DEGREE PROJECT, IN MASTER'S PROGRAMME COMMUNICATION SYSTEMS
SECOND LEVEL

STOCKHOLM, SWEDEN 2015

DTLS Adaptation for Efficient Secure Group Communication

KIRILL NIKITIN

KTH ROYAL INSTITUTE OF TECHNOLOGY

SCHOOL OF INFORMAT ION AND COMMUNICATION TECHNOLOGY

Abstract (English)

The Internet has been growing rapidly during the past three decades, evolving into a new paradigm called the *Internet of Things* where all electronic devices are to be connected to the global network. One of the most crucial needs for communication in this future global network is assuring its security. Datagram Transport Layer Security is a de facto standard protocol to secure end-to-end communication in the IoT. However, there is also an increasing need for secure and efficient group communication due to the frequently resource-constrained IoT environment. A DTLS adaptation for multicast communication has been already proposed but protection of responses to multicast requests has not been fully addressed yet. Furthermore, there is no publicly available implementation of this adaptation.

This thesis work is conducted in collaboration with SICS Swedish ICT which is a research organization with a focus on applied Computer Science. We have implemented the existing DTLS-based approach for multicast communication for the Contiki OS. We have also implemented an approach for efficient response protection that was initially proposed at SICS and that we analyse and enhance in this thesis. Finally, we have experimentally evaluated our and other approaches on a constrained hardware platform in terms of memory requirements, communication performance and energy consumption. We demonstrate advantages of our approach based upon obtained results.

Keywords

DTLS, secure group communication, multicast, Internet of Things, efficient security

Abstract (Swedish)

Internet har vuxit snabbt under de tre senaste decennierna, och utvecklas till ett nytt paradigm kallat Internet of Things där alla elektroniska enheter kopplas till det globala nätverket. Ett av de viktigaste behoven för kommunikation i detta framtida globala nätverk är att garantera säkerhet. Datagram Transport Layer Security protokollet är en de facto-standard för säker end-to-end kommunikation i IoT. Det finns emellertid också ett ökande behov av säker och effektiv gruppkommunikation på grund av att IoT-miljön i regel är resursbegränsad. En anpassning av DTLS för multicast-kommunikation har föreslagits men skydd av svar på multicast-förfrågningar har ännu inte till fullo behandlats. Dessutom finns ingen offentligt tillgänglig implementation av denna anpassning.

Detta examensarbete utförs i samarbete med SICS Swedish ICT som är en forskningsorganisation med fokus på tillämpad datavetenskap. Vi har implementerat den existerande DTLS-baserade lösningen för multicast-kommunikation för operativsystemet Contiki. Vi har också implementerat en lösning för effektivt skydd av svar på förfrågningar som ursprungligen föreslogs hos SICS och som vi analyserar och förbättrar i detta examensarbete. Slutligen har vi experimentellt utvärderat vår och andra metoder på en begränsad hårdvaruplattform när det gäller minnesbehov, kommunikationsprestanda och energiförbrukning. Vi visar fördelar med vår metod baserat på de erhållna resultaten.

Nyckelord

DTLS, säker gruppkommunikation, multicast, Internet of Things, effektiv säkerhet

Preface

This thesis is a final phase of a master's programme in Communication Systems at KTH Royal Institute of Technology, Stockholm. The thesis work is conducted at SICS Swedish ICT AB which is a research institute with a focus on applied Computer Science. The academic supervisor and examiner is Prof. Markus Hidell from COS Network Systems Lab at KTH. The advisers from SICS are Marco Tiloca and Shahid Raza.

The thesis deals with secure efficient group communication in constrained environments that could be applied in the Internet of Things.

Stockholm, 20-June-2015

Kirill Nikitin

Acknowledgements

I would like to thank Marco Tiloca and Shahid Raza, my advisers at SICS, for their guidance and involvement during my work on this thesis. Active discussions with them and their suggestions greatly helped to improve this work. Moreover, I thank Rikard Höglund who provided me with his valuable assistance at the moments of faced difficulties.

I am also thankful to my academic supervisor Prof. Markus Hidell for his helpful feedback and suggestions, and him being a great teacher.

I am sincerely grateful to The Swedish Institute which generously supported me with a scholarship throughout the whole two-year period of my master's programme studies.

I would like to close these acknowledgements with expression of measureless gratitude to my parents and grandparents who have been supporting me at every single moment of my entire life.

List of Acronyms and Abbreviations

6LoWPAN	IPv6 over Low power Wireless Personal Area Networks
AEAD	Authenticated Encryption with Associated Data
AES	Advanced Encryption Standard
AH	Authentication Header
CBC	Cipher Block Chaining
CoAP	Constrained Application Protocol
DICE	DTLS in Constrained Environments
DNS	Domain Name System
DoS	Denial of Service
DTLS	Datagram Transport Layer Security
ESP	Encapsulating Security Payload
GSA	Group Security Association
GSAKMP	Group Secure Association Key Management Protocol
GSPD	Group Security Police Database
HMAC	Hash-based Message Authentication Code
HTTP	HyperText Transport Protocol
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IKE	Internet Key Exchange

IoT	Internet of Things
IPv6	Internet Protocol version 6
IT	Information Technology
IV	Initialization Vector
LLN	Low-Power and Lossy Network
MAC	Message Authentication Code
M2M	Machine-to-Machine
NAT	Network Address Translation
OS	Operating System
RAM	Random Access Memory
ROM	Read-Only Memory
PRF	Pseudo-Random Function
REST	Representational State Transfer
SA	Security Association
SHA	Secure Hash Algorithm
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UDP	User Datagram Protocol
URI	Uniform Resource Identifier
USB	Universal Serial Bus

List of Figures

List of Tables

Table of Contents

Chapter 1

Introduction

The Internet has been growing rapidly during the past three decades, evolving from a small network for specific professional tasks into interconnection of all possible types of electronic devices [1]. It is fairly safe to claim that the modern society is moving towards the world where everything and everyone will be connected to the Internet [2]. This future technological environment has been named *the Internet of Things*.

The Internet of Things [3] is a paradigm of a global network interconnecting different objects which can harvest information from the environment, interact with the physical world and provide different services. An object, or *a thing*, is often a smart device embedded with electronics, software, possibly sensors and connectivity. Examples of such devices can be smart meters, medical sensors, smart grid, or lighting systems [4, pp. 13–17] but also more traditional ones such as laptops, smartphones and personal computers [2]. Being interconnected, the devices are able to communicate and exchange data with each other or with a control node. The devices in the IoT are often of a constrained nature, particularly having limited battery capacity and processing capabilities, therefore the communication must meet several requirements where efficiency is one of the most crucial needs. In order to increase the efficiency, group communication and, particularly, a multicast approach [5, p. 9] can be used in cases when a node needs to deliver an identical message to several recipients. An example here can be switching on outdoor lighting when an identical “switching” message is to be transmitted to several smart bulbs. Moreover, sometimes a sender expects to receive responses from listeners of a multicast group as, for instance, in a case when a control node requests temperature or humidity information from sensors installed in a building and waits for receiving the necessary data.

Another important need for the IoT is a possibility for secure end-to-end communication. This primarily requires lightweight encryption schemes to protect sensitive data, adequate key-management systems, and mutual authentication of communication parties [6]. Protection can be required, for example, for data provided by bio-sensors on bacterial composition of the product used to guarantee required quality in the food industry [3]. The data in this case is clearly confidential because its uncontrolled spreading may harm reputation of the food-producing company. Cryptographic computations are generally highly recourse-consuming tasks, therefore security solutions for the IoT should be designed taking efficiency constraints into account. Establishing one secure multicast session with one sender and multiple listeners instead of establishing multiple unicast sessions among them may significantly increase communication efficiency since the session establishment can be the most resource-consuming part of the whole process. Moreover, there is an evident advantage for the sender of performing encryption procedure only once for a multicast message instead of doing it repeatedly for each unicast message to members of the multicast group. In addition, reusing the same key material of the multicast secure session for the protection of response messages can be beneficial due to avoiding establishing separate sessions for it.

In this thesis we provide implementation for secure multicast communication in the IoT which, to best of our knowledge, does not yet exist in open access, design and implement a protection mechanism for unicast responses to multicast requests, integrate the mechanism in the multicast implementation and experimentally evaluate our developed solutions.

1.1 Background

The general architecture of the Internet of Things consists of several layers where each layer is responsible for different functionality [7]. The Perception Layer is responsible for identifying objects and gathering information. Protocols from the 802.15.4 family [8] can be used there. The Network Layer, consisting of such protocols as Internet Protocol version 6 (IPv6) [9] and User Datagram Protocol (UDP) [10], processes and transmits the information obtained from upper layers. There is a 6LoWPAN compression mechanism [11] in between the Perception and Network layers which enables IPv6 packets to be carried on top of low power wireless networks by minimizing significantly the size of the IPv6 and UDP headers. Finally, the Application layer provides different services depending on a need. The Constrained Application Protocol (CoAP) [12] is typically used in the Application Layer in the IoT. It is a standard web transfer protocol for use with constrained nodes and constrained networks. The CoAP can be called a lightweight version of HyperText Transfer Protocol (HTTP) [13] since it provides similar functionality including request/response interaction between application endpoints and built-in discovery of services. For detailed description of the protocol please refer to Section 2.2 of Background and Related Work Chapter.

The IoT already has a wide range of applications in multiple areas including transportation and logistic, smart environment, personal and social domain, and security systems [4], [14]. Connection of more and more objects into the network will create steeply rising amount of traffic and require bigger data storages [15]. In order to handle this challenge, solutions for more efficient usage of existing bandwidth have to be adopted. Group communication and, particularly, IP multicast is one of such solutions. However, the laying above CoAP must be adapted accordingly to achieve efficient multicast communication in low-power networks. An approach for using CoAP on top of IP multicast and a transport protocol has been presented in [16].

One of the specific features of the IoT is growing presence of low-power and lossy networks which are sometimes the only option for some of application domains. Under such conditions, an unreliable transport protocol, such as UDP, has to be used to achieve desirable efficiency, whereas delivery assurance is handled by upper layers protocols such as CoAP. However, it produces a difficulty when a question of securing data between communicating applications arises. The Transport Layer Security (TLS) [17] protocol is a de-facto standard for end-to-end communication security over the Internet nowadays. The particular part of TLS is that it is designed to work on top of a reliable transport protocol such as Transmission Control Protocol (TCP) [18] what makes usage of the protocol in the IoT problematic. Fortunately, there is a solution to this problem, named Datagram Transport Layer Security (DTLS) protocol [19]. DTLS is based on TLS in its design and provides equivalent security guarantees, being adapted for operation over the unreliable transport. Design of TLS is followed in DTLS as much as possible to avoid unnecessary security inventions and increase code base reuse [19]. There are two major challenges of TLS, preventing it from use in the IoT that have been solved in DTLS:

1. TLS does not allow independent decryption of individual records because the procedure of decryption depends on a sequence number, derived from TCP header. DTLS solves this problem by banning stream ciphers and introducing explicit sequence numbers in its header.
2. If any of the handshake messages are lost, the TLS protocol breaks because it assumes that the messages are always delivered reliably. To solve this difficulty, DTLS introduce retransmission timer and tolerates not in-order handshake messages receiving.

Like TLS, DTLS is clearly designed for unicast communication in a client/server model. However, it is highly desirable to be able to secure CoAP communication over IPv6 multicast with DTLS since group communication is also vulnerable to the usual attacks over the air. Garcia-Morchon *et al.* proposed an adaptation of the DTLS record layer to provide the desirable protection [20]. The core idea is to modify the DTLS header and, thereby, avoid reuse of the same sequence numbers by different senders. Random IDs are assigned to every sender and, furthermore, every listener needs to keep track of these IDs to ensure replay protection in addition to message integrity and confidentiality.

Nevertheless, discussing the protection of multicast requests, Garcia-Morchon *et al.* do not consider efficient approaches for securing unicast response messages from listeners. It suggests establishing separate unicast security sessions in this case which brings to naught all the efforts of increasing efficiency in a security parameters agreement process, attained by establishing one multicast secure session instead of multiple unicast ones. One approach of efficient response message protection has been recently proposed by Tiloca in [21]. It has been proposed that the same key group material of a multicast session can be reused for securing the unicast response messages. It requires modification of the DTLS header for these messages in the way that a unique ID for every multicast group must be introduced so the sender can distinguish unicast answers from different multicast groups. However, we have discovered that some features of this approach may drastically compromise the security level of the whole communication. We discuss the potential problems in detail in the Chapter 3 and design an improved solution that mitigates the problems and meets efficiency requirements at the same time.

Finally, an environment that we use for management of recourse-constrained devices is the open source Contiki Operating System [22] which is designated for use in the IoT. It can run on a range of low-power wireless devices and has an active community of developers. Moreover, Contiki OS includes CoAP and DTLS libraries for unicast communication. Erbium [23] is a low-power REST Engine for Contiki based on CoAP and tinyDTLS [24] is a lightweight implementation of DTLS for resource-constrained networks. We use them as a base for our implementation of multicast support.

1.2 Problem

There is currently no implementation of the mechanism for secure multicast communication for the CoAP application protocol. The DTLS security protocol, which is in use with the CoAP in the IoT, is implemented at the moment only with support of unicast communication. Is secure multicast communication in IoT achievable and how should it be designed and implemented?
Furthermore, there is no agreed approach on securing unicast responses to multicast requests. Therefore, how can responses to multicast requests be protected efficiently and how can the efficiency be evaluated?

1.3 Purpose

This thesis work provides implementation of DTLS-based multicast security protocol that, to the best of our knowledge, does not exist yet in open access. Furthermore, it presents an analysis of current approaches for protection of responses to multicast requests, reveals existing vulnerabilities and designs an improved approach for efficient response message protection. The designed approach is also implemented and integrated into the multicast implementation. The result is a library for complete, efficient and secure group communication to be used in the IoT. Finally, evaluation results, demonstrating advantages of the designed approach, are presented.

