one key-sized piece at a time. The exact approach of block cipher encryption may vary and is called block cipher mode of operation [51]. Popular modes of operation are cipher block chaining(CBC) [section 6.2 in 51] and counter(CTR [section 6.5 51]) with CBC-MAC(CCM [52]) modes, where the latter combines data encryption and authentication.

Data Integrity: To ensure that data is not modified on its way from one peer to another, messages are accompanied with a corresponding message authentication code(MAC) [43]. A MAC is a fixed-size checksum, generated from the message using a one way function. When a message is received, its MAC is validated by running the same function on the message and comparing the result to the received MAC. If the message was modified on its way to the destination, this comparison fails. MACs are typically generated using a keyed hash function, HMAC [37], or a block cipher, CMAC [53].

Peer Authentication: Peer authentication is performed to ensure that the two communicating parties are legit and authorized to connect, and that a third party is not impersonating either of them. It is often achieved via the use of public key cyphers(aka asymmetric cyphers) [50], which use a mathematically dependent public-private key pair. When one of the keys is used for encryption the other is used for decryption. The private key is always kept secret by its owner, while the public key is handed to peers who wish to communicate in a secure manner with the private key owner. Since the private key is known only by its owner, using it to encrypt data serves as a distinct signature of that owner. Additionally, if a peer successfully decrypts public key encrypted data, it effectively proves it owns the corresponding private key. To authenticate each other peers often use certificates [42], containing their public key. Such certificates are digitally signed(private key encrypted) by a known and trusted Certificate Agency. Alternatively, the two parties may posses a unique pre-shared secret, aka pre-shared key(PSK), which serves as a proof that these two sides are authorized to communicate. The PSK is distributed to the two parties via a secure channel prior to any communications between them.

Both message encryption and message authentication rely on algorithms, which use keys. A crucial step for establishing secure message exchange is the distribution of such keys. These can be either securely distributed before the peers begin communication or exchanged in the beginning of their communication via key exchange schemes.

Key exchange schemes may involve the use of asymmetric ciphers. A popular key exchange scheme is the use of RSA encryption [11], based on exponentiation and the modulo operation. This scheme assumes that each peer has a unique key pair. A peer will provide its public key to other peers, who want to begin secure message exchange. Upon receiving a public key, peers can use it to encrypt a symmetric key or a secret and send it to the owner of the corresponding private key. This way only the two parties know the actual encryption key.

Another popular key exchange method is the Diffie–Hellman key exchange [12]. Like RSA, it is based on exponentiation and the modulo operation. It works by allowing two peers A and B to select common base n and modulus m. Then A and B each select a private number a and b, respectively. Next A calculates $n^a mod\ m$, B calculates $n^b mod\ m$, and A and B exchange those values. A and B can now calculate $n^{b^a} mod\ m$ and $n^{a^b} mod\ m$, respectively, which results in the same secret value at both ends.

Specific schemes for key exchange, peer authentication, message encryption and authentication, as well as use of symmetric and asymmetric ciphers and hash functions are defined and specified by security protocols. The approach used in DTLS is presented in Chapter 4.

Random Numbers

An essential part of cryptography is the generation of random numbers. They are used as encryption keys or as material for the generation of such keys. An encryption key can be guessed by an attacker by exhaustively trying all possible keys in a given key space. If a 128bit key is used for encryption, an attacker can guess what that key is in 2^{128} tries. This is an infeasible task even for modern computers. However, if keys are selected in a predictable manner they can be uncovered a lot faster. Therefore, it is important to use unpredictable random numbers when generating keys and key materials.

Cipher Suites

Cipher suite is a notion used in TLS and DTLS and refers to a collection of algorithms used for encryption(including the block cipher mode of operation if applicable), MAC generation, random number generation and key exchange. Cipher suites define the exact function to use for each of these operation as well as the size of the key used for them.

Cipher Block Chaining Mode

Cipher Block Chaining(CBC) [Section 6.2 in 51] is a mode of operation, where a plain text is encrypted using a block cipher, $CIPH$, a key, K, and an initialization vector, IV. In this mode the plain text is treated as a sequence of one or more blocks, say B_0 to B_n, each as big as the block size for the used cipher. If needed, the plain text is padded at the end to force its size to be an integer multiple of the block size of the used block cipher. The IV is the same size as that of the block size for the used block cipher. The cipher text, C, consists of the sequence of blocks C_0 to C_n, where:

- $C_0 = CIPH(K, (IV \oplus B_0))$, and
- $C_i = CIPH(K, (C_{i-1} \oplus B_i))$, where $1 \leq i \leq n$ and $\oplus$ stands for the XOR operation.

The plain text can be recovered from C as follows:

- $B_0 = CIPH(K, C_0) \oplus IV$, and
- $B_i = CIPH(K, C_i) \oplus C_{i-1}$, where $1 \leq i \leq n$

Counter Mode

In Counter(CTR) mode of operation [Section 6.5 in 51] a plain text is encrypted using a cipher, $CIPH$, a key, K, and a nonce, N. A sequence of counter blocks, CB_0 to CB_n, is generated, such that $CB_i = N||i$, where $||$ stands for concatenation and $0 \leq i \leq n$ is the counter value. The number of counter blocks, n, will depend on the size of the plain

text. More precisely, $n = \lceil \frac{Plen}{Blen} \rceil$, where $Plen$ is the size of the plain text and $Blen$ is the block size for the used block cipher. The cipher text, C, consists of the first $Plen$ bytes in the block sequence C_0 to C_n, where:

- $C_i = CIPH(K, CB_i) \oplus P_i$, where P_i is a block from th plain text, which starts at the $(i * Blen)^{th}$ byte in the plain text

Decryption is performed the same way, only the encrypted counter blocks are XOR-ed with the corresponding block from the cipher text, C.

Counter with CBC-MAC

The CCM mode of operation is an instance of what is known as authenticated encryption. In short, authenticated encryption uses a symmetric block cipher to both authenticate and encrypt a message. CCM, in particular, dictates that the same block cipher is used once in CBC mode to produce a tag from the original plain text and once in CTR mode to encrypt the message and the tag. This mode has been defined with AES 128, where 128 stands for the key size in bits of the underlying block cipher - AES. The mode requires a plain text P, a nonce N, optional additional data A, an encryption key K and the desired tag(aka MAC) size $Tlen$. P, N and A are used to produce a formated plaint text FP. FP is structured as a sequence of blocks B_0 to B_n, see Figure 3.4. Each of these blocks will be the same size as AES 128 operates on, namely 16 bytes. B_0 contains a flag byte, followed by N, followed by the byte-wise size of P, denoted as $Plen$. The flag byte provides 3 pieces of information: the presence of additional data, $Tlen$ and the number of bytes used to encode $Plen$, denoted as p. If additional data is present, bit 6 in the flag byte has value 1, otherwise 0. $Tlen$ is limited to even values between 4 and 16. Bits 5, 4 and 3 of the flag octet encode $Tlen$ in the form $(t-2)/2$. This way the value 16, for example, is encoded as 7(or 111 in binary). p is limited by the byte size of N, denoted with n. Their combined length is 15(the number of bytes in B_0, excluding the flag byte). n is defined to be from 7 to 13, inclusive. Thus p is $15 - n$. bits 2, 1 and 0 in the flag byte encode p as p-1. Bytes 1 to $15 - p$ in B_0 contain N. Bytes 16-p to 15 in B_0 contain p. Blocks B_1 to B_m are only present if there is additional data. When that is the case B_1 contains A's byte length, denoted as a. If $a \leq 2^{16} - 2^8$, then a is encoded in 2 bytes. If $2^{16} \leq a \leq 2^{32}$, then a is encoded in 4 bytes and preceded by the value 0xfffe(hexadecimal for 2 byte encoding of 65534). If $2^{32} \leq a$, then a is encoded in 8 bytes and proceeded by the value 0xffff(hexadecimal for 2 byte encoding of 65535). A is encoded in the remaining bytes in B_1 and as many additional blocks as needed. If needed, padding bytes are added to the combined encoding of a and A to make it an integer multiple of the block size. Blocks B_{m+1} to B_n contain P and as much padding as needed to make P's size an integer multiple of the block size. Once FP is generated it is CBC encrypted with K as an encryption key and B_0 as an initialization vector. The last encrypted block serves as an intermediate MAC. Next a sequence of counter blocks C_0 to C_r, where $r = \lceil Plen/16$(the block size in bytes)$\rceil$, are generated. The first byte in each counter block contains only p's encoding the same way as B_0. Bytes 1 to $15 - p$ contain N. The remaining bytes include a counter. The counter's value starts at 0 for C_0 and is incremented for each following block. C_0 to C_r are encrypted with K to produce the encrypted blocks S_0 to S_r. S denotes the concatenation of blocks S_1 to S_r in this order. The final result is P XOR-ed with the first $Plen$ bytes in S followed by the intermediate MAC XOR-ed with the first $Tlen$ bytes of S_0.

Secure Hash Algorithm

The Secure Hash Algorithm(SHA) [see 54] is a popular hashing function, used for various purposes, including securely storing passwords and MAC generation. There are 3 distinct

Formated payload								
Block number	B0			B1 to Bm			Bm+1 to Bn	
Contents	Flags	N	P byte size	A byte size	A	A padding	P	P padding

Flags byte								
Bit number	7	6	5	4	3	2	1	0
Contents	Reserved	A flag	(t-2)/2			q-1		

Figure 3.4: CCM formated plain text

variations of SHA, referred to as SHA, SHA-2 and SHA-3. SHA is considered obsolete, while SHA-3 is still being standardized. SHA-2 is widely used nowadays and comprises a family of functions, which vary in their maximum input size and their output size. They all use six internal functions which operate on 3 words, x, y, and z, at a time. For SHA-224 and SHA-256 x, y and z are 32 bit long and for SHA-384, SHA-512, SHA-512/224 and SHA-512/256 they are 64 bit long. In the context of DTLS 1.2, SHA-256 is used for generating pseudo-random byte material. Some cipher suites use SHA and SHA-2 for MAC generation and in signing procedures as well.

Advanced Encryption Standard

The Advanced Encryption Standard(AES) [see 23] is a symmetric block cipher, which is specified for use with 128, 192 and 256 bit long keys. Regardless of the key size, AES operates on 128 bit long blocks. The AES encryption procedure is performed in several rounds, where the number of rounds depends on the size of the key. For 128 bit keys the number of rounds is 10, for 192 bit keys it is 12 and for 256 bit keys - 14. AES is widely used nowadays. Even when used with the smallest defined key, namely 128 bit, it provides a satisfactory level of security for the foreseeable future. Furthermore, it is often supported on hardware level, which greatly improves the performance when employing the algorithm.

4. DTLS Overview

As the name suggests, Datagram Transport Layer Security [28] is a network communication security protocol. It enables two parties to establish a secure session and protect data exchange over the corresponding connection. As a security protocol DTLS aims to achieve the four security objectives - confidentiality, message integrity, peer authentication and non-repudiation. To do this, the protocol employs the related operations for peer authentication, encryption and MAC generation. DTLS is flexible in design and supports various ways of doing each of these operations. The two communicating parties have to ensure they both support the same algorithms and agree on which of them to use. They both also need to generate matching keys for use with the selected algorithms. This is known as session negotiation and is achieved through a process called a handshake. The outcome of this process is a secure private session between the two communicating parties.

4.1 DTLS Structure and Sub-Protocols

DTLS is composed of 4 sub-protocols - Handshake, Application Data, Alert and Change Cipher Spec protocols. Their dependency is depicted in Figure 4.1. The Record Layer

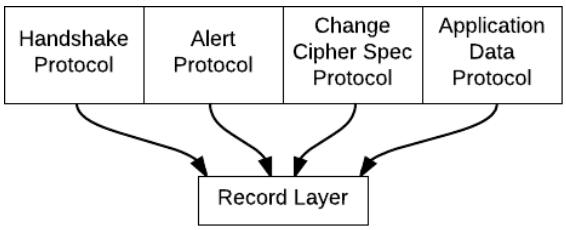


Figure 4.1: The DTLS Sub-Protocols

is the part of DTLS, which fragments, compresses and encrypts messages, includes them in records and passes them further down the communication stack for transmission. The Handshake, Alert, Change Cipher Spec and Application Data protocols produce messages and pass them to the Record Layer. The structure of DTLS records is as shown in Figure 4.2.

```
struct {
    ContentType type;
    ProtocolVersion version;
    uint16 epoch;
    uint48 sequence_number;
    uint16 length;
    opaque fragment[DTLSPlaintext.length];
} DTLSPlaintext;
```

Figure 4.2: DTLS Record Structure

4.2 The Handshake

The Handshake protocol is used in the handshaking stage. It consists of several messages which are exchanged as shown in Figure 4.3a. The figure shows all handshake messages

and the order in which they are sent. The messages marked with an asterisk are situation dependent and will be sent for some of the existing cipher suites, but not all. A typical handshake message flow will resemble the message flow in Figure 4.3a starting from the second ClientHello message. The HelloRequest message is sent after initial session negotiation if the server side wants to renegotiate the parameters of the connection. The HelloVerifyRequest is sent by the server to make sure the client is legitimate, i.e. the ClientHello message was sent from an authentic IP address. The message contains a stateless cookie(one that the server can verify without remembering it), which the client has to include in its second ClientHello message.