1.4 Goal

The goal of this thesis is:

- To create a framework for efficient secure group communication which can be, particularly, used in the Internet of Things.

In order to reach the goal, several tasks need to be accomplished:

1. To implement DTLS-based multicast security protocol;
2. To design, analyze and implement a novel approach for response message protection;
3. Experimentally evaluate the new approach and approaches that have been proposed before.

The deliverables are:

1. The implementation of CoAP group communication;
2. The implementation of multicast support for DTLS;
3. The extended implementation for secure CoAP group communication;
4. Design and validation of the approach for efficient response message protection;
5. The implementation of complete secure group communication;
6. Results of experimental evaluation.

The results are:

1. Implementation of a protocol for efficient secure group communication;
2. Theoretical analysis of a proposed earlier response protection mechanism for group communication, discovering presenting vulnerabilities in it;
3. Design of an improved approach for response message protection;

4. Evaluation results, demonstrating advantages of the designed approach and group communication in general in terms of memory requirements, communication performance and energy consumption.

The implemented framework will be publicly available at:

https://github.com/nikirill/tinygroupdtls

1.4.1 Benefits, Ethics and Sustainability

The popularity of the IoT is growing rapidly. Companies provide solutions incorporating low-power devices for different areas. These companies include IT giants such as Intel [25] and Cisco [26] but also smaller companies [27]. Simultaneously, individual users take advantage of smart technologies and install devices in private environment. Both groups can benefit from the implementations and solutions presented in this thesis. It can become a part of a commercial product, used in home system or be taken as a base for further research in the area.

The outcomes of the thesis may become a part of the long-standing ethical discussion whether a government and federal agencies should have access to all personal and public data in a country to provide a desirable level of security against malicious activities. DTLS is currently known as secure and relies on ciphers that have not been broken yet, therefore only a person who knows a secret key is able to access encrypted data. As a result, no one else will be able to learn the content, even if this one is the police.

Cisco and similarly Ericsson estimated [28], [29] that there would be 50 billions IoT connected devices in the world by 2020. This means that all the devices will be consuming electric energy for their activity. Under such conditions, it becomes crucial to develop systems which rely on low-power devices and, thereby, can consume less energy for accomplishing the same tasks. Moreover, it is not feasible to connect all the devices to power grid due to their huge amount and, therefore, many of them may be supplied using battery power. Disposal and recycling of depleted batteries is already a problem even with a current usage rate because the batteries contain substances that may harm the environment if they are not taken care properly. This problem will undoubtedly grow along with increasing amount of batteries being used. Simultaneously, security is considered as a vital need in the modern society so one of the tasks is to provide desirable security with minimal energy expenditures. In our thesis, we do not only implement an existing security protocol for group communication in low-power networks but also design and develop improvements that increase its efficiency and reduce overall energy consumption.

1.5 Methodology / Methods

The Quantitative research method, particularly the Experimental and the Applied research methods [30], have been used during the work on this thesis. The implementation of the DTLS-based multicast security protocol can be considered as development of a practical application related to a specific need; whereas design, analysis and evaluation of the response protection mechanism are related to experimental and analytical work. There are discovered problems in existing solutions, that emerge in specific situations and that have to be resolved.

The research is driven with Positivism [31] as a philosophical assumption in mind. The outcomes are expected to work in any suitable environment and do not depend on an observer. The work also includes testing performance which is typical for Positivism [30]. Drawing conclusions are based on the Deductive approach [31], [32] because we experimentally test the theory that our proposed approach outperform ones that were proposed earlier.

The Experimental research [30] has been used as a research methodology. Modifying the implementation of the multicast protocol and the approach for response message protection, we aim at discovering the best possible configuration in terms of security and efficiency. When the configuration is found, experimental verification of its quality is to be performed.

Lastly, it should be mentioned that all figures in this thesis have been examined for being informative for color blind people.

1.6 Delimitations

Group key management is outside of the scope of this thesis. This includes group key distribution schemes, group key refreshment and group key revocation mechanisms. Instead, we manually provide applications with group security parameters and keying material.

If we included this topic in our studies, it could have affected evaluation results by increasing time needed for performing multicast operations at least by amount of time needed for propagation of parameters from a group controller to group members.

1.7 Outline

Chapter 1 provides a reader with an introduction and motivation of this work. Chapter 2 presents background and related work information that are necessary for complete understanding of the thesis. The Chapter 3 presents theoretical research contribution of the author, including analysis of existing work and design of a novel approach. Implementation details and related explanations are covered in the first part of Chapter 4. The second part of Chapter 4 describes how an experimental setup is arranged, experimental scenario and settings, and demonstrate and discuss evaluation results. Finally, conclusions and future work are discussed in Chapter 5.

Chapter 2

Background and Related work

This chapter introduces necessary background information related to this thesis including several technologies and protocols to ensure complete understanding of the presented content. Firstly, we discuss recourse-constrained IoT networks, their structure and protocol stack used. The second section describes the Constrained Application Protocol which is commonly used as an application protocol in the IoT. The description includes specifications for CoAP group communication. The Section 2.3 describes Datagram Transport Layer Security (DTLS) that is a de facto standard end-to-end security protocol in the IoT. The Section 2.4 presents an approach for securing multicast communication based on the DTLS protocol. The Section 2.5 describes an existing approach for efficient protection of unicast responses to multicast requests. The last section 2.6 discusses related work, mainly focusing on IPsec and its application in multicast communication and the IoT.

2.1 Resource-Constrained IoT Networks

One of the motive ideas for the Internet of Things is to interconnect networks of devices that may have limited power, memory and processing capabilities. The networks where most connected devices are constrained are called Low-Power and Lossy Networks (LLNs) [33]. LLNs can be comprised of anything from several devices to a thousand of them and, at the same time, be characterized by high loss rates, instability, and low data rates. As a result, technologies and solution for the IoT have to be designed with intention to minimize energy consumption, amount of information sent over the air and amount of data stored in memory. This imposes limitations on the size of a packet transmitted without fragmentation, communication patterns and ability to provide reliable delivery of data. The protocols of the traditional Internet such as HTTP and TCP are not optimized for low-power communication due to verbose headers, carried meta-data and requirements of reliability [34]. As a result, specific protocols suitable for the IoT have been designed and own IoT protocol stacks have been adapted. Protocols used in the layers of the IoT stack may differ depending on task, performing by network devices. Here, we present ones that are commonly used and can be referred as a CoAP-based protocol stack [35]. These are the protocols that are used in our implementation and evaluation work.

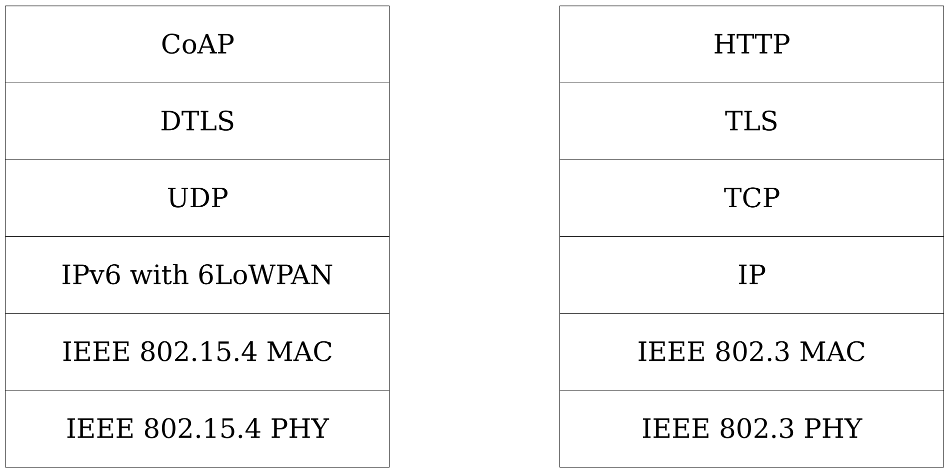


Figure 2.1. CoAP-based IoT protocol stack (on the left) and Web-based protocol stack (on the right)

Figure 2.1 depicts the IoT protocol stack on the left and a traditional Internet protocol stack with corresponding layers on the right. In the depicted IoT stack IEEE 802.15.4 PHY and IEEE 802.15.4 MAC [8] define low-power physical and medium access control layers respectively upon which most IoT technologies are built [34]. The IEEE 802.15.4 protocols family have been designed with intention of saving energy and increasing efficiency in mind so devices are allowed to switch from low-power states and avoid expensive modes such as transmission, reception and channel listening [36]. The upper layers facilitate integration of low-power wireless networks into the IoT.

IPv6 over Low power Wireless Personal Area Networks (6LoWPAN) [37] is often referred in the IoT protocol stack as an independent Adaptation layer. It solves the issues of how to carry IPv6 packets in 802.15.4 frames and how to perform necessary IPv6 neighbour discovery functions. The first issue is addressed by defining fragmentation and reassembly mechanism as well as introducing link, network and transport layers header compression by removing redundant information across the link [38]. 6LoWPAN provides functionality for both unicast and multicast address mapping which is important for the topic of our work.

The IoT is excepted to interconnect billions of devices which can be addressed without the requirement of specialized NAT techniques [39] at the gateways. Therefore, IPv4 has been excluded from the IoT, and IPv6 is an only used IP protocol at the network level to ensure sufficient addressing space. Beside large addressing space, IPv6 has other advantages over IPv4 such as more efficient routing, more efficient packet processing and security improvements.

The transport layer in the IoT stack is responsible for message delivery between communicating applications. UDP is a transaction oriented protocol which does not provide guarantees for message delivery and duplicate protection. However, it becomes a benefit in LLNs because a reliable transport protocol can saturate rapidly low-bandwidth links and, as result, make it very inefficient. Moreover, the efficiency is facilitated by minimizing protocol overhead and mechanisms.

On the top of the network and transport layers in the IoT stack, the security protocol DTLS and the application protocol CoAP are located. Since these protocols are crucial for the work of this thesis, we discuss them in detail in the next sections.

2.2 Constrained Application Protocol (CoAP)

Constrained Application Protocol (CoAP) [12] is a request/response protocol, specially designed for use in low power and lossy networks. It can be easily translated into HTTP for integration into the traditional Web but additionally provides multicast support and has low overhead. The summary of the main features addressed by CoAP [34]:

- Web protocol fulfilling M2M requirements in constrained environments.
- Binding to UDP with optional reliability for unicast requests.
- Asynchronous message exchanges.
- Low header overhead and parsing complexity.
- URI and Content-type support.
- Simple proxy and caching capabilities.
- Optional resource discovery.
- A stateless HTTP mapping, allowing proxies to be built providing access to CoAP resources via HTTP or for HTTP interfaces over CoAP.
- Security binding to Datagram Transport Layer Security (DTLS).

The interaction model of CoAP is similar to the client/server model of HTTP. However, in machine-to-machine interaction both endpoints usually act as clients and servers. The structure of CoAP can be logically divided in two layers. The Message layer deals with asynchronous communication over UDP by providing reliability and sequencing, whereas the Request/response layer works with mapping requests to responses and their semantics.

The Message layer controls the exchange of messages over the network. Requests and responses share the same message format with fixed-length header of 4 bytes, possible options and a payload. Every message carries a Message ID that is a basis for duplicate detection and for optional reliability. It is usually incremented by one for every new message so ID reuse is avoided for given lifetime. There are four message types in CoAP:

- *Confirmable (CON)*: messages that provide reliability. A CON message must be answered with an acknowledgement, otherwise it is retransmitted after a timeout. The Acknowledgement has to carry the same Message ID to be accepted.

- *Non-Confirmable (NON)*: for messages that do not need to be delivered reliably. Nevertheless, they still carry a Message ID, so duplicate detection is present.
- *Acknowledgement (ACK)*: messages confirming reception of a CON message. They can also piggyback a response to the request.
- *RESET (RES)*: sent if a received message cannot be processed.

A CoAP message carry request and response semantics which are a Method code or a Response code and optional information such as an URI or a payload type. The messages can also include a Token field where a token is random for every message unlike a Message ID. Tokens are used to match responses to requests independently from previous messages since the responses may arrive out of order or be lost without notice. This field is strongly recommended for use by the standard to provide at least basic security at the CoAP level. The CoAP message format is presented in Figure 2.2.

Version (2)	Type (2)	Token Length (4)	Code (8)	Message ID (16)
Token (if any)				
Options (if any)				
0xFF	Payload (if any)			

Figure 2.2. CoAP message format with a number of corresponding bits in brackets

As HTTP does, CoAP provides four basics methods: GET, PUT, POST, and DELETE. GET is used to retrieve information from a server, PUT updates or creates resources on a server, POST may create a new resource or update a target one, DELETE deletes the requested resource. Responses can be Success, Client Error meaning that a client incurred in some error, or Internal Server Error when a server is not able to carry out the request.

Another difference of CoAP from HTTP is that it supports making requests to an IP multicast group. This is defined by several deltas to unicast CoAP. We discuss it in the following subsection.

2.2.1 CoAP Group Communication

CoAP group communication implies one-to-many relationship between CoAP endpoints. Specifically, a CoAP client can get or set resources on several servers using CoAP over IP multicast. The technology has been recently standardized by Internet Engineering Task Force (IETF) [16].

In the case of sending a multicast request, the message must be Non-Confirmable and addressed to a multicast IP address instead of a CoAP endpoint. A server may ignore the received multicast request, especially if it does not have anything to answer. If the server does want to answer, it should not, however, send a response immediately but pick a period of time in which it intends to respond. Then the server picks a random point during this time period and sends its UNICAST response. The artificial waiting time is introduced to avoid collisions between unicast responses of different senders. If the approach is not used, the collisions may become very sensible, especially when the amount of nodes reaches a high number like a hundred. Finally, when a client obtains an unicast response to its multicast request, it must check only for token match but not message id.

2.3 Datagram Transport Layer Security (DTLS)

Datagram Transport Layer Security (DTLS) [19] is a security protocol that runs over unreliable datagram transport. It arranges a secure channel between applications on a client and a server that provides communication privacy. The structure of DTLS can be divided into two layers. The first layer can be referred as Handshake layer and includes Handshake, Alert and Change Cipher Spec protocols, whereas the second layer is referred as Record layer and only contains Record protocol. The first layer is responsible for establishing and resuming secure sessions which includes negotiation of cipher suites and exchanging data for the generation of session keys. The Record layer has functions of processing incoming and outgoing data.

While the Alert and Change Cipher Spec protocols can be rather described as supportive protocols for signaling about errors and transition of a cipher suite, the Handshake protocol performs a main function of authentication and negotiation of encryption, hash and compression algorithms. The Handshake data exchange in DTLS looks as follows:

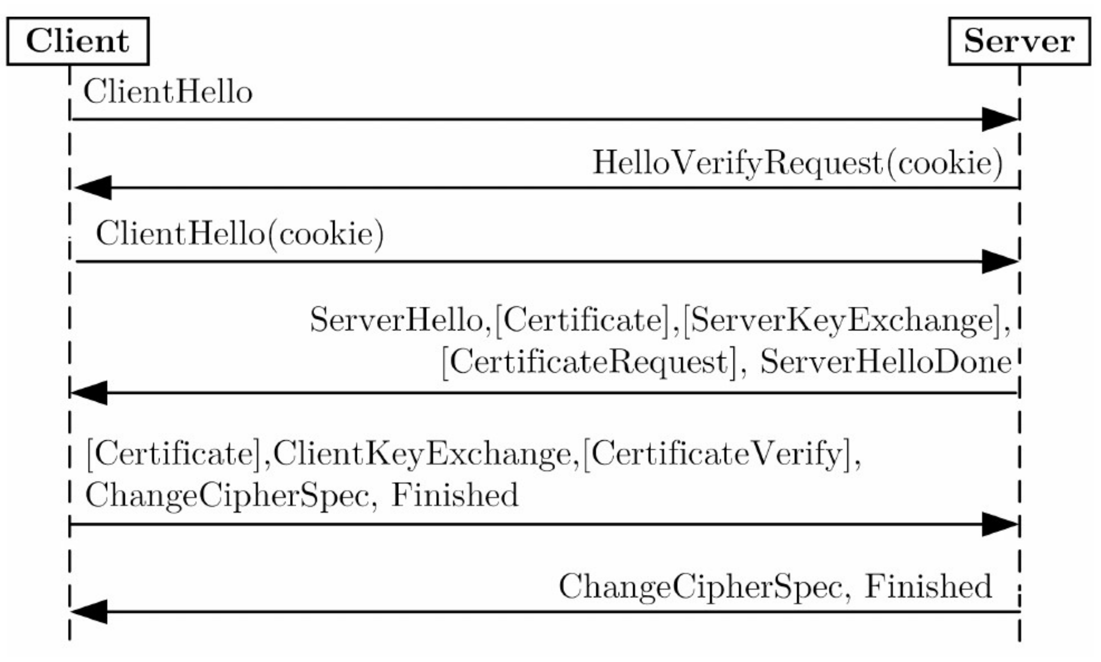

Client
Server
ClientHello
HelloVerifyRequest(cookie)
ClientHello(cookie)
ServerHello,[Certificate],[ServerKeyExchange],
[CertificateRequest], ServerHelloDone
[Certificate],ClientKeyExchange,[CertificateVerify],
ChangeCipherSpec, Finished
ChangeCipherSpec, Finished

Figure 2.3. DTLS Handshake exchange

The Handshake process is similar to this process in TLS [40] but DTLS introduces several features to adapt for running over an unreliable transport protocol. Firstly, a process of cookie exchange is added to stand against Denial-of-Service attacks. Secondly, DTLS adds modifications to the handshake header to handle message loss, reordering, and message fragmentation. It is supported by a simple retransmission mechanism. The main header modification is that each handshake message is assigned with a specific sequence number within that handshake. As a result, the messages can be received out of order, queued and processed when it is time for that. If a message with an expected sequence number is not received during timeout, the retransmission takes place. The results of the Handshake are security parameters including a fresh master secret which is used further to generate a set of keying material, utilizing an agreed Pseudo Random Function (PRF). DTLS reuses a PRF defined in TLS which is based on a construction known as Keyed-Hashed Message Authentication Code (HMAC) [41]. This construction is a current standard for performing keyed hashing which means applying a hash function to a text with use of a secret key. Different hash functions may be used as a basis for HMAC, however SHA256 [42] is preferable.

There are six writing and reading parameters generated from the master secret:

{ *client write MAC key*
server write MAC key
client write encryption key
server write encryption key
client write IV
server write IV }

The set of client parameters is used for writing operations by the client and for reading operations by the server and vice versa. For the moment, the client write IV and server write IV are only generated for implicit nonce techniques in authenticated ciphers.

The Record protocol protects application data by using the session keys generated during the Handshake. All DTLS data is carried in records which can be processed only when an entire record is available. In order to avoid dealing with fragmentation, DTLS requires records to fit within a single datagram. Figure 2.4 presents the DTLS record format.

Content type	Version	Epoch	Sequence number	Length	Ciphertext	MAC
1 Byte	2 Bytes	2 Bytes	6 Bytes	2 Bytes		

Figure 2.4. DTLS record format

The *epoch number* is used by endpoints to distinguish between sessions with different cipher states. The concatenation of the *epoch number* and the *sequence number* in DTLS perform identical functions as the sequence number in TLS which is retrieved from a transport layer header. However, unlike in TLS, epoch and sequence numbers are carried explicitly in the DTLS header in order to provide reliability in case of packet loss, reordering or repetition. The concatenation of the numbers can be used for replay protection, in MAC computation and in counter mode ciphers. First of all, the replay protection is achieved by remembering the epoch and sequence numbers of received records. If a message being processed has the concatenation of the numbers that has appeared before, this message is discarded. The concatenation is also included in a part of the record that is being authenticated to achieve integrity assurance. In the counter mode ciphers, the epoch number and the sequence number are concatenated with an explicit server or client IV from the keying material to produce a nonce to be used as an initialization vector in a block cipher.

Encryption process in DTLS follows a principle MAC-then-encrypt. The steps for message encryption can be defined as follows:

1. HMAC is computed over data and header
   $$\text{Tag} = \text{HMAC}(k_{mac}, \text{Header} + \text{Data})$$

2. Concatenation of Header, Data and Tag is padded
   $$\text{Padded} = \text{Padding}\ (\text{Header} + \text{Data} + \text{Tag})$$

3. The padded data is now encrypted using an encryption key and randomly chosen for this very packet IV
   $$\text{Encrypted} = \text{Encrypt}\ ([k_{enc}, \text{new random IV}], \text{Padded})$$

4. The header is then preprended to the ciphertext and the resultant message is sent to a transport layer.

When an endpoint receives a packet a datagram, it decrypts the data, checks padding format and verifies a keyed hash. It is important to note that stream ciphers are banned in DTLS in order to achieve independence of record's encryption/decryption.

2.4 DTLS-based Multicast Security

The discussed above standard version of DTLS protocol is suitable only for unicast communication. However, the need of multicast security in constrained environments led to development of its version adapted for multicast group communication. The IETF DTLS In Constrained Environments (DICE) Working group proposed such a solution [20]. The resultant protocol fulfils the security requirements of data confidentiality, replay protection, group authentication and integrity for a necessary group size and multicast communication topology.

It is proposed in [20] that a DTLS group session is established without preforming a regular handshake process. Instead, a *group controller device* distributes a Group Security Association (GSA) [43] consisting of security parameters among multicast group members. The security parameters include keying material, used cipher suite, a MAC algorithm a compression algorithm, whether the connection end considered to be a client or a server, and other optional values. As in a traditional DTLS session, the current write and read states of the GSA consist of six keying elements that can be generated from the *master key*: *client write MAC key, server write*

MAC key, client write encryption key, server write encryption key, client write IV, server write IV.

The distribution can be done using standardized key management schemes for constrained networks such as *GSAKMP* [44]. Thus, all group members share the same group security material so sending devices can address multicast messages to listening devices. It should be noted that under such conditions, it is not possible to assure the source authenticity of multicast messages sent within the group.

The format of a traditional DTLS record involves *epoch* and *sequence number* fields which ensure freshness and provide detection of message replay. The *epoch* is fixed by DTLS handshake for a given session and the *sequence number* is initialized to 0 and incremented by one for every new record sent by a sender. In the scenario where multiple senders present and share the same GSA, they may have the same *sequence numbers* for their multicast messages. As a result, the messages may get discarded by listening nodes as replayed packets. Furthermore, DTLS defines for usage authenticated encryption (AEAD) cipher suites as AES-CCM [45] and AES-GCM [46] which employ nonces to achieve security of the ciphers. This security can be completely broken in a case of nonce reuse under a single key which may take place in the multiple senders' scenario because all senders apply the same key to all outgoing multicast messages within the group. In these ciphers, a single key is used for encryption and authentication. In AES-CCM, encryption is provided with Advanced Encryption Standard (AES) [47] in Counter Mode and authentication is ensured with Cipher Block Chaining (CBC) MAC. According to [45], a CCMNonce is a combination of a salt value (the client write IV or the server write IV from the GSA) and an explicit nonce value which is the *epoch* concatenated with the *sequence number*. It can be seen that the scenario may indeed encounter nonce reuse by different senders that will lead to a security breach.

The possible solution to the discussed challenges could be synchronization of all group senders to avoid reuse of sequence numbers. However, this is a hard task and, especially, in constrained networks. The second approach being adopted is to embed a unique sender identifier in the *sequence number*, thereby providing each sender with a separate non-overlapping numbering space. This is done by a group controller who assigns a unique *SenderID* to each sending device in the group. All group members obtain the list of SenderIDs and are further able to sort transmitted multicast messages. The resultant multicast DTLS frame format is presented in Figure 2.5.

Content type	Version	Epoch	Sender ID	Trunc_ seq_number	Length	Ciphertext	MAC
1 Byte	2 Bytes	2 Bytes	1 Byte	5 Bytes	2 Bytes		

Figure 2.5. Multicast DTLS record format

2.5 Protection of Unicast Responses to Multicast Messages

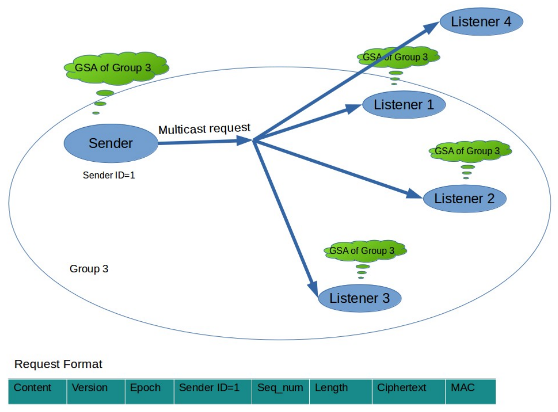
The specification by the Dice Working group claims [20] that individual unicast response messages must be secured and suggests protecting them by establishing a separate DTLS session for each connection multicast sender-listener. This has several disadvantages. Firstly, either the sender or the listener would have to initiate DTLS handshake; both would have to perform the handshake and spend corresponding resources to actually protect the data belonging to another session. Furthermore, the sender would have to perform amount of the handshakes equal to the number of members of the multicast group. According to a standard for group communication in LLNs [48], the number of multicast group members communicating securely may be up to one hundred, so, if the multicast sender is a constrained device and has to perform the handshake with every group member, it may completely deplete its resources and totally block the communication. In addition, the sender is not required to know all multicast listeners in the group in advance because it is the task of the group controller. This means that the sender cannot perform the handshakes in advance, and it is able to perform them only when the listeners request that which is likely to happen in a very short period of time after obtaining the multicast requests. However, according to DTLS specifications, it is a client who must initiate a handshake, not a server. It leads to a problem with communication roles when a CoAP client may happen to be a DTLS server and vice versa.

To resolve the aforementioned issues, Tiloca proposes [21] an extension to the scheme presented in Section 2.4. The goal of the extension is to reuse the session parameters and group keying material to secure the response messages. The approach is as follows. Upon creating the multicast group, the group controller assigns a *GroupID* to it and provides the value to all group members. The *GroupID* is to be included in the DTLS header of the unicast response messages instead of *SenderID*, so in the resulting record format it precedes the *truncated sequence number.* The format of a unicast response message is presented in Figure 2.6.

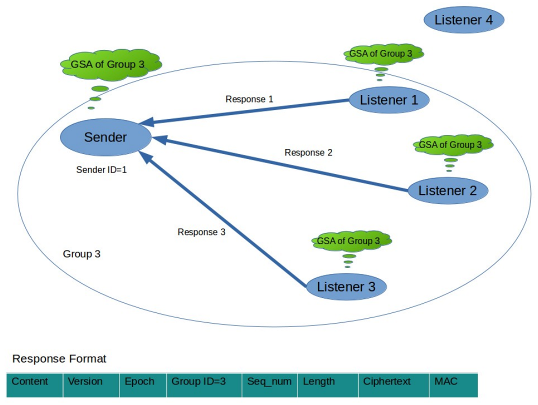
Content type	Version	Epoch	Group ID	Trunc_ seq_number	Length	Ciphertext	MAC
1 Byte	2 Bytes	2 Bytes	1 Byte	5 Bytes	2 Bytes		

Figure 2.6. Format of unicast response DTLS record to multicast request

The listeners maintain a table where each *GroupID* is associated with a corresponding ID of the sender in the multicast group. If a listener receives a protected multicast request and intends to send back a protected unicast response, the listener retrieves a *GroupID* associated with a *SenderID* from the request and prepares a response of a format presented in Figure 2.6, using the *Server write encryption and MAC keys*, and possibly the *Server write IV.* Upon receiving the unicast packet, the multicast sender firstly checks whether there is an established DTLS unicast session with the given node on a local port where the packet has been received. If there is such a session, the packet is processed according to the traditional DTLS; otherwise, the multicast sender assumes that this is a unicast response to the previously sent multicast request and processes the packet as a record of the format in Figure 2.6. The sender retrieves the *GroupID* value and looks up in its connection table for the corresponding session and *server keys*. If the connection is found, the record is decrypted, the MAC is verified and the *sequence number* is checked to detect replays. Figure 2.7 illustrates a possible scenario with one group sender and three group listeners.