The ClientHello message contains an array of random bytes, used later in the key generation phase, and a list of supported cipher suites and compression methods. It may also contain a session ID, indicating that the client wants to update the current session or reopen a previous session. If it is sent after a HelloVerifyRequest message was received, the ClientHello message will contain the cookie from that message. Last, but not least, the ClientHello message may contain a list of extensions, used to negotiate further session details.

The ServerHello message is sent upon receiving a ClientHello message. It contains an array of random bytes, used later in the key generation phase, a cipher suite and a compression method, selected from the lists in the received ClientHello message. If the ClientHello message contains extensions, which the server supports and wishes to use, the ServerHello message will contain a list of those as well.

Certificate messages are exchanged if the selected authentication scheme requires them. The client sends a Certificate message if it received a CertificateRequest message from the server. It also needs to send a CertificateVerify message, containing private key encrypted data, later to prove possession of the private key corresponding to the public key supplied with the client's certificate.

The ServerKeyExchange is sent when the key exchange scheme requires additional parameters. For example, this message is sent if the Diffie–Hellman key exchange method is used and the parameters used by the peers are not fixed.

The server sends a CertificateRequest if it wants the client to authenticate itself.

The ServerHelloDone message is sent to indicate that the server will not send more messages at this point.

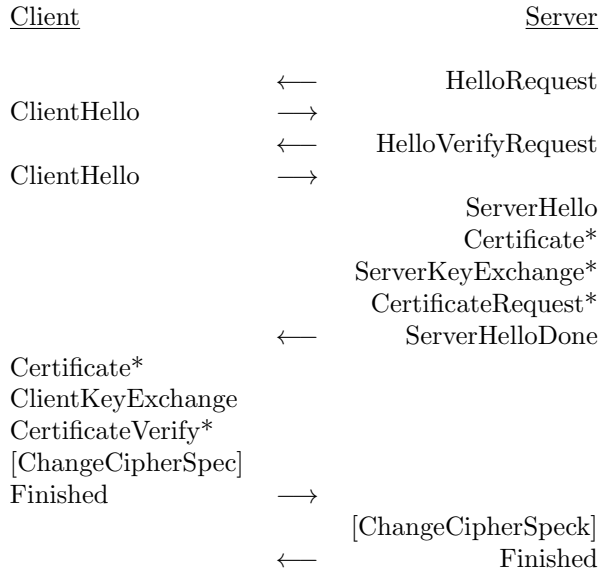
Depending on the selected key exchange scheme the ClientKeyExchange message contains a pre-master secret or data used to derive a pre-master secret(e.g. Diffie–Hellman parameters or PSK identity). The pre-master secret is used later in the key derivation phase.

The ChangeCipherSpec messages are not Handshake messages. However, they are used within the handshake to engage the negotiated algorithms and keys for use on the Finished messages.

The Finished message contains data, derived from the hash of the concatenation of all sent and received message thus far(excluding duplicates, the HelloVerifyRequest and all messages sent and received before it) in the order they were sent/received. It is encrypted and authenticated with the chosen keys and algorithms and serves as a test that the handshake process was successful.

As previously mentioned, DTLS was designed for use with unreliable transport channels. This is why DTLS, for the most part, does not require that all records are received or that they are received in the proper sequence. However, the protocol incorporates mechanisms, which enable retransmissions wherever needed. The handshake process is one such situation, because it requires reliable message exchange to succeed. DTLS handles message retransmission through the handshake by design, effectively incorporating reliable communication when establishing a connection and when renegotiating session parameters. The exact structure of a handshake message is shown in Figure 4.3b. This

structure enables message loss detection as well as the reassembly of fragmented handshake messages. The message_seq field is used to enumerate a peer's messages through the handshake and enables the other party in the communication to detect that a previous handshake message was lost. The length, fragment_offset and fragment_length fields are used to fit an incoming fragment within a buffer until the entire handshake message is reconstructed and ready to process.



(a) DTLS Handshake Message Flow

```
struct {
  HandshakeType msg_type;
  uint24 length;
  uint16 message_seq;
  uint24 fragment_offset;
  uint24 fragment_length;
  opaque body;
} Handshake;
```

(b) DTLS Handshake Message Structure

Figure 4.3: DTLS Handshake Message Flow and Structure

As a result of a successful handshake process, a connection sate is created for the newly negotiated session between the two communicating peers. The said connection state contains information, such as the peer's role in the given session, the algorithms used for encryption, MAC generation and key derivation, as well as the material used for key generation. A conceptual depiction of these parameters is presented in Figure 4.4.

```
struct {
  ConnectionEnd           entity;
  PRFAlgorithm            prf_algorithm;
  BulkCipherAlgorithm     bulk_cipher_algorithm;
  CipherType              cipher_type;
  uint8                   enc_key_length;
  uint8                   block_length;
  uint8                   fixed_iv_length;
  uint8                   record_iv_length;
  MACAlgorithm            mac_algorithm;
  uint8                   mac_length;
  uint8                   mac_key_length;
  CompressionMethod       compression_algorithm;
  opaque                  master_secret[48];
  opaque                  client_random[32];
  opaque                  server_random[32];
} SecurityParameters;
```

Figure 4.4: DTLS Session Security Patrameters

4.3 DTLS Algorithm Selection and Key Establishment

The ultimate goal of DTLS is that the two communicating parties agree on algorithms to use for message encryption and MAC calculation as well as to generate matching keys on both sides for these operations. In the ClientHello message the client provides a list of cipher suites and in its ServeHello message the server includes its preferred cipher suite from that list. This way the peers agree on mutually supported algorithms to be employed for message protection through the session. Encryption keys are established by exchanging 3 random values - a 32 byte long client random value, a 32 byte long server random value and a 48 byte long pre-master secret. The former two values are exchanged via the client and server hello messages and are not secret. The latter is protected via the key exchange scheme of choice. The three values are used to generate a 48 byte long master secret. This value, in turn, is used in combination with the client and the server random numbers to generate as much key material as the selected algorithms require. A pseudo random function(PRF) is used to generate the master secret and the keys. This function is defined in [63] as:

```
PRF(secret, label, seed) = P_hash(secret, label + seed)
```

P_hash, in turn is defined as:

```
P_hash(secret, seed) =  HMAC_hash(secret, A(1) + seed) +
                        HMAC_hash(secret, A(2) + seed) +
                        HMAC_hash(secret, A(3) + seed) + ...
```

where + stands for concatenation and A() is defined as:

```
A(0) = seed
A(i) = HMAC_hash(secret, A(i-1))
```

For TLS 1.2 and DTLS 1.2 HMAC uses SHA 256 as the underlying hash function.

The PRF should not be mistaken with the source of randomness used to generate the server and client random values and the pre-master secret. To generate the master secret PRF is called as follows:

```
master_secret = PRF(pre_master_secret, "master secret",
                    ClientHello.random + ServerHello.random);
```

where + stands for concatenation.
The PRF is later run using the 48 byte long master secret value as follows:

```
key_block = PRF(SecurityParameters.master_secret,
                "key expansion",
                SecurityParameters.server_random +
                SecurityParameters.client_random);
```

Once again + stands for concatenation.
When called, the PRF generates as many bytes as it is prompted to. For the key block the required number of bytes depends on the selected cipher and its mode(in DTLS stream ciphers are not used). The following values are extracted from the key block in the order they are listed: client MAC key, server MAC key, client encryption key, server encryption key, client IV and server IV. The MAC keys are not generated when the selected mode is one of the authenticated encryption modes(e.g. CCM). The write IV values are generated for modes which utilize partial nonces [see Section 3.2.1 in 25].

4.4 DTLS State Machine

DTLS uses the state machine presented in Figure 4.5 to guide message transmission, retransmission, and reception. The state machine dictates that messages are sent in flights, where a flight is a sequence of messages sent before a response should be expected. Once a flight is generated it is transmitted and a response is awaited. If a response does not arrive within a given time frame, the entire last flight is retransmitted. A peer also retransmits its last flight if it did not receive all expected messages from the other communicating party's current flight. Thus, if a peer receives a retransmission it retransmits its last flight to allow the opposing peer to receive any missed messages. A peer's last flight concludes with the transmission of a Finished message.Once the last flight, for the server side, is sent or the peer's last flight is received, for the client side, the done state is entered. Sending and receiving HelloRequest or ClientHello messages restarts the handshake process.

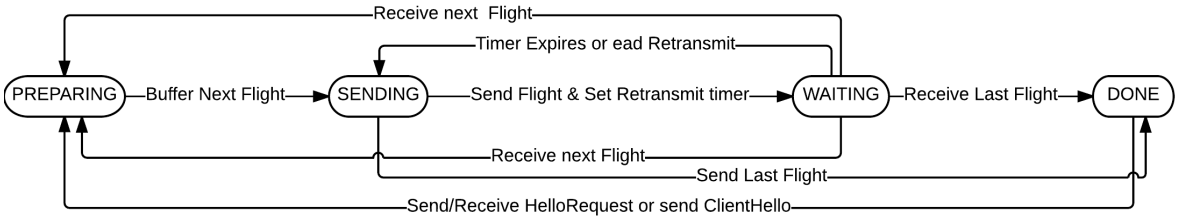


Figure 4.5: DTLS Timeout Retransmit State Machine

4.5 Components of DTLS Implementations

A DTLS implementation will typically consist of several components, namely:

Source of Randomness
DTLS's key generation process and the strength of the resulting keys depends heavily on the use of random numbers. If the random numbers are predictable, the generated keys will have weak security properties. This is why DTLS implementations should provide a way to generate unpredictable random bytes.

Cryptographic Algorithms
This component contains the symmetric and asymmetric(public key) ciphers used for encryption, peer authentication and key exchange. This component should also include functions to enable the use of these ciphers in schemes relevant to the version of DTLS, being employed, and the supported cipher suites.

DTLS State Machine
This component is responsible for handling the handshake process. It addresses state transitions, transmissions and retransmissions.

Message Related Functions
This component refers to the functionality responsible for message and record generation and processing, as well as sequence number and epoch maintenance. Message fragmentation, compression and encryption are addressed here.

Key Exchange Functionality
Depending on the supported key exchange procedures additional functionality may be needed. For example, if RSA is used as authentication and key exchange method

the RSA key will be included in a certificate. Additional functions will be needed to process and validate this certificate.

4.6 DTLS for CoAP and the IoT

In their publication "The Constrained Application Protocol(CoAP)" [58], the IETF specifies the use of DTLS for securing CoAP. Furthermore, the document addresses the three key exchange strategies - pre-shared keys, raw public key and certificates. The RFC elects TLS_PSK_WITH_AES_128_CCM_8 and TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8 as the mandatory cipher suites to implement for pre-shared keys, and raw public key and certificates, respectively. These cipher suites utilize compact and efficient record layer protection using AES 128 in CCM mode and only 8 byte long MAC. Selecting CCM mode reduces the size of the required key material per connection as only 1 key is used for both encryption and MAC generation.

A further, much more detailed analysis of the 2 cipher suites is present in IETF's draft "TLS/DTLS Profiles for the Internet of Things" [66]. It explores the handshake message exchange and related operations for both. Furthermore, the draft elaborates with an exploration of the additional requirements and schemes involved when either cipher suite is used. Last, but not least, it includes recommendations concerning specific aspects of the use of the cipher suites as well as suggestions for use of hello extensions in general and for each cipher suite for use in constrained environments.

4.6.1 Pre-Shared Keys

When pre-shared keys are used the DTLS handshake message flow proceeds as shown in Figure 4.6. Messages marked with asterisk are situation dependent. The client and server hello are used to agree on the use of the TLS_PSK_WITH_AES_128_CCM_8 cipher suite. If the server wants to use a particular pre-shared secret it sends an identity hint in its ServerKeyExchange message. If the client receives a ServerKeyExchange message, it checks if it has the associated pre-shared secret. If it does, it includes the psk identity for this secret in its ClientKeyExchange message. If no ServerKeyExchange message is received, the client selects a pre-shared secret and sends its identity to the server in a ClientKeyExchange message. The handshake concludes with both sides sending a change cipher speck and a finished message.

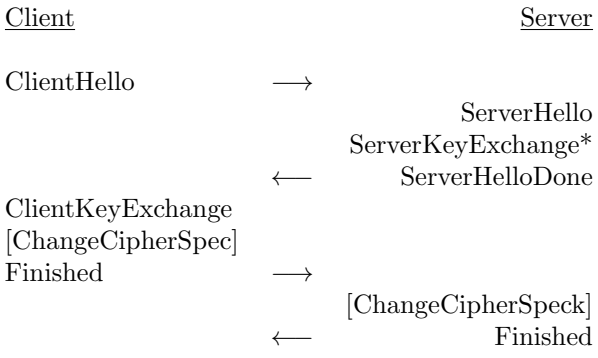


Figure 4.6: DTLS PSK Handshake Message Flow

There are two important implications of using pre-shared keys for the handshake process. First, the handshake consists of fewer messages. No certificate messages are exchanged, which also means significantly less data is transmitted as certificates can be quite lengthy. Second, no signing, verification and public key encryption operations are required, which also eases the handshake from a computational point of view.

The pre-shared secret is used to form the pre-master secret, which is, in turn, used to generate the master secret and eventually the session encryption keys. The pre-master secret is comprised of the PSK size - N, encoded in 2 bytes, followed by the N zero bytes, followed by N,encoded in 2 bytes, followed by the PSK itself. The pre-master secret construct is shown in Figure 4.7.

2 bytes	N bytes	2 bytes	N bytes
N	0....0	N	PSK

,where N = PSK byte size

Figure 4.7: PSK Pre-Master Secret

Using pre-shared keys requires that communicating peers are pre-configured with secrets for each desired connection as well as with corresponding identities for these secrets. Schemes have to be used to configure devices with keys as well as to update those keys over time. Anther potential drawback of using plane pre-shaded keys is that there is no perfect forward secrecy. Perfect forward secrecy is a property of key exchange schemes, which ensures that even if the key exchange is compromised for a given session, data exchanged in previous sessions remains protected. The Ephemeral Diffie–Hellman key exchange method, for example, provides perfect forward secrecy, because it does not use fixed parameters for the key exchange procedure.

4.6.2 Raw Public Keys

When raw public keys are used the DTLS handshake message flow proceeds as shown in Figure 4.8.

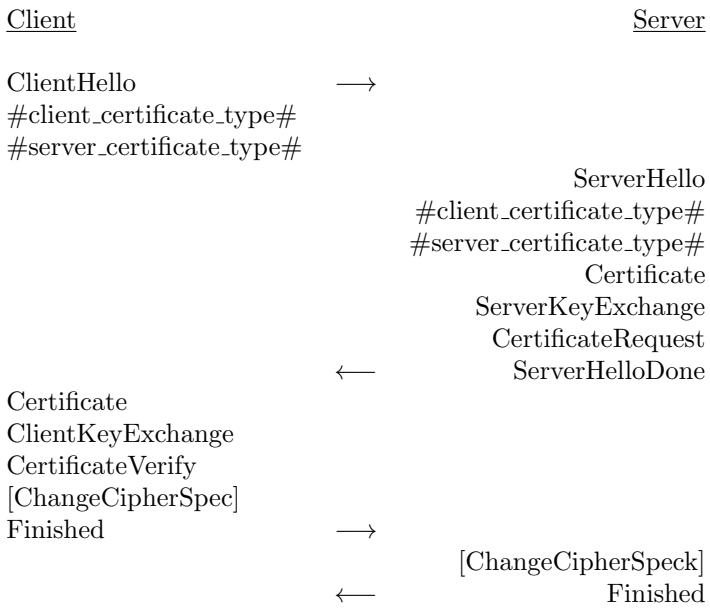


Figure 4.8: DTLS Handshake Message Flow

The certificate message is used to transmit the sender's public key. The server and client hello messages should include server and client certificate type extensions [71] to indicate their support for raw public keys. If the client and the server agree to use raw public keys for authentication, they use an out of band key verification approach.

Raw public keys are the middle ground between pre-shared keys and certificates. The overhead of per-device certificate is avoided, but handshakes may still involve performing expensive public key operations. IETF has selected TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8 as the mandatory to support cipher suite in constrained environments when raw public keys are used. This cipher suite provides

perfect forward secrecy. However, maintaining the raw public keys for each device still requires some care. Furthermore, the related handshake involves considerably more messages as well as computationally expensive operations for the key exchange procedure.

4.6.3 Certificates

When certificates are used the handshake process is similar to the one for raw public keys. However, certificates are exchanged and verified. The use of certificates requires devices to have a public-private key pair as well as their own valid certificate. Additionally, devices should be configured with the public keys of each trusted certificate agency as well as certificate revocation lists. The support of related cipher suites will result in longer, more computationally expensive handshakes. Furthermore, the exchanged messages will be rather big compared to using pre-shared keys or raw public keys, because certificates tend to be hundreds of bytes long and more often than not multiple certificates are sent in a chain. For constrained nodes processing incoming certificates will require large buffers and certificate verification functionality. TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8 is defined as the must implement cipher suite for implementations that wish to use certificates. The benefit here is that this cipher suite provides perfect forward secrecy. A last thing to consider is that even if certificate structure and size can be controlled, maintaining certificates on all related devices is just as problematic as with the previous approaches.

4.7 DTLS and Multicast

CoAP was designed as the HTTP equivalent for constrained environments. It provides similar functionality, but is more compact. The protocol was designed to support functionality that enhances communication efficiency, such as multicast. DTLS has been elected the must implement security protocol for CoAP implementations by the IETF. It has been profiled for use in constrained environments and schemes have been recommended to reduce the protocol's implementation size and performance impact in both transmission and computation overhead. Unfortunately, the standard DTLS does not support multicast natively. Other ways to secure multicast transmissions exist [44] [65] [62] [45] [36], but those are not suitable for use in constrained environments, as noted in [see the Introduction in 32]. Furthermore, due to the nature of constrained environments, CoAP's target area, it is desirable that a single security protocol is used for all network communications, so that implementation size stays minimal. Therefore, there is an ongoing effort in the development of schemes to adapt DTLS so that it can be used to protect multicast transmissions. A special working group was formed in the IETF to carry this task out. The current state of their work is presented in the IETF draft "DTLS-based Multicast Security in Constrained Environments" [32]. The draft has expired, so it should not be used as a standard. However it provides multiple useful concepts for the further development in the area. Firstly, several terms, roles and building blocks are defined. Those are as follows:

Group Controller: An entity, whose responsibility is to create multicast groups, generating and distributing as well as updating security associations among authorized group members.

Sender: A device/entity that sends data to a multicast group. Depending on the multicast group, there can be one to 50 senders in CoAP based applications.

Listener: A device/entity, which listens to a given multicast IP address and receives incoming messages.

Security Association(SA): A policy and encryption/decryption keys providing and guiding the security services for network traffic protection in networks following this policy. SAs consist of three components, namely:

- Selectors: identifiers for the senders and the listeners in a multicast group.

- Policy: the cipher suites and the keys' lifetimes for the given multicast group.

- Keying Material: used for key generation.

Group Security Association(GSA): A collection of SAs, which collectively define how to secure communications in the multicast group.

Keying Material: The same concept as for the SA, only expanded to all involved SAs for the multicast group.

The document outlines a generalized solution that fulfills the following requirements:

- It addresses both one-to-many and many-to-many communication topologies.
- It should support the typical group sizes defined for CoAP. The total expected members in a group(both senders and listeners) range from 2 to 100, where 1 to 50 of the members could be senders. Larger groups should be divided into sub-groups.
- It should provide multicast data confidentiality.
- It should provide multicast data replay protection.
- It should provide multicast data authentication and integrity to ensure a message originated from a member of the group and that the message was not modified along the way.

The basic guidance the document provides for achieving the requirements involves several aspects.

First, there is the set-up. It consists of the creation of multicast groups and associated GSAs and their distribution to the group members. This process is carried out by the controller and may involve devices discovering this controller.

Once the setup is done, senders can start using the cipher suites and the key materials, designated in the GSA for the multicast group, to encrypt and authenticate messages. Messages are then sent to and received from the designated multicast IP address. Listeners use the GSA for the group connection, associated with the multicast IP address, to decrypt and authenticate incoming messages.

Provided that the setup is complete, the following guidelines should be followed:

- The device's role for the DTLS connection should be set to 'server' sor senders and 'client' for listeners.
- GSAs should set the client and server random numbers to identical values on all devices so that the same keys are derived on each device. Another, more efficient, alternative is to directly include the encryption keys, the MAC keys and the IVs in the GSA.
- When a group has multiple senders the controller has to assign a unique 1 byte long id to each sender. This id will be used as the first octet in the sender's record sequence number. This removes the need for synchronization between record sequence numbers among senders(each sender has their own sequence number). Additionally, when CCM modes are used, the uniqueness of each sender's record sequence number ensures that no nonce reuse will occur in records sent from different senders. This is

important, because senders use a common group key to encrypt their messages. It is essential, when CCM mode is used, that no message is encrypted with the same nonce and key more than once.

- When senders start sending application data, they have to use their sender id as the first byte in the sequence number. The remaining 5 bytes of the sequence number are set to 0. Senders manage their own epoch and sequence numbers. Upon sending a record, the sequence number for the corresponding connection is incremented.

- Listeners need to store multiple DTLS connection states for the given multicast connection - one for each sender. All these connection states share the same group keying material. However, the epoch and last received sequence number will be different for different senders.

- Should a listener have to respond to a sender's message, this response should be secured. This means that responses should be sent over unicast DTLS connections, which would require that the sender and the listener have a separate session for use for those responses.

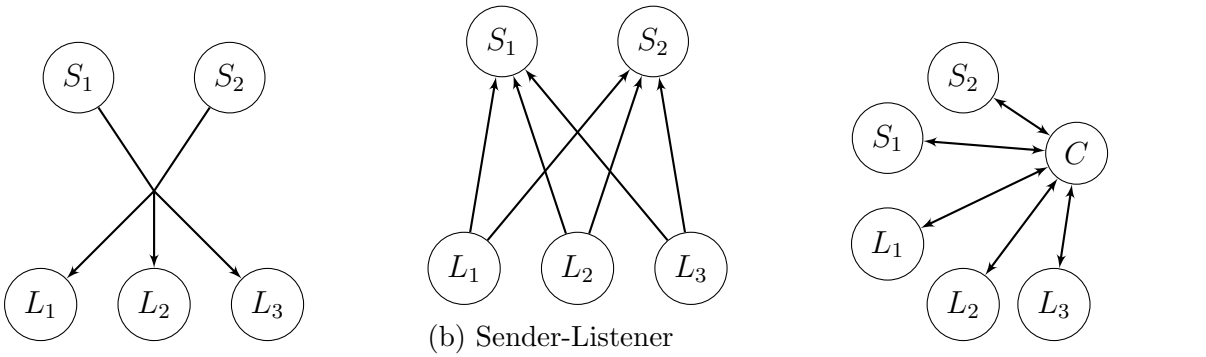
- If the client decides to use a proxy in a multicast scenario, a two-step approach should be taken. That is, the client sends a unicast DTLS request to the proxy. The proxy will decrypt and authenticate the message and create a new multicast message with the same content as the original unicast message. The proxy will first secure the message using DTLS based multicast and transmit it to the corresponding group.

The described approach results in the topology depicted in Figure 4.9. As the figure suggests, the approach, proposed by the IETF requires 3 sets of connections, namely - a multicast connection, depicted in Figure 4.9a, unicast listener-to-sender connections, depicted in Figure 4.9b, and controller-to-group members connections, depicted in Figure 4.9c.

The multicast connection is used to send requests from senders to listeners. It requires that all senders and listeners posses the same multicast keys for encryption and MAC generation. It also requires that senders have unique IDs and that each listener maintains a list of sender ID-to-sender sequence number for all senders.