(a)



(b)

Figure 2.7. Scenario of group communication with a multicast request (a) and unicast responses (b)

In the scenario, a group consists of one sender and three listeners who all share security parameters as a part of GSA. Listener 4 also subscribes to the multicast address so it can receive requests but it does not have proper keying material and, therefore, cannot decrypt the requests. Request and response formats only differ in the ID field where the sender includes a senderID, whereas the listeners include a groupID.

2.6 Related Work

In this project, we use DTLS to provide communication security in the IoT and build the solution for multicast communication security based on that. Nevertheless, other approaches for end-to-end communication security exist one of which is discussed in this section. We also explain why the solution based on DTLS is better under conditions of this project.

2.6.1 The Internet Protocol Security (IPsec)

The Internet Protocol Security (IPsec) [49] is a widely-used security architecture to provide peer-to-peer secure channel across the Internet. It rather consists of three independent protocols. The IP Encapsulating Security Payload (ESP) [50] provides confidentiality, origin authentication, and connectionless integrity of an IP packet payload. The IP Authentication Header (AH) [51] is used to provide authentication and integrity of the payload and the packet header. Both protocols also offer protection against replay attacks and can be used in different modes, with some features possibly inactive. Moreover, the protocols can even be combined together for authentication of the header and the payload and encryption of the payload, but this approach is rarely used due to implementation complexity and high overhead. The third protocol, the Internet Key Exchange (IKEv2) [52], is responsible for performing mutual authentication and establishing keying material and other security parameters. These parameters are collectively referred to as Security Association (SA). IPsec can be implemented in a host, more often integrated with an Operating System (OS), or in a router [53]. Operating on top of the network layer, IPsec may ensure confidentiality and integrity of transport layer headers, which cannot be done with protocols operating on top of the transport layer such as DTLS.

What is important for relation to our work, IPsec has been extended for use in IP multicast [54]. Particularly, the IPsec encapsulation of IP multicast must specify an IP multicast address as a destination address, and both the destination and origin addresses must be indicated explicitly for correct routing. The SAs of the multicast type are introduced as well as the entity of Group Security Police Database (GSPD) to support both unicast and new multicast SAs. Finally, authentication and integrity on a group level are provided with a shared authentication key.

Despite many advantages, IPsec has also several drawbacks, especially for use in the IoT. Firstly, it adds a significant overhead to IP packets, whereas the IoT imposes a strict limit on the packet size. The overhead mainly comes from the fact that additional headers have to included in every datagram for both IPsec's ESP or AH. In order to alleviate this issue, [55] proposes a solution for integration IPsec with 6LoWPAN, meaning possible compression of IPsec header by techniques of 6LoWPAN. In the work, implementation was presented and significant decrease of packet overhead was demonstrated; however, energy consumption and processing time still stayed considerably high.
Furthermore, IPsec only offers secure channel to a machine or a device, while DTLS allows more precise access control by allocating a tunnel to a specific application. Another aspect is related to domain names security (DNS) [56]. DTLS allows using a DNS name to securely identify a server which is not supported in most implementations of IPsec. This is important because an IP address is rarely known in advance so DNS has to be involved in authentication. Finally, the Internet Engineering Task Force (IETF) strongly encourages the use of DTLS in their recent proposals. This may help to unify security architectures in the IoT to DTLS and, thereby, avoid possible problems with interoperability.

Chapter 3

Design of Response Protection Mechanism

In this chapter we analyse an existing mechanism for response protection in group communication, propose improvements to it and discuss why the proposed solution is secure. In the analysis, we explain underlying reasons of discovered vulnerabilities and demonstrate possible attacks and corresponding scenarios. Thereafter, we propose improvements that fix the vulnerabilities and justify our choices. Finally, remaining security considerations are discussed in the last section.

3.1 Analysis of Existing Mechanism

The mechanism proposed in [21] aims at reuse of material from a Group Security Association (GSA) for protection of unicast responses to multicast requests. Thus, the establishment of a separate point-to-point DTLS session can be avoided. The mechanism introduces a *group ID* which is to be assigned to every GSA and proposes to include this ID as a new field into a DTLS header. If a multicast sender receives an unicast message from a node with which it does not keep a point-to-point DTLS session, it assumes that there can be a group DTLS session and, subsequently, retrieves a byte corresponding to the group ID field. The multicast sender checks if there is a GSA with a group ID equal to the retrieved number. In case if there is such a GSA, the sender uses group material to decrypt and verify the message.

The described approach mostly focuses on achieving high communication efficiency and does it well by avoiding establishment of a new secure session since as we show in next chapter, the handshake is the most consuming process in the DTLS protocol. However, we have discovered that the approach omits certain security considerations and, therefore, appears to be vulnerable to specific malicious attacks. Further, we describe underlying reasons and the actual attacks that can be successfully performed.

3.1.1 Discovered Security Weaknesses and Underlying Conditions

As we describe in the Section 2.3, key material in DTLS consists of six elements derived from a master secret. It is *Client Write MAC Key*, *Server Write MAC Key*, *Client Write Encryption Key*, *Server Write Encryption Key*, *Client Write IV* and *Server Write IV*. All the elements are distinct and a possible adversary cannot benefit from the fact that they come from the same origin.

In the group scenario of DTLS-based secure communication proposed by [20], [21], ALL senders and listeners (or clients and servers in other terminology) use the same client and server parameters for performing the writing and reading operations. According to [21], to protect response messages the listeners include a group ID in a record header to be distinguished among groups by the sender. In contrast to multicast senders who all hold unique sender IDs and include them in the record header, the listeners hold the same number as a group ID because there is only one identifier for the group. As it is shown in Figure 2.6, the DTLS record header for unicast responses to multicast requests include Content, Protocol Version, Epoch, GroupID, Truncated sequence number, and Data Length fields. Obviously, Content, Version and GroupID fields are likely to be identical for responses from all listeners of the same group. Content is Application data, Version can be 1.2 (for DTLS). At the same time, the epoch is incremented for every new cipher state and the truncated sequence number is incremented for every record sent. However, both the epoch and the truncated sequence number are set to zero at the beginning of group communication so all group listeners start with the same values and, moreover, increment it in a uniform way. As a result, the listeners may likely end up with identical DTLS record headers (not considering Length because a client does not have any prior information about it and accepts any Length value). Furthermore, sharing of the same keys and explicit IVs by the listeners may result in reuse of {key, nonce} in authenticated encryption systems. All this appears to be weaknesses leading to feasibility of the attacks described below.

3.1.1.1 Replay Attack

When a message is to be transmitted using DTLS, it is fragmented, optionally compressed, hashed with a key (HMAC) and finally encrypted. The operations are done in this exact order. The keying material is started to be applied from the process of HMAC calculation. The HMAC is generated as follows:

HMAC (Server MAC write key, epoch+groupID+trunc_seq_number+
Content type+
Protocol version +
Data length +
Data),

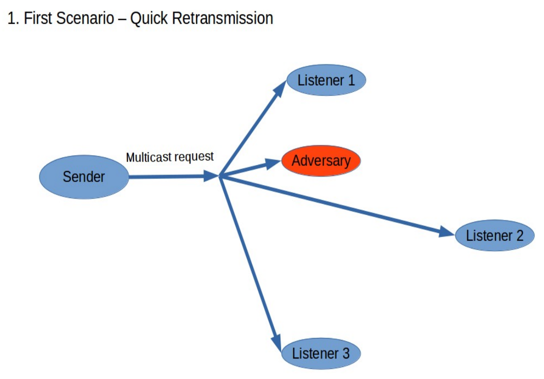
where values from *epoch* to *Data length* constitute a record header.

As we discuss above, the listeners share the same *Server MAC write key* and are likely to end up using the same epoch and sequence numbers. Afterwards, the listeners encrypt concatenation of data, HMAC and padding again using identical *Server write key* and freshly generated random IV:

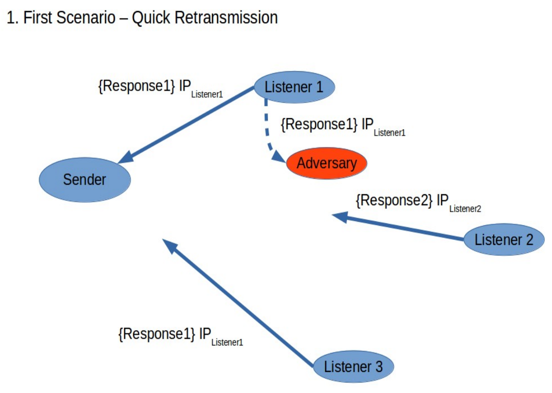
Encrypt([Server write key, random IV], Header + Data + HMAC)

Considering the identical values, what will happen even if Data and Data Length of responses from two listeners differ? The answer is that a DTLS record from one listener can be equally accepted as a valid DTLS record from another listener! Basically, a client identifies a server by a source IP address and port number of an incoming packet. However, neither the IP address nor the port number are protected by a DTLS header because DTLS operates higher at the protocol stack. Furthermore, the DTLS header does not include any distinguishes between members of the same group, allowing sequence numbers to overlap. As a result, the client will not be able to say if this DTLS record is indeed coming from this exact listener.

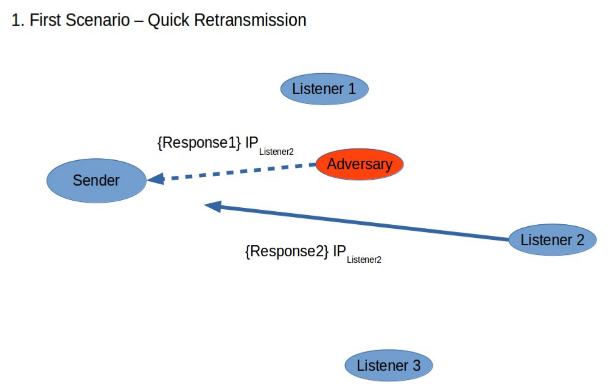
Let us consider a scenario with presence of an active attacker who is able to intercept packets, substitute unprotected data at transport, network and data link layers, and resend the resultant packets. We give an example of a group of one client (multicast sender) and three servers (listeners) who communicate securely using the discussed above approach for response protection. If some group listeners have started operating at the same moment, they will likely to hold the same values for the sequence numbers since multicast requests are every time received by all nodes listening to the multicast address. If the group listeners have started their sequencing from different values, the attacker can store responses of one node and use them when it is the same sequence number on another node. The attacker then may intercept a response message from one node and, spoofing an IP address, resend this message as an answer from another node. The message will be accepted by the client as valid due to the described above reasons. This is known as a **replay attack** performed on top of IP spoofing. But how does the attacker perform this malicious activity? There are at least two possible sub-scenarios for the attack which are illustrated in Figure 3.1 and Figure 3.2.



(a)



(b)

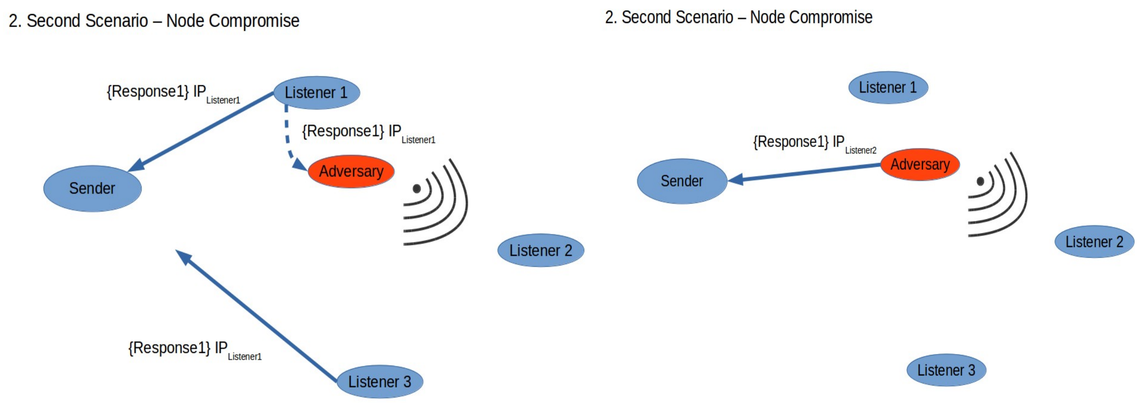


(c)

Figure 3.1. Replay attack on response protection mechanism with quick retransmission: (a) sender transmits multicast request; (b) listeners send replies, adversary intercepts reply of listener 1; (c) the adversary retransmits the reply with spoofed IP address of listener 2

In the first sub-scenario shown in Figure 3.1, a sender transmits a multicast request to group members listening on a specified multicast address. An attacker receives this request along with the group members because he is also listening on the multicast address. This way he can know that the group members are going to send responses soon. Let us assume that Listener 1 replies first due to closer distance to the sender or smaller response delay in case of presence of random replying delay that listeners undergo. The attacker intercepts this response from Listener 1 (radio transmission is easy to intercept) and resends it quickly with an IP address of Listener 2 who has not replied yet or its response is still on the way. If the attacker is sufficiently quick to surpass a message from Listener 2, his spoofed and replayed message will be accepted as a response from this listener. The multicast sender will also receive the true response but will discard it because it has lower sequence number than the response that has just been accepted and, therefore, is a stale message.