The listener-to-sender connections are standard unicast DTLS connections and are used to deliver responses from listeners to senders. They can also be used for non-multicast related communications between listener and sender devices.

The controller-to-group member connections are used to distribute multicast related information from the controller to each group member. More precisely, they are used to change the multicast parameters upon adding and removing group members to the multicast group.



(a) Multicast Connection

(b) Sender-Listener Connections

(c) Controller Connections

Figure 4.9: IETF DTLS Multicast Topology

5. Implementation

As stated in Chapter 2, the targeted hardware for this project has significant limitations, especially in RAM and flash memory. The design of both the unicast DTLS implementation and the proposed multicast scheme were guided by these limitations. While limited processing power has a direct impact on device responsiveness and overall speed and reliability of network communications, given enough time and a good communication scheme data exchange is achievable. Memory usage, on the other hand, has to be well managed to ensure operations can be performed. Unlike general purpose PCs, embedded devices do not have the luxury to dynamically allocate a required amount of memory, due to potential memory leaks and fragmentation. If a running process requires more memory than the system has, the process cannot run. Therefore, wherever necessary, code footprint and RAM usage are prioritized over performance. Nevertheless, both criteria have been addressed by the implementation. An additional factor for the implementation decisions and future improvement considerations is the size and number of the transmitted messages. Application data messages will have a constant header and encryption overhead. Handshakes, on the other hand, require performing DTLS specific message exchange and related operations. Depending on how frequently handshakes are performed, the associated overhead may have varying impact on the overall communication and energy consumption. Devices in constrained environments are mostly battery powered, which means they tend to often enter a sleeping state to preserve power. Unfortunately, when this happens session information may be lost. This implies that DTLS handshakes will be performed every time a device wakes up. Therefore, the associated overhead is going to have a noticeable impact on communications and power consumption, and should be as negligible as possible.

On a separate note, this implementation will run on all devices within the home area network and possibly on devices outside it. Some devices, like the gateway, have more processing power and memory than others. They may be able to overcome some of the assumed limitations. However, the implementation assumes the weakest link's perspective, which is a standalone radio module(for example on a sensor).

5.1 Selected Cipher Suite

Both cipher suites, suggested by IETF for use in constrained environments(see section 4.6) use AES in CCM mode for data encryption and authentication, and define 8 byte MAC size. The selected block cipher, AES, as well as the key size, 128 bits, are sensible choices fora good trade-off between performance key size and security. The CCM mode of operation uses a single key for encryption and MAC generation, which allows for memory savings and reduced encryption-related message size overhead. The difference between the two cipher suites is in the key exchange and peer authentication mechanisms. TLS_PSK_WITH_AES_128_CCM_8 defines the use of pre-shared secrets for key exchange and indirect peer authentication, while TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8 uses either a raw public key or a certificate for these authentication and elliptic curve ephemeral Diffie–Hellman operations for key exchange. Both cipher suites have advantages and disadvantages.

5.1.1 TLS_PSK_WITH_AES_128_CCM_8

The use of this cipher suite requires that two peers be pre-configured with a matching set of pre-shared secrets designated for use in connections between these peers. Upon session negotiation one of these secrets is selected and used to form a pre-master secret for the key generation procedure. The resulting encryption keys are verified upon exchange of Finished messages(see section 4.2). The benefit of using pre-shared secrets is that handshakes require fewer messages and no public key operations are performed, as opposed to using raw public keys or certificates. Figure 4.6 in Chapter 4 depicts the related message flow. An additional benefit of using pre-shared secrets is that the pre-shared secret is never directly used for encryption. As specified in [49], pre-shared secrets can be up to 64 bytes, or 512 bits, in size. Using the maximum key size and assuming that no information about the pre-shared secret is leaked we can expect a 2^{512} level of security. This means it will take 2^{512} tries to brute-force guess the pre-shared secret. However, it might be impractical to use 64 byte long secrets, especially if the device supports multiple connections. A much more practical pre-shared secret size range is 16-32 bytes, which gives 2^{128} to 2^{256} level of security. This security level rivals that of RSA and DSA with 3072 bit long keys or higher [13].

A downside of using pre-shared keys is that they do not provide perfect forward secrecy. Furthermore, the amount of memory required to store pre-shared keys is proportional to the number of connections a device maintains. Last, but not least, if a pre-shared secret is compromised it has to be replaced on both devices using it. This also implies that if a device is hijacked or compromised, all devices that used to communicate with it should reject the corresponding connection to that device.

5.1.2 TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8

This cipher suite requires the support of elliptic curve cryptography. It can be used with both Raw Public Key and Certificate based peer authentication. In either case the use of elliptic curve ephemeral Diffie–Hellman key exchange provides perfect forward secrecy. Furthermore, it allows the use of 256 to 521 bit numbers to achieve security levels of 2^{128} to 2^{256} for the key exchange procedure, respectively [14].

Raw Public Keys

When used with Raw Public Key this cipher suite requires that target devices possess a public-private key pair. The handshake behavior is as descrybed in [71] and the related message exchange is depicted in Figure 4.8 in Chapter 4. The client and server hello messages contain the client_certificate_type and server_certificate_type extensions with RawPublicKey as the certificate type. The server, and optionally the client, send DER encoded SubjectPublicKeyInfo structures, containing their public key in the corresponding certificate messages. These structures are verified using an out of band method.

The benefit of Raw Public Keys is that a single key pair can be used for all connections the key pair owner participates in. Furthermore, since the authentication is performed in an out-of-band fashion(assuming that means the SubjectPrivateKeyInfo is verified on by a dedicated server) devices do not need to support key verification functionality. If a Raw Public Key is compromised only the verification server needs to be aware of this. Last, but not least, the overhead of using certificates is avoided.

The drawback here is the use of out-of-bound key verification. There is no clear definition of what this out-of-bound method may be, but it is fair to assume it will involve a connection to and communication with a dedicated server. If that is the case, then the handshake depends on extra message exchange over a different connection.

Certificates

The use of certificates for authentication requires that devices posses certificates, a public-private key pair and a certificate agency's public key. It also requires the support for ASN.1 [38] and DER [39] functionality, unless certificates are verified in an out-of-band fashion.

The benefit of using certificates is that the same certificate can be used for all connections the certificate owner part takes in. Two devices need not have any previous knowledge of each other's existence. They only need to know the certificate agency, responsible for distributing certificates.

The drawback of using certificates is that they are quite spacious, especially if certificate chains are used. Their size can be minimized if the device manufacturer creates the certificates as well, but their overall size will still hover around 1 kilobyte. Additionally, processing certificates requires support of related functionality, unless the certificate is verified in an out-of-band fashion(which has the same implications as for Raw Public Keys). Certificate exchange imposes noticeable communication overhead in the number of exchanged messages, the overall amount of exchanged information and the computation cost of the handshake. If a certificate is compromised all devices need to be notified and remember to reject it. This impact can be mitigated if an out-of-band verification is used.

The use of TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8, regardless of the authentication method used, requires support of elliptic curve cryptography, which impacts the code footprint. Additionally, the handshake complexity, cost and size, in number of messages, increases.

Ultimately, due to memory limitations, performance concerns and anticipated number of connections and handshake frequency, the use of TLS_PSK_WITH_AES_128_CCM_8 was deemed most appropriate for the targeted hardware and environment. This approach does not involve the use of public key operations and requires the shortest, most efficient handshake procedure. Furthermore, elliptic curve functionality is not supported in the target system and is outside the scope of this project. The required memory size for the pre-shared secrets grows proportionally to the number of connections a device maintains. In dynamic environments this may eventually result in greater flash requirements than what is needed for certificates. However, for more static applications this should not be a real problem(that still depends on the number of connections and the number and size of corresponding pre-shared secrets). The use of TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8 may be a better choice when scalability is a priority.

5.2 Memory Usage Considerations

The DTLS 1.2 standard has a clear description of the information required for a connection to function. The session keys, the epoch number and the incoming and outgoing sequence numbers are essential for secure data exchange. The required data to securely conduct a handshake is also well defined. The per session and per handshake data usage can be minimized, especially in cases where a single cipher suite is used(e.g. there is no need to remember what algorithms are used). However, I believe that much more noticeable memory savings can be achieved with a proper strategy for using the implementation. Here I present a few exemplary usage tricks which can reduce the per session and per handshake memory usage.

In order to showcase my ideas of how to preserve memory I assume the existence of two streams in the communication stack - a read and a write stream. It is also assumed that the streams' size is between 1500 and 1000 bytes (see Figure 5.1). Both assumptions

are fair in a network communication protocol stack, where UDP is the transport protocol of choice. The read stream contains incoming messages, while the write stream is used to store currently generated messages. These streams are shared between connections, which means only one connection can send and/or receive messages at a given instance in time.

(a) Write Stream (b) Read Stream

Figure 5.1: Assumed Streams

To reduce the amount of space needed per connection, handshake specific fields can shared among connections. This implies that only one handshake takes place at a time.

The Finished message is generated using the hash of the concatenation of all previously sent and received messages. In an attempt to preserve space, the implementation does not store the concatenated version of the handshake messages, but rather maintains a running hash, which is updated each time a handshake message is received or sent(unless the message is a retransmission). This effectively allows to reduce the required space for the hash to the size of the hash function digest and the hash function block size. The implementation uses SHA 256 for hashing, which has a digest size of 32 bytes and a block size of 64 bytes for a total of 96 bytes. The implementation uses Oliver Gay's open source implementation of HMAC and SHA 256 [15].

To further reduce the memory overhead of the handshake logic, spacious fields, such as the client and server random numbers and the handshake message hash can be kept at the end of the write stream(see Figure 5.2), rather than as dedicated fields. If a cookie exchange takes place, the client implementation can store the cookie at the end of the write stream as well(see Figure 5.2b). A single byte, preceding all fixed size fields at the end of the write stream, indicates the presence of a cookie. When a client message is generated the implementation checks this byte to see if a cookie was sent by the server. This use of the write stream is problematic, because other connections may receive and send messages whilst a handshake is taking place. If this happens the handshake fields at the end of the write stream may accidentally be overwritten. The total size of the write stream should be reduced through a handshake to prevent this from happening. Upon generating a message each connection has to check the available space in the write stream.

Available Space	Reserved Space			
	1 byte	≈ 96 bytes	32 bytes	32 bytes
.....	0	hash	client random	server random

(a) No Cookie

Available Space	Reserved Space				
	≤ 255 bytes	1 byte	≈ 96 bytes	32 bytes	32 bytes
.....	cookie	cookie size	hash	client random	server random

(b) With Cookie

Figure 5.2: Write Stream Usage

Bit fields are used in places, where the maximum value of a variable requires less bits than the type used to save it. For example a session needs to know the role the device assumes for a given connection [Secttion 6.1 in 63] - a client or a server. A single bit

can be used to save this information rather than a whole byte. Internal representation of fields does not affect their encoding in messages.

As previously mentioned, handshakes will predominantly happen due to devices falling asleep and waking up again, which implies complete handshakes resulting in new sessions. This means re-handshakes, updating just the client and server random numbers are unlikely. To address this, the implementation does not re-handshake, but negotiates a new session each time. This means the epoch value does not exceed 1. Once again, a single bit suffices to encode the current epoch value. A further implication of this design decision is that there is no need to remember the generated master secret or the server and client random numbers in sessions as they will change at each handshake. However, these values will be needed for the duration of the handshake.

TLS and DTLS sessions are given IDs by the server in the communication. This is important when given client and server maintain multiple sessions on a single connection and the server caches them. The target system maintains a single connection between devices. Furthermore, due to memory constrains and the sleep-awake character of the devices, session caching is infeasible. Therefore, this implementation does not support session caching, so there is no need to remember session IDs. Sessions are bound to device MACc addresses.

5.3 Random Number Generator

For the purposes of generating the client and server random values, this implementation uses the source of randomness, supported by the targeted system. More precisely, the system uses radio noise readings as a source of entropy.

5.4 Path Maximum Transmission Unit Discovery

This implementation's approach towards PMTU discovery assumes these three possible communication scenarios:

- The client and the server are both in the same home area network and run this implementation of DTLS 1.2.
- The server is a device in a home area network and the client communicates through the gateway via the Internet. The server runs this implementation of DTLS 1.2, while the client may run this or another implementation of DTLS 1.2.
- The client is a device in a home area network and the server communicates through the gateway via the Internet. The client runs this implementation of DTLS 1.2, while the server may run this or another implementation of DTLS 1.2.

The implementation does not perform true PMTU discovery. Instead, the Maximum-FragmentLength extension [see Section 4 in 68] is used. It allows clients to declare a preferred maximum plain text size for the record. The extension defines 4 values, namely 512, 1024, 2048 and 4096(encoded as 1, 2, 3 and 4 in a single byte), which correspond to the maximum allowed byte size of the unencrypted fragment in a record.

When the client and the server are in the same home area network, they are both devices running this DTLS 1.2 implementation. This means they are the communication bottlenecks and know their own limitations. In such cases the client sends a ClientHello message with the MaximumFragmentLength extension, containing 512 or 1024(higher values are avoided, due to UDP's limitation of 1500 bytes). The server also includes the MaximumFragmentLength extension, containing the value sent by the client, to agree with this limitation. Communications proceed with the imposed limitation.

When the client is not a part of the home area network of the server, the client's implementation of DTLS 1.2 is unknown. If it sends an acceptable value in a MaximumFragmentLength extension in its ClientHello message, the server responds with the same value in the ServerHello message and communications proceed as in the previous case. If the client requests an unacceptable maximum plain text size value, the server rejects the connection. If the client does not include the MaximumFragmentLength extension in the ClientHello message, the server assumes the client has performed a PMTU discovery and uses the smallest maximum plain text size when generating records.

When the server is not a part of the home area network of the client, the server's implementation of DTLS 1.2 is unknown. The client sends a ClientHello message, containing a MaximumFragmentLength extension with a minimum value. If the server responds with an appropriate extension in the ServerHello message, communications proceed as in the first case, otherwise the connection is rejected.

Rejecting connections on the basis of unpredictable incoming record sizes is justified by the limited statically sized buffer of the targeted radio module. Optionally, connections with unbound record sizes could be accepted and messages that overflow the buffer rejected.

5.5 Message Sizez

Since pre-shared secrets are uses, the handshake may, at most, involve the exchange of 8 distinct messages plus the change cipher speck message. The hello messages in question, in the sequence they would be sent, are: HelloRequest, ClientHello, HelloVerifyRequest, ServerHello, ServerKeyExchange, ServerHelloDone, ClientKeyExchange, and Finished. Here I analyze the maximum size of the hello messages, generated by this implementation.

5.5.1 Record Header

All messages are encapsulated in a record structure, which contains a header with statically sized fields. The combined size of these field is 13 bytes(see Figure 5.3).

Substructure	Record Header						Record Body
Field	Content Type	Protocol Version		Epoch	Sequence Number	Length	Fragment
Size	1 byte	1 byte	1 byte	2 bytes	6 bytes	2 bytes	The fragment's length

Figure 5.3: Record Structure

5.5.2 Handshake Message Header

Different handshake messages have their own structure. However, they are all encapsulated in a generic handshake message(see Section 4.2) which contains a statically sized set of header fields. The combined total size of these fields is 12 bytes. This is important, because each handshake message will contain this header overhead. Figure 5.4 depicts the general structure of a handshake message, where the body is one of the previously listed handshake messages. Each handshake message will include the record and the handshake header for a total of 25 additional bytes.

5.5.3 HelloRequest

The hello request message is empty. It is used by the server only to prompt the client for a re-handshake. This message will be the same size as the handshake message header.

Substructure	Handshake Message Header					Handshake Message Body
Field	Type	Length	Sequence Number	Fragment Offset	Fragment Length	Body(fragment)
Size	1 byte	3 bytes	2 bytes	3 bytes	3 bytes	The body's length

Figure 5.4: Handshake Message Structure

5.5.4 ClientHello

The general client hello message structure is depicted in Figure 5.5. As the figure suggests, this message contains at least 42 bytes. The message is exactly 42 bytes if the session id is empty(the session id size is 0), the cookie is empty(the cookie size is 0), only one cipher suite and 1 compression method are included and there are no extensions. This implementation does not support session caching, so the session id will always be empty. The cookie, if present, will be determined by the server, which does not necessarily run the same implementation. Therefore, in worst case scenario the cookie will be 255 bytes(as defined in [28]). At its current state the implementation supports only the cipher suite TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8 and does not support compression. Therefore, the encoding of the cipher suite list is 2 bytes(excluding the encoding of the cipher suite list's size), while the encoding of the compression method list is 1 byte(excluding the encoding of the compression method list's size). The implementation only supports the MaxFragmentLength extension. The general structure of extensions is presented in Figure 5.6. The data field for the MaxFragmentLength extension is 1 byte. The entire encoding of the extension is 5 bytes(excluding the 2 bytes for the total size of of the list of extensions). Summing it all up, the client hello message tops up at 304 bytes.

Field	Protocol Version		Random		Session ID		Cookie		Cipher Suites		Compression Methods		Extensions (optional)	
Encoding	major	minor	time	random bytes	byte size	id	byte size	cookie	byte size	cs list	byte size	cm list	byte size	ext. list
Size	1 byte	1 byte	4 bytes	28 bytes	1 byte	≤ 32 bytes	1 byte	≤ 255 bytes	2 bytes	≥ 2 bytes	1 byte	≥ 1 bytes	2 bytes	varies

Figure 5.5: Client Hello Message Structure

Field	Extension Type	Extension Data	
Encoding	type	data size	data
Size	2 bytes	2 bytes	varies

Figure 5.6: Handshake Extension Structures

5.5.5 ServerHello

The structure of this message is similar to that of the client hello message. As you can see in Figure 5.7, it has all the fields of the client hello message except for the cookie. Also it contains a single cipher suite and compression method values, rather than a list. This message contains at least 38 bytes - when the session id is empty and there are no extensions. As explained in the previous section, the session id will always be empty and there is support only for the MaximumFragmentLength extension. Its size was previously stated. The server hello message will be at most 45 bytes.

5.5.6 ServerKeyExchange

Figure 5.8a depicts the structure of this message when pre-shared secrets are used [49]. The total size of this message depends on the selected format for the pre-shared secret

Field	Protocol Version		Random		SessionID		Cipher Suite	Compression Method	Extensions	
Encoding	major	minor	time	random bytes	byte size	id	cs	cm	byte size	ext. list
Size	1 byte	1 byte	4 bytes	28 bytes	1 byte	≤ 32	2 bytes	1 byte	2 bytes	varies

Figure 5.7: Server Hello MessageStructure

identity hint. While [49] does not provide clear definition for the format and size of the identity hint, it is unlikely that it will exceed the size of the identity itself, which is limited to 128 bytes. In any case, the size can be controlled by the person responsible for configuring the system. For the purposes of this report I assume the limit for the identity hint is 128 bytes, which gives us a maximum of 130 bytes for this message.

5.5.7 ClientKeyExchange

Figure 5.8b depicts the structure of this message when pre-shared secrets are used [49]. [49] defines the limit of the pre-shared secret identity to be 128 bytes, which gives us a maximum of 130 bytes for this message.

Field	PSk Identity Hint	
Encoding	Identity Hint Size	Identity Hint
Size	2 bytes	varies

(a) Server Key Exchange Message Structure

Field	PSk Identity	
Encoding	Identity Size	Identity
Size	2 bytes	varies

(b) Client Key Exchange Message Structure

Figure 5.8: Key Exchange Messages

5.5.8 Finished

This message contains a fixed size value, produced by the pseudo random function from the master secret, a finished label and the hash of all handshake messages preceding this one. The produced value is referred to as verify data and for DTLS 1.2 its size may be defined by the cipher suite, but its default is 12 bytes if the cipher suite does not specify the size. TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8 does not specify a size for the verify data, which means the total size of this message is 12 bytes.

5.5.9 ChangeCipherSpeck

Change cipher speck messages contain the value 1 encoded in a single byte.

5.5.10 Application Data

The size of the application data will depend on the amount of application data, being sent and the selected maximum plain text size(see Section 5.6).

5.5.11 Alert Messages

In DTLS sending alert messages is not mandatory and is discouraged when implementations are coupled with UDP [28]. This implementation follows the recommendation and does not send alert messages.

5.5.12 Incoming Messages

Incoming messages will not necessarily be produced by this implementation of DTLS 1.2. It is, therefore, possible that they will differ in size from the equivalent messages for this implementation. This is a potential problem for messages with DTLS related contents. The application data messages should be handled by the application, residing on top of DTLS. Alert and change cipher speck messages have small static sizes and do not pose a threat. The potentially problematic messages are the handshake messages. More precisely, the client hello message. This implementation supports a single cipher suite, no compression methods and only 1 extension, which allows for maximum size estimation of client hello messages generated by it. Other implementations may support a wide range of cipher suites(see [16]), compression methods(see [17]) as well as extensions(see [18]), which can result in rather big client hello messages. To preserve memory, this implementation does not use a separate buffer for message reconstruction. A big client hello message could be received one fragment at a time, which would make it impossible to process it as a whole. Clients, wishing to communicate with this implementation's servers, should avoid including unnecessary cipher suites and extensions in their client hello messages. The server key exchange message also has the potential to grow out of hand if unreasonably sized identity hints are used. This is not expected, because pre-shared secrets and associated identities and identity hints are controlled by the person, who sets the system up. It is unexpected that they will choose big identities and identity hints as those will also affect flash memory availability.

5.6 Message Fragmentation

The decisions for message fragmentation are guided by the analysis of the various message's sizes and the selected strategy for establishing maximum plain text size. More precisely, none of the handshake messages needs to be fragmented, because their sizes are within the limits of what is considered acceptable in the target environment. The only message type, which may require fragmentation is the application data message. If it exceeds the agreed upon handshake maximum plaitext size, the message is fragmented in as many records as required to send the entire application data. Message reconstruction is left to the application layer, because DTLS cannot interpret the application data.

5.7 Efficiency

The selected cipher suite does not require expensive public key operations for the handshake. Furthermore, the target device, the radio module, has hardware support for AES 128. The implementation uses it for message encryption and authentication. The system supports both CBC and CRT modes of operation. To enable CCM mode, the cipher is used 2 consecutive times, first in CBC mode and then in CTR mode. Apart from the speed and energy efficiency benefit of using hardware supported cipher, the system is free to perform other operations whilst the cipher is encrypting a message.

6. Proposed Multicast Approach

6.1 Flaws of Current Approaches

The DTLS based multicast security approach, presented in [32] includes a great analysis of the problematic aspects of introducing multicast support in DTLS. While the proposed topology(see Figure 4.9) addresses all aspects, recognized in the document, it is not scalable. There are multiple reasons for that. To better describe the individual issues I assume a setup with a single controller, m senders and n listeners. I also assume the worst case scenario, where each sender addresses all listeners.

Numbe of connections: In the assumed setup, if the IETF proposed topology is used, each sender will have individual connections with all users. This adds to n connections per sender, m connections per listener and $m * n$ total connections for all senders and listeners. If we also include the $m + n$ connections the controller has to maintain, we end up with $m * n + m + n$. As m and n grow, so does the complexity. If the setup were to reach the defined maximum of 50 senders and 50 listeners(due to the group limitation of 100 members) there will be 2600 connections.

Memory impact: The worst case scenario from an individual node's point of view is that it has to maintain 99 connections(if there is 1 sender and 99 listeners). Each connection has a distinct encryption key for the corresponding listener. If the sender only communicates to the listeners via the multicast connection, it can reuse the common multicast key for all connections. However, it is possible that the sender will use distinct keys for each connection, so that it can send confidential unicast messages to individual listeners as well. For TLS_PSK_WITH_AES_128_CCM_8 a peer's key constitutes of a 16 byte long encryption key and a 4 byte long nonce salt, adding to 20 bytes of key material. Since we asume distinct keys are used for both peers one each connection, we get a total of $99 * (20 + 20)$, or 3960 bytes only for the keys from all connections. Also, for the multicast connection each listener needs to maintain a list of sender ids(1 byte per id) and their associated sequence numbers(5 bytes per individual sequence number). If there are 50 senders, this gives a total of 300 bytes only for the id to sequence number association in the multicast connection. It is very likely that individual listener's lists will have overlapping entries, which is somewhat redundant.

Group awareness: The proposed topology does not address the issue of group members' awareness of each other's existence. The document states that members can join and leave the multicast group. However, since listeners need to distinguish between senders, all concerned listeners need to be made aware when a new sender joins the group.