The second sub-scenario shown in Figure 3.2 assumes that the attacker has ability to perform jamming on a specific node, blocking node's reception and transmission functionality on a radio level, or physically compromise the node and block its functionality by, for example, shielding it with a material that is not transparent for radio waves. In this case the attacker does not even need to be quick at resending the response because there will be no answer from the legitimate node at all. If the adversary blocks both receiving and transmission functionality of Listener 2 completely, disruptive effect of the attack will last even when it is stopped. Listener 2 has not been aware of the multicast requests, it has not tried to send responses and, therefore, the sequence number has not been incremented, whereas the multicast sender has been receiving valid responses with an IP address of this listener and has incremented the counter. As a result, when the listener starts sending responses after the blocking is ceased, its sequence numbers will be stale, thus the responses will be discarded. The situation will take place for the same amount of fresh responses as the amount of responses that have been forged by the attacker until the incrementing counter of the listener will become equal to the counter of the sender.



(a) (b)

Figure 3.2. Replay attack on response protection mechanism with jamming or physical node compromise: (a) adversary intercepts a response from listener 1 and at the same time performs jamming on listener 2; (b) the adversary retransmits the reply with spoofed IP address of listener 2

To estimate potential harm that this attack can cause, let us consider an example of a simple security system where the sender is a central control node and the listeners are motion sensors installed along the area of some object. The sender periodically requesting if there is any movement near the sensors. The listeners reply with a simple “Yes” or “No”. A trespasser wants to enter the object close to one of the sensors. Then he retransmits the “No” reply from some other sensor to the requests as a reply from one that he is passing. As a result, the control node thinks that compromised node is still in operation and there has been no movements near the sensors.

3.1.1.2 Nonce Reuse in Authenticated Encryption

DTLS offers a possibility to use Authentication Encryption with Associated Data (AEAD) cipher suites like AES-CCM and AES-GCM which provide authenticated encryption. They provide AES encryption in a counter mode with CBC-MAC and Galois MAC respectively. Both AES-CCM and AES-GCM take a nonce value as an input in order to randomize encryption but only a part of this nonce is random because it is used as a counter in encryption. According to Section 3 in specification of AES-CCM [45], a nonce called CCMNonce to be used in AES-CCM and AES-GCM is constructed as a combination of a salt value and a sequence number.

```
struct {
        opaque salt[4];
        opaque nonce_explicit[8];
} CCMNonce;
```

The 4-bytes salt value is Client Write IV or Server Write IV stored in the GSA. The 8-bytes nonce_explicit is constructed as *epoch+sequence_number* which is translated into *epoch+group_id+truncated_seq_number* for the servers in the case of the proposed response protection mechanism. Upon analysing the resultant Server CCMNonce, it can be seen that nonces are likely to be identical for different servers in the group! The servers share the same Server Write IV from the group security parameters and nonce_explicit can happen to be the same due to identical pace of incrementing epoch and truncated sequence numbers discussed in previous Subsection 3.1.1.

The main feature of authenticated encryption is that one key is used for both authentication and encryption of data. Thus, only Client write key and Server write key are used but not the MAC ones. [45] requires that a distinct pair key/IV MUST be used only once, otherwise security of the cipher may be completely broken.

Both AES-CCM and AES-GCM encrypt and decrypt data using a counter mode (CTR). In order to understand what effect nonce reuse may have, let us demonstrate how encryption in the counter mode works:

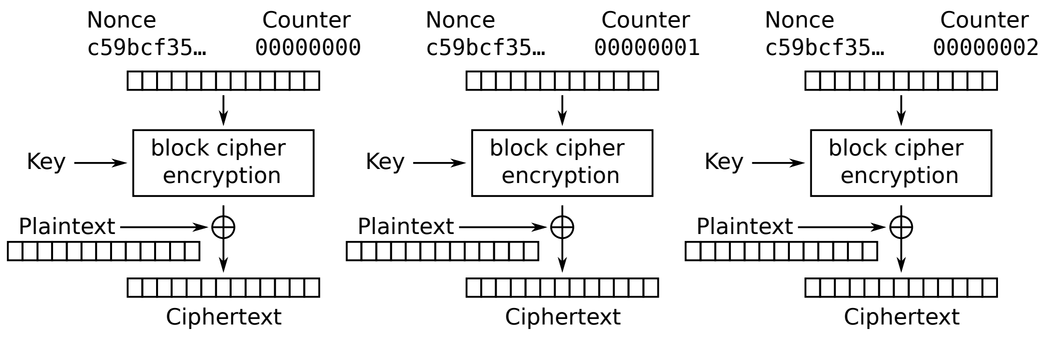


Figure 3.3. Counter (CTR) mode encryption

It can be seen that a block of ciphertext is basically obtained as

Ciphertext = Plaintext XOR E(Key, Counter)

This means that if two messages have been encrypted using the same key and counter, the XOR of their ciphertexts will be the XOR of the original plaintexts! So an adversary may be able to completely recover original data by having only two ciphertexts. But even this is not the end. Security study [57] of AES-GCM revealed that the nonce reuse caused even worse problems. It allows the attacker to solve underlying keyed hash, making subsequent forgeries trivial and also enabling the attacker to choose IVs who will be able to produce colliding counters.

3.2 Design of Improved Approach

In order to prevent the attacks described in the previous section, an issue with reuse of the same {key, nonce} pairs by the group listeners has to be resolved. Simultaneously, taking into account all the advantages of reuse of group key material, we would like to follow this strategy and avoid establishment of separate point-to-point DTLS sessions. We also want to propose an approach which would maximally reuse functionalities provided by the existing traditional DTLS protocol and the DTLS-based multicast protocol so unnecessary inventions are avoided and smooth integration can be achieved. Therefore, we propose to derive individual Server write key for each group listener in such a way that a group client is also able to derive these keys for each listener. The idea is to apply HMAC for derivation using a part of group keying material as a secret, and IP address and port number of a listener as plaintext to be hashed. The IP address and the port number are publicly known and, therefore, easily accessible by the client characteristics that are distinct for every group member. We outline the derivation algorithm in following Subsection 3.2.1.

3.2.1 Algorithm for Derivation of Individual Server Keys

In a communication group with presence of a client and an unknown number of listeners the algorithm for derivation of individual server keys is as follows:

1) Before sending a response to the first multicast request, every listener constructs a byte-string consisting of concatenation of its IPv6 unicast address and its port number used for secure communication within the group. The string has a size of 16 + 2 = 18 bytes (144 bits). Only 2 least significant bytes of the port number are taken and leading zeros are discarded if the port number is stored as a 4-byte value.
2) The 32-bytes (256-bits) string Client_write_key+Server_write_key from the group keying material is prepared to be used as a secret.
3) Then HMAC in conjunction with SHA-256 is used as a Pseudo-Random Function (PRF) to generate a new individual secret string. Since SHA-256 is underlying hash function, the length of the output string is 256 bits (32 bytes).
4) The first 16 bytes of the output string are saved as the Server MAC write key and the last 16 bytes as the Server write key. If only authenticated encryption is used in the group and nodes do not store MAC write keys, the last 16 bytes are saved as the Server write key and the first 16 bytes are discarded.
5) When a group client receives the first response message from a listener, the client identifies which group the listener belongs to by the group_ID field, derives listener's IP address and port number from the IP and UDP headers and computes the individual server keys following the same algorithm. After the individual key is derived, the client processes the received response.

3.2.2 Choice Argumentation

The HMAC with SHA-256 is chosen for our key derivation algorithm because it is an option used by current versions of TLS and DTLS for both key expansion and MAC computation for all cipher suites. The good thing about HMAC is that its security strength completely relies on cryptographic properties of an underlying hash function and a key used. SHA-256 is considered to be cryptographically strong during next decade [42]. Moreover, the concept of HMAC with a cryptographically strong hash function implies that knowledge of a plaintext without knowledge of a key will not give an adversary any information about the produced hash output. Upon condition of proper choice of a key used, it will assure security strength of the hashing part of the proposed algorithm.

HMAC specification [41] strongly discourages use of keys that are shorter than output length of an underlying hash function because it may significantly decrease security strength but also mentions that keys longer than the output length do not significantly increase the strength. Hence, the concatenation of the Client write key and the Server write key is used as a key in the HMAC. These keys are obtained from the master secret using the PRF defined in Section 5 of [17]. This means that they satisfy all randomness requirements. Furthermore, the concatenation of the Client write key and the Server write key rather than the Server MAC write key and the Server write key is used because some implementations support only authenticated encryption, especially in restricted environment, to save resources and therefore do not even store the MAC write keys. In order to provide consistency and avoid forcing implementations to store the MAC keys when it is not needed, the first key option has been chosen.

The concatenation of an unicast IPv6 address and a port number is used as the input string because it is unique for every node and also is included in lower level packet headers. The latter fact allows the group manager to avoid distribution of any additional data and a client does not have to store information in advance. The client receives the first message, computes the corresponding key once and then simply add the listener to its list to keep track of sequence numbers.

Lastly, note that an idea of deriving a separate Server write IV instead of a write key was also considered but rejected due to a small size of the IV material and a fact that it would solve only the problem with authenticated encryption. The distributed IV is only 32 bits and in case of a big number of listeners in the group, it could have led to possible identical hash outputs for different nodes due to the birthday paradox. To be more precise, 93 listeners in the group give a probability of a key collision equals to one millionth [58] which is a non-negligible number considering a possibly high frequency of rekeying.

3.3 Security Considerations

Several security issues related to the novel approach that should be taken into account are discussed below.

3.3.1 DoS Prevention

There are two possibilities of multicast sender getting aware of all listeners present in a group. First, the multicast sender obtains information about all group members present from a group controller in advance so it has a list with IP addresses and port numbers used for the communication of all group members. Second, the sender considers every response from an unknown address which has a DTLS header suitable for group communication as the first message from a listener. In the first case if an adversary tries to send a response message with its own IP address, the message will be discarded due to address mismatch but even if the message is with a spoofed IP address, it will be encrypted and/or MACed using an invalid key and therefore also discarded by the sender. In the second case if the adversary uses its own address, the client will have to perform the hash computation before checking and discovering that the message is not valid. In conclusion, the adversary is unable to forge messages or deceive the group members in any of the cases but to prevent a possible vulnerability of a DoS attack, it is preferable to distribute information about present group members in advance.

3.3.2 Group level confidentiality

Even though the proposed approach assumes different write keys for every server in a group, still only group confidentiality is provided. This means that all group members are trusted within the group but also that any group member can retrieve or compute a key of any other group member and even forge a message if the member is compromised and it knows addresses of other members. However, this form of confidentiality is sufficient for most IoT scenarios where the communication is used to send commands and especially in low-power networks where efficiency is one of the prior concerns.

3.3.3 Uniqueness of Group IDs

The group controller is responsible for assignment of group IDs. Although it is important to assure uniqueness of the group IDs, it is not that crucial as uniqueness of sender Ids. A duplicated group ID will probably harm communication for one of the affected groups because the sender will always stop searching when it finds the first group in the list with a given ID but the sender still might be able to distinguish among the two groups by a port number. The communication with different groups is supposed to be performed by different applications on a client and, therefore, using different ports. The most important note is that the repeated ID will not ruin security because all members in the both groups will have individual, distinct within the network keys.

Chapter 4

Implementation, Experimental Setup and Evaluation

In Chapter 4 we describe functionalities that we have implemented, experimental scenarios and their realization in client and server applications, experimental evaluation on hardware and discuss results of the evaluation. For the implementation we indicate what functionality was available before the start, what steps have been undertaken to enrich the functionality and what it becomes in the end. We describe what problems have been solved to support necessary communication and how the experimental setup is arranged. Finally, we analyse and discuss results of memory requirements, communication performance and energy consumption evaluation that have been obtained in the experiments.

4.1 Implementation

We implement our library for secure group communication and subsequent prototypes on top of Contiki OS v2.7 [22] which is an open source operating system for the Internet of Things providing Internet communication for low-power wireless devices. The Contiki OS is supplied with an entire development environment called Instant Contiki. It includes different development tools like the Cooja network simulator [59] and Erbium REST Engine and CoAP implementation [60] which we use as a CoAP communication library. The starting point for implementation is a publicly available TinyDTLS 0.5 library [24] by Olaf Bergmann. It is a lightweigth DTLS implementation for constrained environments providing secure unicast communication. The TinyDTLS enables two nodes running the DTLS application perform the handshake and afterwards communicate securely with each other using negotiated security parameters. The ciphersuites supported by the implementation are TLS_PSK_WITH_AES_128_CCM_8 and TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8 where the first ciphersuite provides authenticated encryption based on AES in CBC mode and the second ciphersuite provides encryption based on elliptic curves. The first ciphersuite requires pre-shared keys to perform the handshake and derive session keying material. It must be admitted that the implementation of the latter ciphersuite is still under development and optional, therefore, we mainly focus on the first one and do not include support of the second one in compiled files to save memory. Finally, the library is included into Contiki OS as an application and the considered version is present in the Instant Contiki environment.

In order implement multicast communication support described in [20], we first of all have created a GSA structure that group nodes use to store a multicast address assigned for group communication, security parameters used, and epoch and sequence number counters. Moreover, group listeners store a list of all group senders and keep track of epochs and sequence numbers separately for each of them. The group senders also store their sender IDs that are assigned to them by a group controller, and listeners identify the senders in the list by the ID. Then we have implemented all functions to create the GSA structure and add parameters to it including addition of senders in the corresponding list. Finally, we have implemented additional functions for encryption and decryption procedures that use security data from the GSA and are able to process the new type of the DTLS header for multicast packets.

Next, we have implemented response protection support proposed by [21]. A group ID is included in the GSA structure. The group senders store a list of the group listeners for the same purposes as the list of the senders stored by the listeners. The DTLS group packet header is reused for the response messages with a substitution of the group ID for the sender ID. If a listener needs to send a response but there is no unicast secure session a requesting multicast sender, the listener uses security parameters from a GSA related to a multicast address at which the listener has received the request. If the sender receives a secure unicast message but there is no unicast DTLS session established with the message origin, the sender tries to treat the packet as it has been encrypted using group keying material.

Finally, we implement the improvements to the response protection mechanism proposed in this thesis. The "key_derivation.c" file includes functions to perform keyed hashing of server-related parts of the key block. The functions are invoked from the application directly and process data from a supplied GSA. The DTLS client then store an individual key block for every listener in the list, whereas the DTLS servers overwrite their key block with new values.

It should be mentioned that during our work on the implementation we have discovered and fixed a bug in the version of TinyDTLS provided in the Instant Contiki. The bug was located in a check_server_hello function used on the client side and was inducing permanent failure of the handshake process. When elliptic curves support was not used and thereby "ECC" pre-compiler condition was set to 0, the check_server_hello function never returned a non-negative value which resulted in the handshake failure. To fix the bug the following code, checking that there are no extensions in the header, should be added as a last check to the function:

```
if (data_length < sizeof(uint16)) {
   if (handshake->cipher == TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8)
      {
            goto error;
      }
    return 0;
   }
```

4.1.1 CoAP Adaptation

We introduce a random server response delay as suggested by [12] to avoid collisions between responses from different servers. When a server receives a request, it picks a random point during so called *leisure time* period and waits its expiration before sending the response. A rough lower bound for the leisure time value may be calculated using a group size estimate, a target data transfer rate, and an estimated response size. We specifically enrich a CoAP library in Contiki with the implementation of the leisure time for group communication.

4.2 Experimental Setup

The experimental setup consists of four constrained nodes which communicate using 2.4GHz IEEE 802.15.4, 6LoWPAN, CoAP and DTLS in their protocol stack. The constrained nodes are CC2538 evaluation modules [61] used with SmartRF06 evaluation boards [62]. The CC2538 evaluation module is based on CC2538SF53 wireless microcontroller system-on-chip [63] which incorporates ARM®Cortex®-M3 processor, 512 KB of flash memory and 32 KB RAM. The assembled constrained node is presented in Figure 4.1.

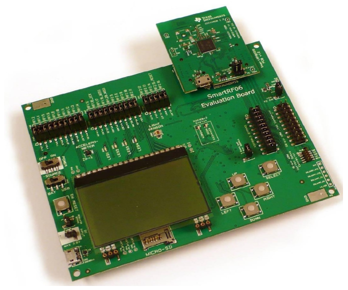


Figure 4.1. CC2538EM with SmartRF06

The experimental setup with one client and three listeners is shown in Figure 4.2. Beside the constrained nodes, we also use CC2538 USB dongle to sniff packets on the air. A python script [64] by Andrew Dodd is used to pipe intercepted packets to Wireshark network analyser [65]. Thus, we can validate our experimental results by examination of intercepted packets.

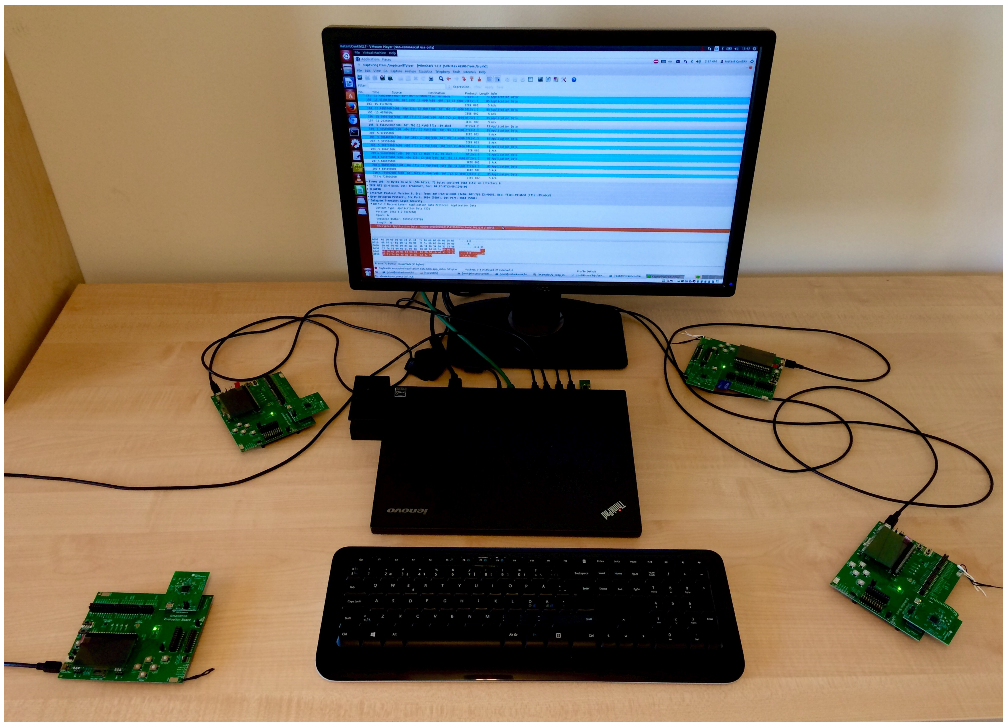


Figure 4.2. Experimental setup with one client and server applications being run on CC2538DK boards

In Contiki applications for this board should be compiled with a target cc2538dk. The command is:

```
make TARGET=cc2538dk
```

To flash the boards, the serial boot loader is used because there is no rule in Contiki for direct upload to cc2538dk platform. The flashing is performed using a python script written by Jelmer Tiete [66]. Two operations have to be performed to be able to use the script for flashing. Firstly, the 8-bit boot loader backdoor field in the CCA area in flash needs to be set to 0xF3FFFFFF. Secondly, a configuration of the cc2538dk platform in Contiki (file `contiki/platform/cc2538dk/startup-gcc.c`) has to be modified to keep the boot loader backdoor enabled after flashing. The boards can be flashed using the following command:

```
sudo python cc2538-bsl.py -e -w -v -p “interface” “application.bin”
```

4.2.1 Experimental Scenario

In our experimental scenario the constrained nodes compose a star topology. The communication is performed over a single hop only because there is no stable implementation of multicast routing protocols in Contiki at the moment and, therefore, the Instant Contiki environment does not include it. One central node which we further call *the client* acts as a CoAP client, a DTLS client and a group sender. Three other nodes which we further call *the servers* acts as CoAP servers, DTLS servers and group listeners. The scenario is shown in Figure 4.3. The scenario includes five experimental settings to obtain data for comparison of unicast-multicast and secure-insecure approaches. The settings are described in detail in next subsection.

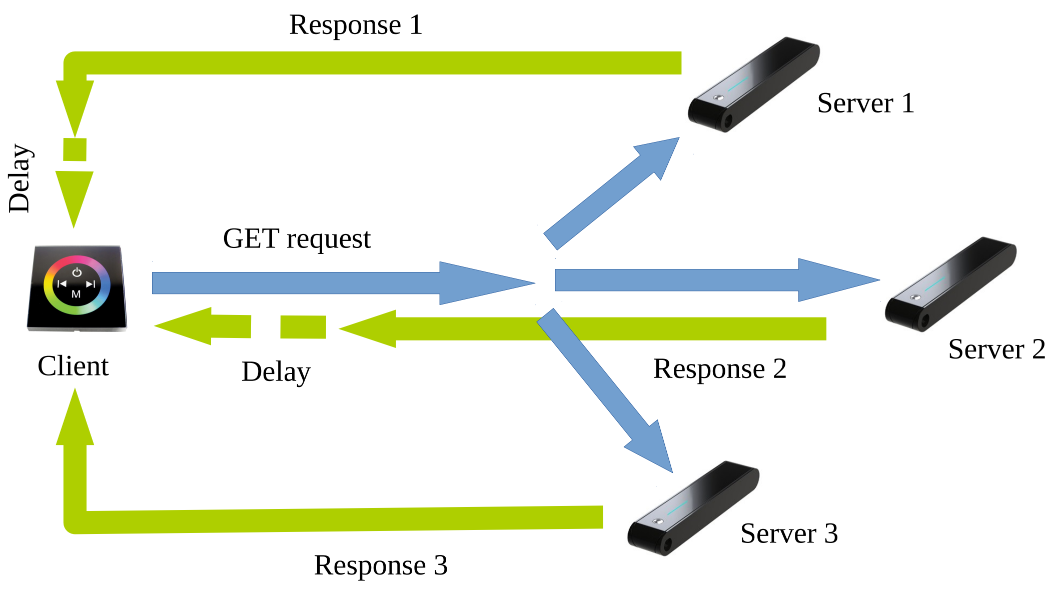


Figure 4.3. Experimental scenario

The client sends a CoAP "/hello" request of the GET type to the three servers through unicast or multicast communication. Unique 4-bytes token values are assigned by the sender to every group request and included in the CoAP header. The servers are supposed to process the request and answer with a response including a "Hello World!" string and a mirrored token. The scenario accomplishment differs depending on whether the GET request is sent as a unicast or as a multicast message. In the unicast case the client sends a Confirmable CoAP blocking request to a listener, waits for its reply and upon obtaining it moves to next one. The blocking request means that the client does not proceed to the request for next server until it receives the reply. In the multicast case, the client sends only one multicast CoAP request which is Non-Confirmable due to reasons discussed in Subsection 2.2.1. Moreover, having processed the multicast request, the servers undergo a random delay before sending the response as described in Subsection 4.1.1. The delay is configured to be in a range **from 0 to 0.25 second**. We have examined different values for the leisure time from one second and to a quarter a second and have stopped at the current value, having reached the condition when communication is efficient and collisions still do not occur. The client constantly waits for the responses and process them upon receiving. In both cases the client uses a token value in the received packet to match with an earlier request. Toggle time between requests done by the client is 10 seconds. The settings are aimed to have conditions which are as close as possible to real life applications, therefore the Confirmable requests are used in the unicast settings and the Non-Confirmable are in the multicast settings. Admittedly, the GET requests are specifically used to provide a possibility to evaluate the response protection mechanism.

4.2.2 Scenario Settings

We configure five settings under the described scenario to evaluate communication performance and characteristics in all of them. We preconfigure the applications with a destination IP address and a port number in all the settings. Moreover, since there is no router in our setup, we manually add entries into neighbour routing tables of all nodes using their IP and hardware addresses. If it is not done, the first messages are dropped because a node does not know a MAC address of a destination node so it has to drop the message and perform ICMP exchange instead.

The five configured settings are as follows:

1) CoAP Unicast (CoAP Uni)

The client subsequently transmits an insecure unicast blocking request message to every server and waits for a response. This means that the client sends the request to the first server and does not do anything until it gets the response. Then the client proceeds to next server. The client and the servers are aware of IP addresses, port numbers and MAC addresses of each other.

2) CoAP Multicast (CoAP Mul)

The servers subscribe for listening on a given multicast address from the beginning. The client transmits only one message per request with a multicast address as a destination and waits for responses. The multicast address is provided to the client. The entry in the neighbour routing table is not needed for the client because multicast packets are currently transmitted in Contiki using broadcast MAC address. The servers answer to a unicast address of the client which is provided in the application.

3) DTLS Unicast Request, Unicast Response (DTLS Uni-Uni)

In this setting we add a DTLS layer to CoAP unicast communication to make it secure. The client and the servers are provided with a pre-shared keys to be used in the handshake process. After booting and configuring, the client subsequently performs a separate handshake with every server. In order to avoid mixing handshake messages from all servers, we have set a time period of 3 seconds for the handshake with one node. Thus, the client performs the handshake with the first server; if it takes less than 3 seconds, waits for the time period to end and then moves to the handshake with next server.

When DTLS secure sessions are established with all the servers, the client starts transmitting unicast requests individually secured using parameters shared with each server as it is in the CoAP Unicast setting. The servers protect responses also using security parameters individually shared with the client and send them to the unicast address of the client.

4) DTLS Multicast Request, Unicast Response (DTLS Mul-Uni)

In this setting the client still starts with the handshake with every server but then it sends one DTLS-secured multicast request message instead of three unicast messages. To protect the multicast request, the client uses parameters from a Group Security Association (GSA) which is preloaded into the client and the servers. In real settings the delivery of the GSA to group members would be done by a group controller and there are separate protocols for that. However, group key management is outside of scope of this thesis, therefore we manually preconfigure the GSA in the nodes. The servers use unicast secure DTLS sessions, which have been established through the handshakes, to protect their responses.

5) DTLS Multicast Request, Multicast Response (DTLS Mul-Mul)

The servers still send responses to the unicast address of the client, however they use security parameters from the GSA to protect the responses. This means that the nodes do not have to and therefore do not perform the handshake processes at all in this setting because the GSA is sufficient to secure messages in both directions. As in the setting 4, the GSA is preloaded into the client and the servers. The pre-shared keys for the handshake are not needed any more so it is excluded from the applications.

The key derivation algorithm described in Subsection 3.2.1 is performed on both client and server sides. The servers compute individual keys directly after the applications start. The client computes an individual key of a listener when the client receives the first response from this listener. Thereby, key derivation on a client can be considered as a part of response processing.

4.3 Evaluation Results

To evaluate our approach for secure group communication and compare it with others, we measure characteristics which are important for usability and conditions of constrained environment. First of all, memory occupancy is measured to evaluate memory overhead. Secondly, time required for performing a transaction, computation and handshake process is measured. Finally, the measurement of energy consumption for different operations is done as crucial for constrained devices.

All the experiments are done under the same layout of the nodes, hence signal transmission time can be considered constant.

4.3.1 Memory Footprint

We measure Read-Only Memory (ROM) and Random Access Memory (RAM) footprints for all our settings. RAM consumption is defined by statically preinitialised and pre-zeroed variables, whereas ROM consumption is basically a size of an image loaded into a board. The obtained results refer to memory consumption for the whole Contiki image which includes the whole communication stack. Admittedly, TCP and RPL routing protocol are switched off and not included in the images to save memory since they are not used in our scenario.