An alternative approach was proposed in [35]. It suggests that all listeners use the same key when responding to a multicast message. This approach greatly reduces the memory used to store keys. The use of a group id is also proposed. The author, Marco Tiloca, explains that the group id will be distributed to group members by the controller together with the rest of the multicast parameters. The group id is used by listeners for generating responses the same way the sender id is used by the sender when generating requests. The id will occupy the first byte in the sequence number encoding. The last 5 bytes of the sequence number will be set to the same value as that of the sequence number in the associated request. Since both the multicast write key and the multicast

response key are the same for each sender and listener, there is no need to initialize new connections via handshaking.

While this approach improves upon IETF's proposition, it too suffers from a couple of drawbacks. First, all responses are encrypted with the same key, which would allow every member of the group to interpret all responses. Secondly, while the overhead of performing handshakes between senders and listeners is avoided, the memory overhead per connection, while minimized(due to the reuse of the response key for all connections), remains. Furthermore, listener nodes still have to maintain a list of sender ids and related sequence numbers. Last, but not least, the connections between the group controller and the group members still remain.

6.2 Proposed Approach

The approach I propose for enabling multicast support to DTLS aims to address the above listed shortcomings of the existing propositions. It relies on the IoT device classification, presented in Chapter 3, as well as the assumption that the multicast group controller is connected to all devices in the multicast group via unicast DTLS connections. My approach builds on top of the concept of one-to-many multicast distribution and the proxy operation, discusses in Section 4.6 in [32]. When there is a single sender in the multicast group, listeners need not maintain lists of sender ids and sequence numbers. Furthermore, each listener only maintains one unicast connection in addition to the multicast connection. Adding new listeners increases the total number of connections in a linear fashion. In the IETF proposed topology there already is one node, which connects to all others and needs to create a single connection upon including a new group member, and this is the controller. I propose that the already existing connections between the controller and all multicast group members be used to forward messages from the senders through the controller to the listeners. This approach requires that the multicast connection is established between the controller and the listeners in the group. Figure 6.2a and Figure 6.2b depict the aforementioned connections. In my proposed approach senders are still given unique identifiers, which they use the same way as in the other approaches - as the first byte in their sequence number. The last 5 bytes of the sequence number are the actual sequence number used by the sender for its unicast DTLS connection with the controller. The resulting record structure is depicted in Figure 6.1.

Field	Content Type	Version		Epoch	Sequence Number		Length
Encoding	type	major	minor	epoch	sender ID	truncated sequence number	length
Size	1 byte	1 byte	1 byte	2 bytes	1 byte	5 bytes	2 bytes

Figure 6.1: Modified Record Header

Upon receiving a request from a sender, the controller decrypts it using the decryption key for its connection with that sender and generates a multicast message, containing the received request. The controller forms the multicast message's sequence number from the sender's ID, as the first byte, and the controller's sequence number for the multicast connection, as the last 5 bytes. This sequence number is maintained by the controller and is incremented each time a sender sends a message. Once the controller constructs the sequence number for the given message, it sends that message on the multicast connection, encrypted with the common multicast key. This process is depicted in Figure 6.2d. When a listener wants to respond to a request, it includes the sender id, received with the associated message, as the first byte in the response's sequence number. The last 5 bytes in the response's sequence number contain the listener's sequence number for its unicast DTLS connection to the controller. Upon receiving a response, the controller decrypts it with the decryption key for its connection to the corresponding listener and checks the

sender id in the response to identify the targeted sender. The response is then encrypted with the encryption key for the corresponding connection and forwarded to the appropriate sender with the controller's sequence number for the associated connection. This way no participant in the group should mistake new messages for retransmissions.

This strategy has several advantages over the other propositions. These are as follow:

Number of Connections: Using this strategy, the total number of connections required are equal to the number of group members. Also, upon adding a new member to the group, only one new connection is created - between the controller and the new member.

Memory Overhead: Senders maintain a single connection with the controller. Listeners, on the other hand, maintain two connections with the controller - one unicast and one multicast. The multicast connection only requires 1 key, used by the controller to encrypt requests. Additionally, listeners need not maintain a list of sender IDs and sequence numbers, since the controller handles that part. The controller itself is connected to all group members, which is also true in the other approaches. The overhead for the controller is that it also maintains a single multicast connection with the listeners. The overall per-node memory usage is greatly reduced, because group members do not need extra connections between each other. The controller pays a minor price of maintaining a multicast connection and a list of sender IDs. This provides the additional benefit that redundant mappings between sender IDs and sender sequence numbers on each listener node are avoided. Figure 6.2c depicts the information maintained at each node for the proposed strategy.

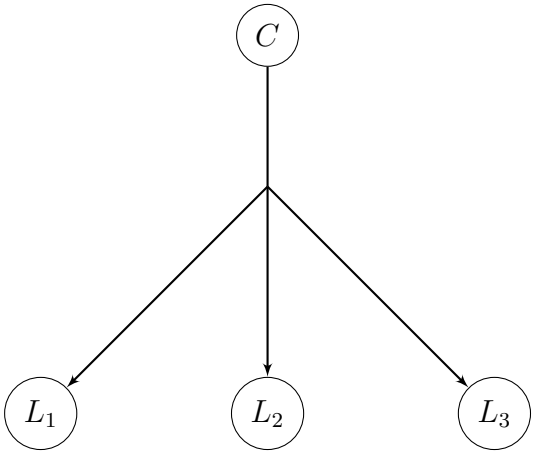
Secure Response: All responses are sent over unicast DTLS connections, which ensures that only the targeted sender can interpret them. Furthermore, listeners cannot interpret other listener's responses.

Group Awareness: Group members need not be aware of each other's existence, because the controller connects them all. This means that no prior knowledge is required for a sender and a listener to communicate.

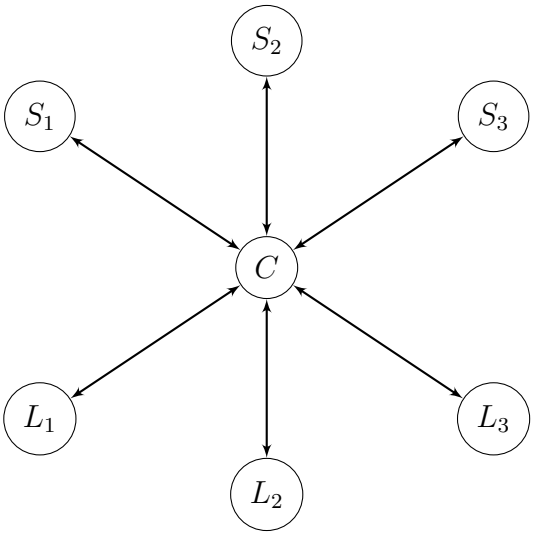
There are a couple of drawbacks with the proposed solution as well.

First, the controller is given the responsibility of forwarding messages back and forth. Unfortunately, this requires that requests and responses are decrypted and the re-encrypted mid way between the sender and the receiver. This is an obvious performance overhead for the controller. In the context of IoT and Smart Home devices not all devices can handle such a task. Therefore, the controller should be a dedicated device with the capacity to buffer requests and responses as needed to handle the higher than usual network traffic.

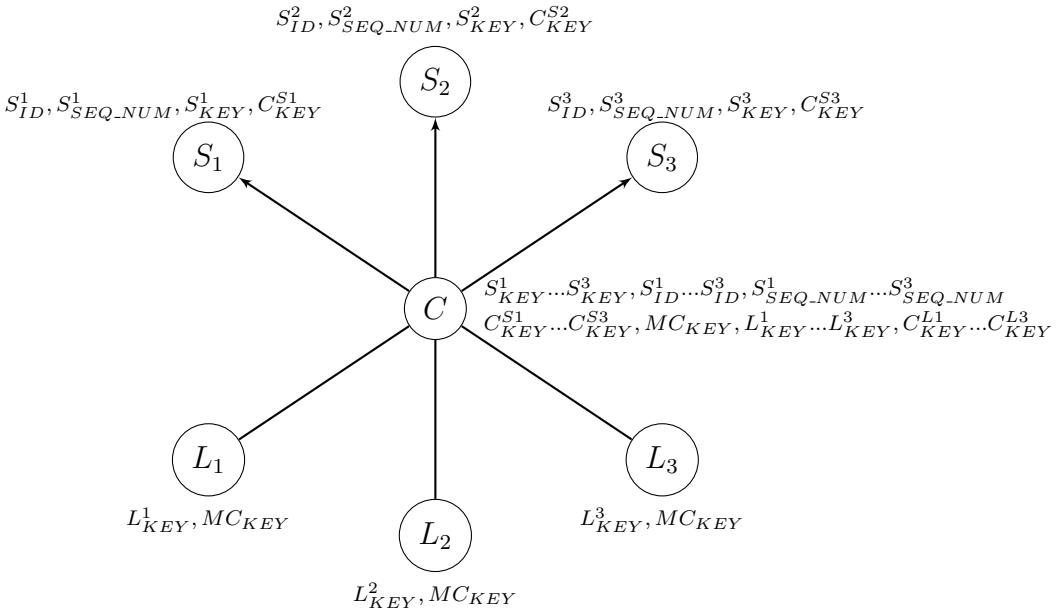
Second, the controller manages multicast keys and communications between senders and listeners. This means that if it is compromised, all other devices in the network will be affected. This means that the device that takes the role of the controller should be safely placed away from physical reach. Furthermore, it should not be connected to the Internet, unless necessary.



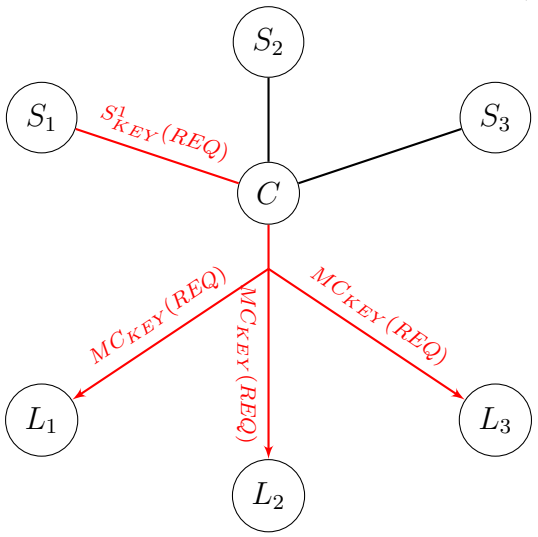
(a) Multicast Connection



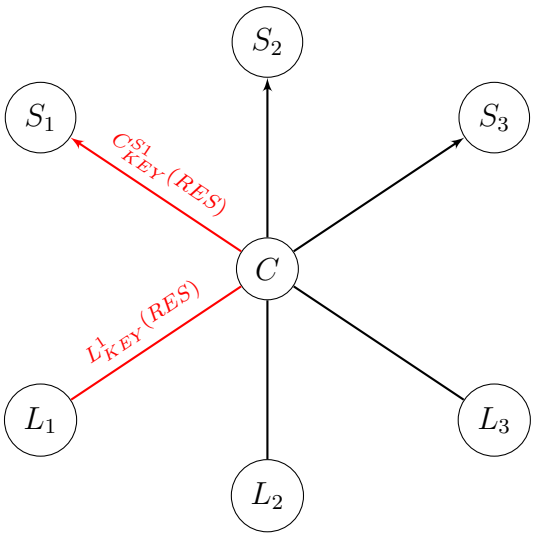
(b) Controller/Group Member Connections



(c) Keys



(d) Request



(e) Response

Figure 6.2: Proposed Topology

7. Testing and Evaluation

Chapter 2 provided details about the targeted hardware. Ideally, testing the my DTLS 1.2 implementation should involve running the cone on the target device. Unfortunately, at the time of writing this report I do not have access to Seluxit's system, which makes it impossible to properly test the implemented functionality. In an attempt to verify the correctness of the code logic, I am currently using a simulation of a server-client communication, which runs on a single machine. The machine in question is almost 5 year old DELL XPS l502x with the following specifications:

CPU: A quad core i7-2670QM with 2.2 GHz base frequency and turbo boost up to 3.1 GHz

RAM: 4GB dual channel DDR3-1333 MHz

Based on the DTLS specification, presented in 4 I have outlined the following test areas:

- Handshake message generation and verification.
- Handshake state transition.
- Timeouts and retransmissions.
- Message encryption and decryption.
- Message fragmentation.
- Compatibility with other implementations.

The simulation I have currently created tests a straight forward handshake between the client and the server. This addresses the message generation and verification, the handshake state transition and the message encryption and decryption functionality.