Before proceeding to the settings, one additional detail has to be discussed. As we mention in the Subsection 4.1.1, we have implemented the CoAP adaptation for multicast communication. Obviously, it contributes to memory consumption and can affect our measurements because in the two settings only unicast communication is used and hence the CoAP multicast adaptation can be switched off. To estimate the contribution, we measure memory consumption for the first setting *CoAP Unicast* with included CoAP multicast adaptation and without it. The results are presented in Table 4.1.

As it can be seen, the adaptation adds only 48 bytes into ROM to both the client and server applications. This overhead can be considered as negligible so further on we keep the adaptation included for all settings to achieve consistency.

	ROM, bytes		RAM, bytes	
	client	server	client	server
Without multicast	44,936	45,312	13,722	13,970
With multicast	44,984	45,360	13,722	13,970

Table 4.1. Memory occupancy without and with CoAP adaptation for multicast communication

The results of followed memory consumption measurement for our experimental settings are shown in Table 4.2. The ROM and RAM footprints are also shown as bar charts in Figure 4.4 and Figure 4.5 respectively for illustrative purposes.

Setting	ROM, bytes		RAM, bytes	
	client	server	client	server
1 CoAP Unicast	44,984	45,360	13,722	13,970
2 CoAP Multicast	44,248	45,412	13,428	13,970
3 DTLS Unicast-Unicast	63,276	63,364	18,922	19,194
4 DTLS Multicast-Unicast	64,052	64,992	18,962	19,202
5 DTLS Multicast-Multicast	64,848	66,364	16,632	16,142

Table 4.2. Memory occupancy in the experimental settings

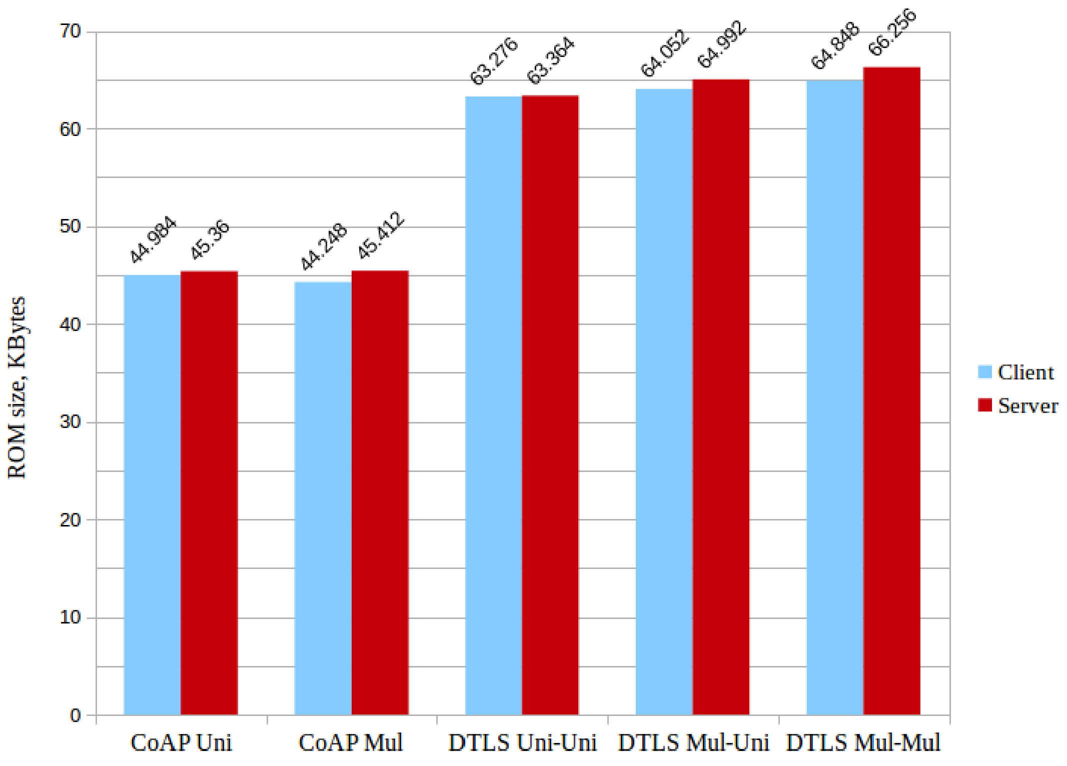


Figure 4.4. ROM footprint in the experimental settings

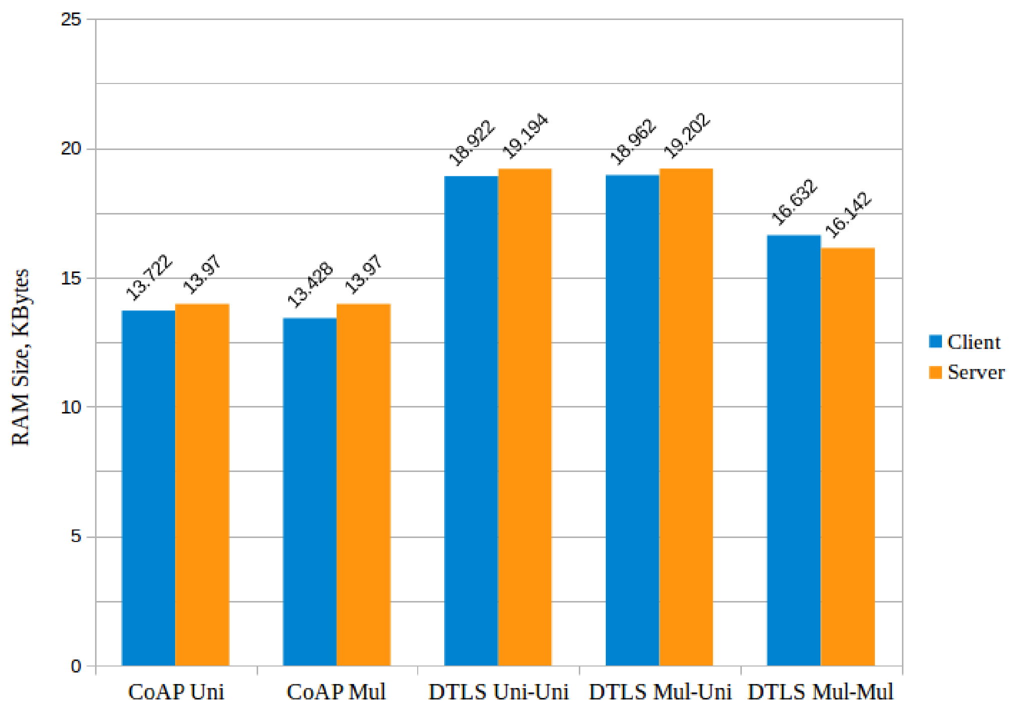


Figure 4.5. RAM footprint in the experimental settings

The ascending trend of ROM size is explicable because we gradually add functionality to the scenario from setting to setting. The decrease in ROM and RAM consumption for the CoAP Multicast setting compare to the CoAP Unicast setting is mainly due to the fact that the client here does not have to have entries with unicast and MAC addresses of the servers in its neighbours routing table to deliver them the multicast requests and no other unicast communication is needed to take place.

Furthermore, the server application mostly requires slightly more ROM and RAM than the client application because it includes functionality for processing REST request in application itself whereas the client does not need it.

Probably the most important result is that the approach proposed by us requires 2.3 KB and 3 KB less RAM for the client and server applications respectively for full group communication than an approach proposed in IETF draft by DICE Working group [20]. Simultaneously, ROM requirement is only about 1 KB higher for both the client and the server. Overall, RAM requirements are the most crucial for constrained devices because it has much less size capacity than ROM. There are several reasons for that such as RAM requires constant energy supply and is generally more expensive than ROM in terms of money per KB.

The main cause of the difference in memory requirements between the secure setting is how many cipher contexts and hash contexts can be used in parallel and, therefore, memory for them needs to be allocated. In our scenario, DTLS Uni-Uni requires 6 cipher contexts and 9 hash contexts, DTLS Mul-Uni – 7 cipher contexts and 10 hash contexts, DTLS Mul-Mul – only 4 cipher contexts and 4 hash contexts. Particularly, a client needs one write context to send multicast messages and three read contexts to process replies.

Lastly, we would like to examine scalability of our approach and compare it against traditional DTLS, i.e. to see how memory requirements change depending on number of listeners in the group. We measure changes on the client because size of the group does not affect servers and they are not supposed to know what other servers are present in the group. Figure 4.6 demonstrates memory requirements based on a number of listeners. We limit our examination by 12 listeners in DTLS Uni-Uni and by 40 listeners in DTLS Mul-Mul because the cc2538 has 32 KB of RAM and further increase of the number of listeners exceeds available memory.

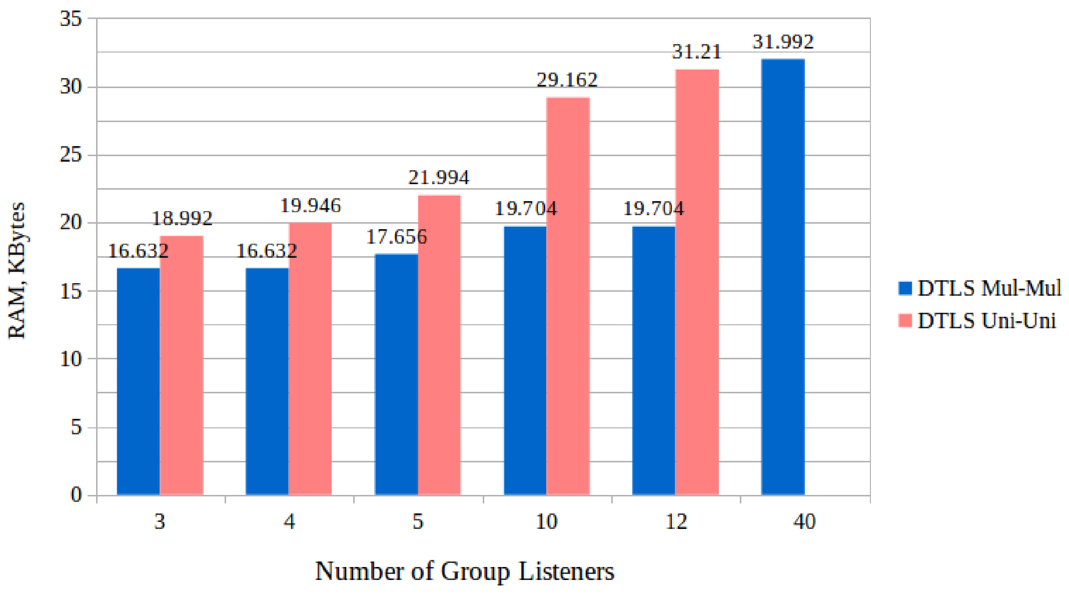


Figure 4.6. RAM occupancy on a client for DTLS Mul-Mul (blue) and DTLS Uni-Uni (pink) depending on number of group listeners

It is clear that traditional DTLS approach reaches the maximum amount of memory already with 12 listeners, whereas our approach requires similar amount of memory for 40 listeners. Therefore, we can safely claim that our approach is more scalable than the traditional one and, overall, it can be used in practice. It should be mentioned that an approach by DICE working group requires even more memory than traditional DTLS because group contexts need to be allocated in addition to unicast ones. Also, Contiki OS does not allocate memory gradually but rather by bulks. Hence, two additional listeners may not make any difference in some cases.

4.3.2 Communication Performance

To evaluate usability and efficiency, we measure time required to perform one *group transaction*. The group transaction is a term that we define as obtaining information by a client with one request from all servers. Different actions need to be accomplished by the client and the servers, depending on what setting is performed in order to accomplish required information exchange. Here are what transaction consists of for every setting:

1. In CoAP Unicast setting, the transaction starts when the client starts preparation of the CoAP request message to transmit it to the first server and ends when the client finishes processing a response from the last server. In this setting, the client sends the request to next server only when it has received and processed a response from previous one.
2. In CoAP Multicast, the transaction starts when the client starts preparation of the CoAP multicast request and ends when the client finishes processing the third response at the CoAP level. Note that since the servers undergo a random delay before replying, order of incoming responses will also be arbitrary so the third response can be arriving from any of the servers. Also, transaction duration becomes rather a random value in some range. However, it should reach some stable mean value.
3. In DTLS Unicast-Unicast, the transaction starts when the client initiate a handshake process with the first server and then proceeds with other two servers. After finishing the handshake with the last server, the client proceeds similarly to the CoAP Unicast setting, with only difference that the CoAP request is passed to the DTLS layer to be protected before the transport layer and responses are decrypted before passing to the CoAP level.
4. In DTLS Unicast-Multicast, the transactions also starts with performing handshakes but then follows multicast approach from the CoAP Multicast setting with addition of DTLS layer

for message processing. The transaction is done when the last reply is received.

5. In DTLS Multicast-Multicast, no handshake is performed so the transaction does not include it. The transaction includes sending the request, performing the key hashing computation upon receiving every response and the actual processing of the received responses. As in the CoAP Multicast setting, the transaction starts when the client starts preparation of the CoAP multicast request which is then passed down to the DTLS layer. The transaction proceeds with the client receiving the first response, creating an entry with the responding group listener and calculating individual Server Write and Server Write MAC keys based on a unicast IP address and a port number of this new group listener, and processing the actual response. The same operations are repeated for the second and third responses. The transaction ends when the last third response is processed.

Note that in real operation, handshakes and individual key derivation will be performed only once for a given period of key freshness so only the very first transaction includes them and later ones include only the request-response part. However, the handshakes and the key derivation are still needed to be performed after every rekeying which can be quite frequent in arrangements with many group members, especially if they periodically join and leave. Hence, we consider a full version of the transactions to give a picture of the worst case scenario in terms of communication performance and energy consumption.