7.1 Handshake Message Generation and Verification

My initial tests show that handshake messages are properly generated, according to the formats presented in [28] and [63]. However, the generated messages still have to be compared to the ones generated by other implementations.

7.2 Message Fragmentation

As mentioned in Chapter 5 only application data messages are fragmented with the currently adopted approach. The fragmentation behavior is well described in Chapter 5. Current test show that the fragmentation functionality performs as expected.

7.3 Handshake State Transition

The implementation seems to properly transition between handshake messages, handshake states and internal DTLS protocols(Handshake, Change Cipher Spec and Application Data) in the simulation environment. The handshake process flows as specified for the selected cipher suite(see Figure 4.6) and the state transitions properly mimic the defined state machine(see Figure 4.5). Once again, the transition logic has to be tested against another implementation.

7.4 Message Encryption and Decryption

The implementation properly encrypts the outgoing and incoming Finished messages. It uses the nonce and additional data approach defined Section 6.2.3.3 in [63] and [24].

7.5 Timeouts and Retransmissions

As I already mentioned, I run a simulation of client-sever communication as the environment for my tests. This makes it hard to delay messages between the client and the server, as they reside on the same machine and exchange messages via shared streams. I am currently incorporating random delays in the simulation environment to artificially force force a timeout and retransmission behavior in the implementation. I haven't run any conclusive tests so far, so I cannot report on the implementation's timeout and retransmission behavior.

7.6 Performance

Performance is the hardest part to analyze, since the hardware, used for the simulation, bares no similarity to the target device. A full initial handshake, where only the Finished messages are encrypted is completed in ≈ 0.003 seconds. It is hard to use this time to predict performance on the target device for several reasons.

First, message generation and verification will certainly be more time consuming on the target device. Second, AES is software supported on for the simulation hardware, whereas it is hardware supported in the target device. This means on the target device the actual encryption may turn out to be less time consuming that the message generation process. The opposite is true for the simulation hardware. Third, the device will only run the client or the server side. Handshake duration will depend on the performance on both ends as well as the message travel times. In the current simulation all message exchange procedures are internal for the hardware. No real network transmission occur.

Overall, the observed performance seems promising, but no conclusive evaluation is possible for the time being. One important performance test is that of a fully encrypted handshake.

7.7 Compatibility With Other Implementations

As discussed in Chapter 5, it is possible that the client or the server will run a different implementation of the protocol. This means it is crucial that the implementation be compatible with other implementations of the DTLS 1.2 standard. My initial plan was to test for compatibility with tinyDTLS. Unfortunately, upon attempting to compile the tinyDTLS library I run into several compilation errors. I haven't had the time to resolve those yet. Therefore, I cannot make claims of compatibility as of now. However, compatibility testing is at the top of my list for future work.

8. Future Work

The current implementation will have to undergo extensive testing in the targeted environment. Specifically, the currently lacking test cases, listed in Chapter 7 should be addressed as soon as possible.

As discussed in earlier sections, the use of TLS_PSK_WITH_AES_128_CCM_8 requires a shorter, less computationally demanding handshake. There is a currently pending proposition for a "PSK Identity Hint Extension" [67], which can further reduce the handshake procedure. The related documentation [67] explains that the client can include a list of available pre-shared secrets' identities as an extension in the client hello message. If the server selects a cipher suite that uses pre-shared keys from the list of cipher suites in the client hello message, it can also select an identity from the clients list. The server includes the selected identity in an extension in its server hello message. The associated handshake looks like the one in Figure 8.1. Adding support for the extension can improve the performance of the handshake process(fewer transmissions and less messages to hash), which is desired, due to the anticipated frequency of the handshakes.

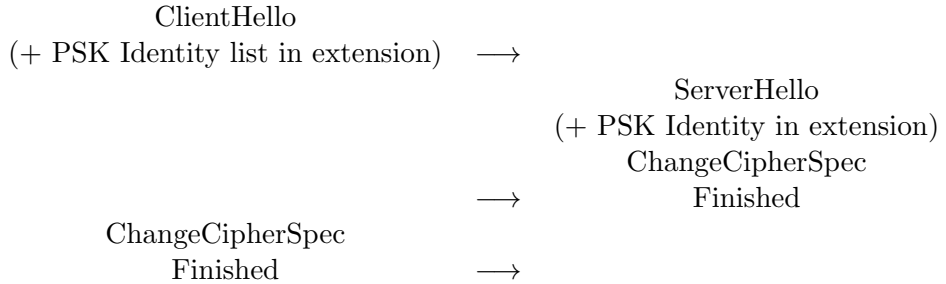


Figure 8.1: Handshake with PSK Identity extension

One improvement I have in mind is the generation of handshake message flights. As of the time of this writing, the implementation generates a single handshake message at a time. The message is encrypted, if need be, and transmitted before the next handshake message in the flight is generated. However, my analysis of the maximum message sizes leads me to believe that generating, encrypting and storing multiple messages in the write stream is possible and worth considering. This will improve message transmission efficiency and reduce the risk of individual messages being lost. However, if the transmission fails for some reason, none of the messages will be received. This modification will not provide a great improvement for cipher suites, which require exchange of big messages(f.x. TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8 with certificates).

The current implementation provides a reasonable security approach, given the targeted environment and the specification of the targeted hardware. However, the selected cipher suite does not provide perfect forward secrecy, which may be a desired feature in environments, where confidential data is exchanged. In such cases data confidentiality and data integrity have equal importance and due to it's nature, data should persistently remain confidential.

The use of TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8 can provide perfect forward secrecy. Including support for that cipher suite will require support for elliptic curve cryptography and possibly functionality for processing and verifying certificates. Additionally, there will be need for a mechanism, ensuring that the same ECDHE parameters are not reused. Alternatively, the the cipher suites proposed in [33] can be adopted once they are formally defined. The document defines two pre-shared secret based cipher suites, namely - TLS_ECDHE_PSK_WITH_AES_128_CCM and TLS_ECDHE_PSK_WITH_AES_128_CCM_8.

Including either of them in the implementation will require support for elliptic curve cryptography. The document also advocates the use of the cipher-based MAC AES-CMAC to effectively remove the need for SHA 256 support. If approved, the use of AES-CMAC will reduce code footprint and reduce the overhead of including DTLS support on embedded devices.

Support for other cipher suites will improve the compatibility of the implementation with other systems. The targeted hardware for this project only has hardware support for AES 128. However, future hardware improvements may provide support for RSA and elliptic curve functionality. This will make the use of cipher suites with alternative key exchange and authentication approaches viable options. Therefore, including support for various cipher suites can improve the applicability of the implementation.

9. Security Considerations

The cipher suite, supported by the implementation, uses AES 128 in CCM mode for message encryption and authentication. It makes use of SHA 256 for pseudo random byte generation. Additionally, pre-shared keys are used for the key exchange procedure. The security level and the future relevance of the implementation will depend on the strengths and weaknesses of the employed algorithms as well as on their resilience to known threats for current security technologies.

9.1 CCM Considerations

In [52] NIST point out that the same nonce-key pair should not be reused, because such reuse may compromise the security of the authenticated encryption procedure. The implementation uses IETF's approach to generating the nonce [see Section 3 in 24], which ensures that each message is encrypted with a unique nonce-key pair. The explicit part of the nonce will always start with the same value upon performing a handshake. However, the implicit part of the nonce as well as the encryption key will be updated each time a handshake occurs. It is, therefore, highly unlikely that the exact same nonce-key pair will be reused in close succession.

9.2 SHA Considerations

As stated in [14] SHA 256 provides security level of 2^{128} and is currently recommended for use for preserving message integrity. To the best of my knowledge, at the time I am writing this report, there are no reports of successful preimage or collision attacks against the complete SHA 256 function. However, [26] presents attacks against reduced versions of SHA 256, which are doable in reasonable amount of time. Nevertheless, SHA 256 should be safe for use in the foreseeable future.

9.3 General TLS/DTLS Attacks

In February 2015 IETF published a list of known TLS attacks [59]. Their report contains 15 distinct attacks, most of which are related to support of old TLS versions, certificates, RSA and Diffie-Hellman functions and stream ciphers. This implementation does not currently support any of these, which means the related attacks have no relevance to it. The report mentions the Padding Oracle Attack, which is applicable to MAC-then-encrypt schemes involving block ciphers. This implementation uses AES 128 in CCM mode, where no padding is used in the encryption step. Therefore, this attack should not be applicable to the implementation. IETF list of known attacks includes 2 attacks, which require additional research. These are - renegotiation and triple handshake attack. I need to investigate the extent to which these attacks apply to this implementation and what recommendations exist of how to protect against them.

9.4 Quantum Computing

Quantum computing is a promising field in computer science, which will enable great computation improvements in some problem solving tasks. Unfortunately for security,

number factorization is one such problem [19]. A big part of cryptography relies on the hardness of factoring big numbers, including the RSA, Diffie-Hellman and elliptic curve functions. Therefore, commercial availability of quantum computers will obsolete the use of these algorithms. Luckily, quantum computers have a long way to go before they become a feasible tool for attackers. In reference with the cipher suite, supported by this implementation, quantum computing poses a much smaller threat. Assuming there are no information leaks, AES, SHA and pre-shared keys are subjects only to brute force attacks. Quantum computing only speeds up brute-force search by a factor of square root. This means that the cryptographic strength of AES 128, SHA 256 and a 16 byte long pre-shared secret will be reduced from 2^{128} to 2^{64}. This is bad, but can be addressed by doubling the AES key sice, the size of the pre-shared key and switching to SHA 512.

9.5 Physical Attacks

If a person can obtain a device, which runs this DTLS implementation, they can effectively read the device's memory(unless its file system is encrypted). This means that all pre-shared secrets for this device will be known and all related connections will be compromised. This impact will be even more severe if the same pre-shared secret is used by multiple devices for a multiple connections. Therefore, each connection should have its own distinct set of pre-shared secrets. This way is a device is compromised all other devices should suppress their connections to the compromised device and can continue exchanging data on over other connections. It is worth mentioning that physical access to a device's memory has the same effect for all key exchange schemes that use fixed parameters(e.g. RSA, non-ephemeral Diffie-Hellman and pre-shared secrets).

10. Conclusion

Smart Home environments are potential gateways to our private lives for intruders. As such, they require that manufacturers invest into incorporating good security mechanisms into their Smart Home solutions. Introducing communication security has a noticeable overhead from the perspective of embedded smart devices. Thankfully, the associated authorities are continuously improving existing security protocols and devising profiles to enable their use in constrained environments. Based on the findings in my research I conclude that a lightweight DTLS 1.2 implementation with support for TLS_PSK_WITH_AES_128_CCM_8 is the simplest way to introduce communication security on embedded devices. It meets the requirements of current standards for securing CoAP and minimizing the overhead of incorporating communication security in systems targeting constrained environments. This approach works best in situations where there is hardware support for AES. For systems with hardware support for elliptic curve cryptography TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8 may be preferable, because of the additional property of perfect forward secrecy. Cipher suites which merge the simplicity of using pre-shared secrets whilst maintaining perfect forward secrecy are being developed and are worth considering in the future.

Careful analysis of the target environment can result in compact implementation of DTLS, while good usage strategies can decrease the runtime RAM requirements.

Strategies for introducing multicast support to DTLS are still not standardized. However, promising ideas are emerging more and more often. IETF's working draft on the subject provides a great theoretical foundation for multicast support in DTLS. Most related papers concentrate on reducing the memory overhead of IETF's proposed scheme through improved key usage. By carefully analyzing the existing propositions I saw potential for memory savings in improving the proposed topology. My analysis of the implication of using a star-like topology shows that it could be a good fit for multicast DTLS purposes. It allows for a simplified, more scalable approach to managing group communications, which does not require further modifications to the standard unicast DTLS. Furthermore, it has lower RAM requirements and equivalent security properties compared to IETF's proposed approach.

Bibliography

[1] Tech. rep. URL: http://www.cisco.com/c/dam/en_us/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf (visited on 05/24/2016).

[2] URL: http://www.intel.com/content/www/us/en/internet-of-things/infographics/guide-to-iot.html (visited on 05/24/2016).

[3] URL: http://www.grandviewresearch.com/industry-analysis/smart-homes-industry (visited on 05/24/2016).

[4] URL: http://www.windowsecurity.com/articles-tutorials/firewalls_and_VPN/VPN-Options.html (visited on 05/30/2016).

[5] URL: https://github.com/cetic/tinydtls (visited on 05/13/2016).

[6] URL: http://postscapes.com/internet-of-things-examples/ (visited on 05/11/2016).

[7] URL: http://postscapes.com/internet-of-things-technologies (visited on 05/11/2016).

[8] URL: https://ibmcai.com/2014/10/16/internet-of-things-iot-important-enabling-technologies/ (visited on 05/11/2016).

[9] URL: https://tools.ietf.org/id/draft-baker-ietf-core-04.html (visited on 05/11/2016).

[10] URL: http://postscapes.com/internet-of-things-protocols (visited on 05/11/2016).

[11] URL: http://courses.cs.vt.edu/~cs5204/fall00/protection/rsa.html (visited on 05/31/2016).

[12] URL: https://www.cs.utexas.edu/~byoung/cs361/lecture52.pdf (visited on 05/31/2016).

[13] URL: http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57pt1r4.pdf (visited on 05/19/2016).

[14] URL: http://www.cisco.com/c/en/us/about/security-center/next-generation-cryptography.html (visited on 05/20/2016).

[15] URL: http://www.ouah.org/ogay/hmac/ (visited on 06/02/2016).

[16] URL: http://www.iana.org/assignments/tls-parameters/tls-parameters.xhtml (visited on 05/21/2016).

[17] URL: https://www.iana.org/assignments/comp-meth-ids/comp-meth-ids.xhtml (visited on 05/21/2016).

[18] URL: http://www.iana.org/assignments/tls-extensiontype-values/tls-extensiontype-values.xhtml (visited on 05/21/2016).

[19] URL: https://www.schneier.com/blog/archives/2015/08/nsa_plans_for_a.html (visited on 06/05/2016).

[20] *200 Information technology - Open Systems Interconnection - Basic Reference Model: The basic model*. 1994. URL: http://www.itu.int/rec/T-REC-X.200/en/ (visited on 05/11/2016).

[21] *2015 State of the Smart Home Report*. Tech. rep. URL: https://www.icontrol.com/wp-content/uploads/2015/06/Smart_Home_Report_2015.pdf (visited on 05/24/2016).

[22] *A DTLS Based End-To-End Security Architecture for the Internet of Things with Two-Way Authentication*. Tech. rep. URL: http://kothmayr.net/wp-content/papercite-data/pdf/kothmayr2012dtls.pdf.

[23] *ADVANCED ENCRYPTION STANDARD (AES)*. Tech. rep. Nov. 26, 2001. URL: http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf (visited on 05/12/2016).

[24] *AES-CCM Cipher Suites for Transport Layer Security (TLS)*. Tech. rep. URL: https://tools.ietf.org/html/rfc6655 (visited on 06/03/2016).

[25] *An Interface and Algorithms for Authenticated Encryption*. Tech. rep. URL: https://tools.ietf.org/html/rfc5116 (visited on 06/01/2016).

[26] *Analysis of SHA-512/224 and SHA-512/256*. Tech. rep. URL: https://eprint.iacr.org/2016/374.pdf (visited on 06/05/2016).

[27] Bill Chamberlin. *Internet of Things (IoT): Important Enabling Technologies*. Oct. 16, 2014. URL: https://ibmcai.com/2014/10/16/internet-of-things-iot-important-enabling-technologies/ (visited on 05/11/2016).

[28] *Datagram Transport Layer Security Version 1.2*. Tech. rep. URL: https://tools.ietf.org/html/rfc6347 (visited on 05/21/2016).

[29] *DTLS Adaptation for Efficient Secure Group Communication*. Tech. rep. URL: http://www.diva-portal.org/smash/get/diva2:847246/FULLTEXT01.pdf (visited on 05/25/2016).

[30] *DTLS In Constrained Environments (dice)*. URL: https://datatracker.ietf.org/wg/dice/charter/.

[31] *DTLS-based Multicast Security in Constrained Environments*. Tech. rep. URL: https://tools.ietf.org/html/draft-keoh-dice-multicast-security-08 (visited on 05/12/2016).

[32] *DTLS-based Multicast Security in Constrained Environments draft-keoh-dice-multicast-security-08*. Tech. rep. URL: https://tools.ietf.org/html/draft-keoh-dice-multicast-security-08 (visited on 05/22/2016).

[33] *ECDHE-PSK AES-CCM Cipher Suites with Forward Secrecy for Transport Layer Security (TLS)*. Tech. rep. URL: https://tools.ietf.org/html/draft-schmertmann-dice-ccm-psk-pfs-01 (visited on 05/23/2016).

[34] *Efficient Protection of Response Messages in DTLS-Based Secure Multicast Communication*. Tech. rep. URL: http://soda.swedish-ict.se/5709/1/Tiloca_SIN2014.pdf (visited on 05/25/2016).

[35] *Efficient Protection of Response Messages in DTLS-Based Secure Multicast Communication*. Tech. rep. URL: http://soda.swedish-ict.se/5709/1/Tiloca_SIN2014.pdf (visited on 05/22/2016).

[36] *GSAKMP: Group Secure Association Key Management Protocol*. Tech. rep. URL: https://tools.ietf.org/html/rfc4535 (visited on 06/01/2016).

[37] *HMAC: Keyed-Hashing for Message Authentication*. Tech. rep. Feb. 1997. URL: https://tools.ietf.org/html/rfc2104.

[38] *Information technology – Abstract Syntax Notation One (ASN.1): Specification of basic notation*. Tech. rep. URL: https://www.itu.int/ITU-T/studygroups/com17/languages/X.680-0207.pdf (visited on 06/02/2016).

[39] *Information technology – ASN.1 encoding rules: Specification of Basic Encoding Rules(BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)*. Tech. rep. URL: https://www.itu.int/ITU-T/studygroups/com17/languages/X.690-0207.pdf (visited on 06/02/2016).

[40] *Internet of things research study*. Tech. rep. URL: http://www8.hp.com/h20195/V2/GetPDF.aspx/4AA5-4759ENW.pdf (visited on 05/24/2016).

[41] *Internet of Things: Standards and Guidance from the IETF*. URL: https://www.internetsociety.org/publications/ietf-journal-april-2016/internet-things-standards-and-guidance-ietf (visited on 05/11/2016).

[42] *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*. Tech. rep. URL: https://tools.ietf.org/html/rfc5280 (visited on 05/12/2016).

[43] *Message Authentication Codes (MACs)*. URL: http://euler.ecs.umass.edu/ece597/pdf/Crypto-Part12-MAC.pdf (visited on 05/12/2016).

[44] *MIKEY: Multimedia Internet KEYing*. Tech. rep. URL: https://tools.ietf.org/html/rfc3830 (visited on 06/01/2016).

[45] *Multicast Security (MSEC) Group Key Management Architecture*. Tech. rep. URL: https://tools.ietf.org/html/rfc4046 (visited on 06/01/2016).

[46] *On the effectiveness of end-to-end security for Internet-integrated sensing applications*. Tech. rep. URL: https://www.google.bg/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&cad=rja&uact=8&ved=0ahUKEwjS6smu6fTMAhWJIpoKHcvLDpwQFggaMAA&url=https://www.cisuc.uc.pt/publication/showfile?fn=1425996365_On_the_effectiveness_of_end-to-end_security_for_Internet-integrated_sensing_applications.pdf&usg=AFQjCNHI7_4mZjz0hfTzllL5qnPEREaMFA (visited on 05/25/2016).

[47] *On the Feasibility of Secure Application-Layer Communications on the Web of Things*. Tech. rep. URL: https://www.cisuc.uc.pt/publication/show/3116 (visited on 05/25/2016).

[48] J Postel. "User Datagram Protocol". In: (1980). URL: https://tools.ietf.org/html/rfc768 (visited on 05/11/2016).

[49] *Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)*. Tech. rep. URL: https://tools.ietf.org/html/rfc4279 (visited on 05/19/2016).

[50] *Public Key Cryptography*. URL: http://www.tutorialspoint.com/cryptography/public_key_encryption.htm (visited on 05/31/2016).

[51] *Recommendation for Block Cipher Modes of Operation*. Tech. rep. Dec. 2001. URL: http://csrc.nist.gov/publications/nistpubs/800-38a/sp800-38a.pdf (visited on 05/12/2016).

[52] *Recommendation for Block Cipher Modes of Operation: The CCM Mode for Authentication and Confidentiality*. Tech. rep. May 2004. URL: http://csrc.nist.gov/publications/nistpubs/800-38C/SP800-38C_updated-July20_2007.pdf (visited on 05/12/2016).

[53] *Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication*. Tech. rep. May 2005. URL: http://csrc.nist.gov/publications/nistpubs/800-38B/SP_800-38B.pdf (visited on 05/12/2016).

[54] *Secure Hash Standard (SHS)*. Tech. rep. URL: http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf (visited on 06/03/2016).

[55] *Secure Multicast with Source Authentication for the Internet of Things*. Tech. rep. URL: http://www.diva-portal.org/smash/get/diva2:739094/FULLTEXT01.pdf.

[56] *Security and Resilience of Smart Home Environments*. Tech. rep. Dec. 1, 2015. URL: https://www.enisa.europa.eu/publications/security-resilience-good-practices (visited on 05/14/2016).

[57] *Security Architecture for the Internet Protocol*. Tech. rep. URL: https://tools.ietf.org/html/rfc4301 (visited on 05/12/2016).

[58] Zach Shelby, Klaus Hartke, and Carsten Bormann. "The constrained application protocol (CoAP)". In: (2014). URL: https://tools.ietf.org/html/rfc7252 (visited on 05/11/2016).

[59] *Summarizing Known Attacks on Transport Layer Security (TLS) and Datagram TLS (DTLS)*. Tech. rep. URL: https://tools.ietf.org/html/rfc7457 (visited on 06/05/2016).

[60] *Terminology for Constrained-Node Networks*. Tech. rep. URL: https://tools.ietf.org/html/rfc7228 (visited on 05/30/2016).

[61] *The Internet of Things: Security Research Study*. Tech. rep. URL: https://www.veracode.com/sites/default/files/Resources/Whitepapers/internet-of-things-whitepaper.pdf (visited on 05/24/2016).

[62] *The Multicast Group Security Architecture*. Tech. rep. URL: https://tools.ietf.org/html/rfc3740 (visited on 06/01/2016).

[63] *The Transport Layer Security (TLS) Protocol Version 1.2*. Tech. rep. URL: https://tools.ietf.org/html/rfc5246 (visited on 05/21/2016).

[64] *Threat Landscape and Good Practice Guide for Smart Home and Converged Media*. Tech. rep. Feb. 9, 2015. URL: https://www.enisa.europa.eu/publications/threat-landscape-for-smart-home-and-media-convergence (visited on 05/14/2016).

[65] *Timed Efficient Stream Loss-Tolerant Authentication (TESLA): Multicast Source Authentication Transform Introduction*. Tech. rep. URL: https://tools.ietf.org/html/rfc4082 (visited on 06/01/2016).

[66] *TLS/DTLS Profiles for the Internet of Things*. Tech. rep. URL: https://tools.ietf.org/html/draft-ietf-dice-profile-17 (visited on 05/12/2016).

[67] *TLS/DTLS PSK Identity Extension*. Tech. rep. URL: https://tools.ietf.org/html/draft-jay-tls-psk-identity-extension-01 (visited on 05/23/2016).

[68] *Transport Layer Security (TLS) Extensions: Extension Definitions*. Tech. rep. URL: https://tools.ietf.org/html/rfc6066 (visited on 05/22/2016).

[69] International Telecommunication Union. *The Internet of Things*. International Telecommunication Union, 2005. URL: http://www.itu.int/net/wsis/tunis/newsroom/stats/The-Internet-of-Things-2005.pdf (visited on 05/11/2016).

[70] *Using CoAP with IPsec*. (Visited on 05/12/2016).

[71] *Using Raw Public Keys in Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)*. Tech. rep. URL: https://tools.ietf.org/html/rfc7250 (visited on 05/19/2016).

[72] *What Exactly Is The "Internet of Things"?* URL: http://postscapes.com/what-exactly-is-the-internet-of-things-infographic (visited on 05/11/2016).

[73] *What the Internet of Things (IoT) Needs to Become a Reality*. Tech. rep. Freescale, ARM. URL: http://www.nxp.com/files/32bit/doc/white_paper/INTOTHNGSWP.pdf (visited on 05/11/2016).

A. Smart Home Vulnerabilities