The precision of our measurement is 1 clock tick which equals to about 8 ms on CC2538DK. To accomplish the evaluation, we run 20 experiments for every setting and then calculate average values and average deviation for every value. Figure 4.7 and Table 4.3 presents overall results of transaction time measurement.

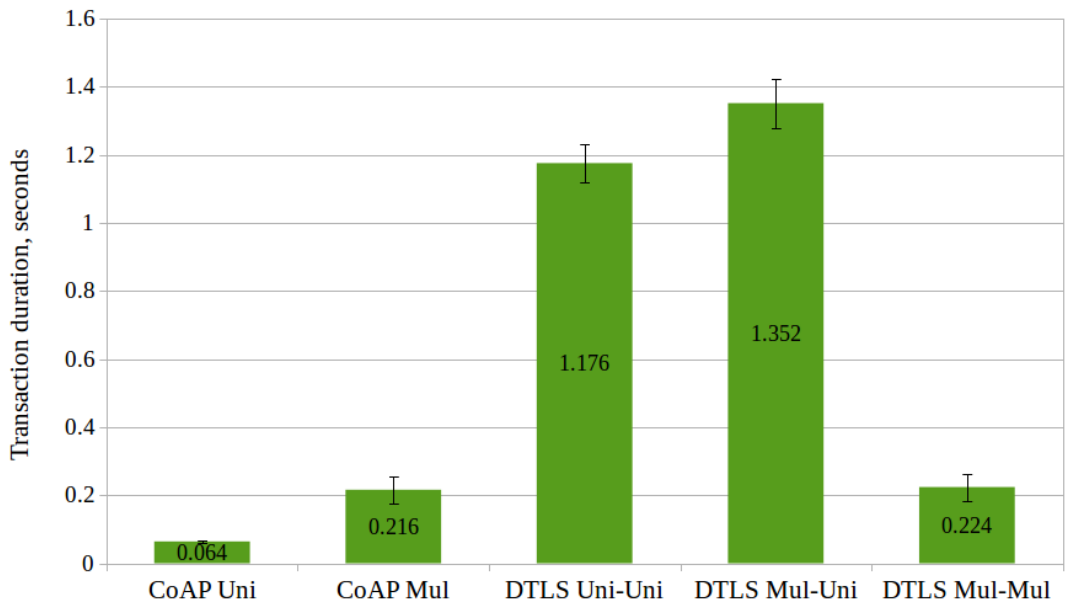


Figure 4.7. Transaction duration in examined scenario for settings of insecure unicast, insecure multicast, secure unicast, secure partially multicast, and secure multicast communication settings

	Average Transaction duration, sec	Average deviation, sec
CoAP Uni	0.064	0.004
CoAP Mul	0.216	0.040
DTLS Uni-Uni	1.176	0.056
DTLS Mul-Uni	1.352	0.072
DTLS Mul-Mul	0.224	0.040

Table 4.3. Transaction duration values in examined scenario for five settings

It can be seen from the table that there is more than three times difference in duration between the CoAP Unicast and CoAP Multicast settings. This is due to a random delay in the range from 0 to 0.25 second for replies to multicast requests which we have decided to keep consistent for all the multicast settings. Without the delay, the transaction in theory could be performed even faster that in unicast scenario because the client needs to send only one message. Development of an algorithm which would allow the servers efficiently pick a moment to transmit a reply and avoid collisions can be considered as a future work. This is not an easy question because a distance between the sender and each server differs and, thereby, signal propagation time differs, while the algorithm should solve the issue for a random and possibly dynamically changing layout of nodes.

Settings 3 and 4 include the handshake processes and this is a main reason why they demonstrate significantly longer time. This slowness will grow more and more with increasing number of group members. What is important, our approach is about six times faster than the mixed multicast-unicast approach, more than five times faster than the traditional unicast approach, and even comparable with insecure CoAP Multicast, keeping in mind the identical leisure time in them. Since the mixed approach by the DICE Working group represented by the Setting 4 includes both handshakes and random delays, it is inferior in terms of computation performance even to the straight approach with separate unicast sessions.

We also individually measure time spent on every handshake and on actual request part which compose together overall transaction duration. The results for settings 3, 4 and 5 are shown in Table 4.4. Average deviation values are presented by the numbers with ± sign.

	Handshake 1, sec		Handshake 2, sec		Handshake 3, sec		Request, sec	
DTLS Uni-Uni	0.552	±0.032	0.264	±0.032	0.280	±0.040	0.080	±0.004
DTLS Mul-Uni	0.560	±0.024	0.280	±0.032	0.296	±0.032	0.208	±0.040
DTLS Mul-Mul							0.224	±0.040

Table 4.4. Timing for individual phases of a transaction for settings with secure communication

There are several valuable observations that can be taken from these results. Firstly, the handshake process with the first server takes about twice more time than with next servers. This is probably due to the fact that the client has to initialize hash and cryptographic functions at first invocation of the DTLS layer. Another important observation is that the actual request-response part in DTLS Unicast-Unicast requires only 0.016 seconds more time to be fully accomplished. This overhead is only about a quarter of initial time for CoAP Unicast, and the proportion will be only decreasing with growth of distance between nodes. Finally, duration of the request part is similar for the Setting 4 and Setting 5, taking into account the high deviation values caused by presence of randomness. This means that the key hashing operation practically does not make any considerable impact on transaction duration.

Finally, we have experienced a collision only once during our experiments, in the CoAP Multicast setting. Measurements for the affected repetition were discarded and the repetition was re-done. It is most likely that collision rate would grow substantially with decreasing the leisure time value.

4.3.3 Energy Consumption

Energy that a constrained node consumes during a defined amount of time can be composed of energy consumed by CPU, energy spent on data transmission, energy spent on data reception, energy spent on actual listening radio medium and, finally, energy consumed during staying in a low-power mode. In our evaluation, the communicating devices do not use any Radio Duty Cycling which is a class of mechanisms for decreasing node's power consumption by switching off the radio when it is not used [67]. Despite obvious advantages of power saving, usage of these mechanisms leads to a significant increase in packet loss which can create problems during, for example, performing a DTLS handshake. We consider a topic of efficient integration security protocols with radio duty cycling being outside of the scope of this work and leave it as a future work, mainly as a networking subject. Absence of radio duty cycling means that the devices never switch to low-power mode, always listen the medium and CPU is constantly working.

In order to measure energy consumption for the settings in the examined scenario, we use the same transaction structure as in the Subsection 4.3.2. The transactions are identical to ones considered before, nevertheless, performed measurement is more fine-grained. That being said, we measure energy consumed by CPU only during processing received messages or messages to be transmitted. This can include preparation of a CoAP request, processing CoAP responses or, additionally, their encryption and decryption in case of secure communication. We also measure energy consumed for message transmission and reception but omit energy that the nodes consume to constantly listen the medium. The scheme of consumption components being measured is depicted in Figure 4.8.

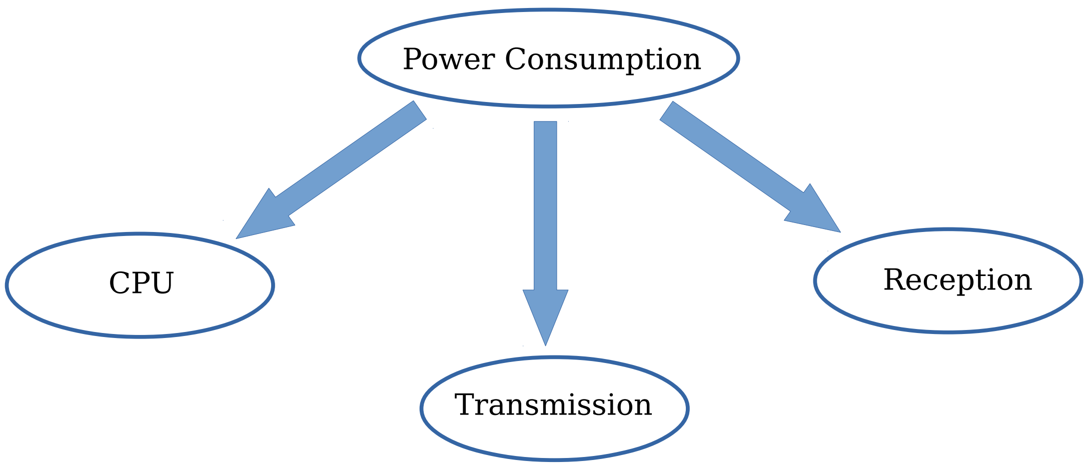


Figure 4.8. Considered components of power consumption on a constrained device in conducted experiments

Energy consumption of a component can be calculated as a multiplication of component's power consumption and a period of time during which the component has been active. Power consumption values can be collected from a data sheet of a specific board. The values for a board with the CC2538 chip that we use in our experiments are presented in Table 4.5. We measure duration of activity for each component using an Energest tool which is provided with Contiki OS. It is able to measure time spent of each activity during a given period with a precision 1/32768 seconds. We then multiply obtained numbers by values from the Table 4.5.

Component	Power Consumption, mW
CPU	21
Radio Transmission	72
Radio Reception	60

Table 4.5. Power consumption of different components on CC2538 chip

The measurement of time of activity for CPU and Transmission is straightforward. The Energest can derive necessary values at exact moments of program execution. The most challenging part of the experiment is how to measure duration of packet reception. The boards are listening all the time, and there is no difference in energy consumption from the listening or receiving points of view. It is also not feasible to anticipate when next packet will arrive and turn on measuring right before that. This leads to a difficulty of how to measure time spent precisely on reception but not hollow listening. We argue that for a given message with a constant size, it takes the same amount of time to be transmitted and to be received. The argumentation is that there is a single bit rate of communication among the boards, and this bit rate is applied to reception as to transmission. In order to examine validity of this assumption, we have conducted a simple experiment shown in Figure 4.9 where we derived how many bits a listener can receive per one minute if a stream of incoming packets is continuous.

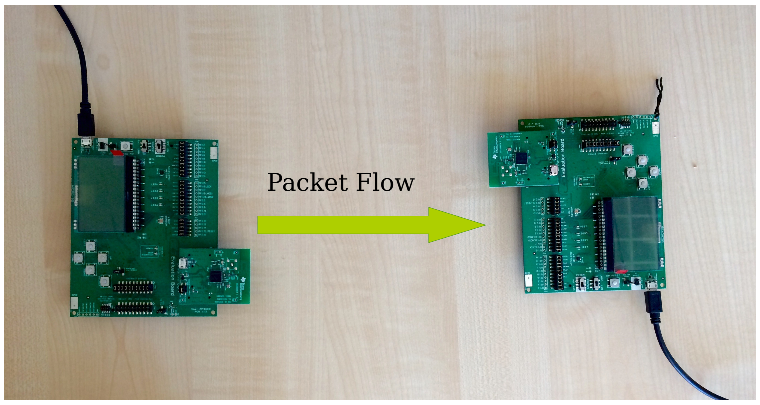


Figure 4.9. Experimental setup for deriving actual bit rate

Having precise values for transmission time obtained by Energest, we have calculated a bit rate for packet transmission. The results were in the same range as the values obtained in the experiment. This has proved our assumption being valid under given conditions. As a result, estimation of reception energy consumption is done in our scenario based on message size and how much time it has taken to be transmitted.

Figure 4.10 and Figure 4.11 demonstrate overall energy consumption per transaction in the examined settings for the client application and for the server application respectively. Error bars are not included due to their negligible values. Table 4.6 presents detailed values for each consumption component.

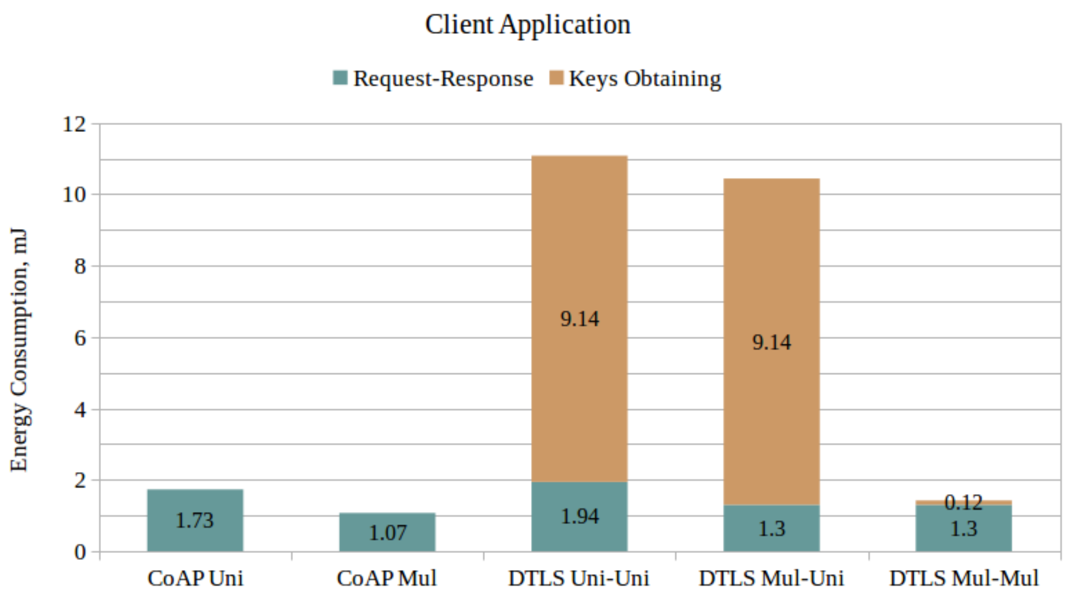


Figure 4.10. Energy consumption per transaction on client side for five examined settings. The Keys obtaining is DTLS handshake in Settings 3 and 4, and key hashing in Setting 5

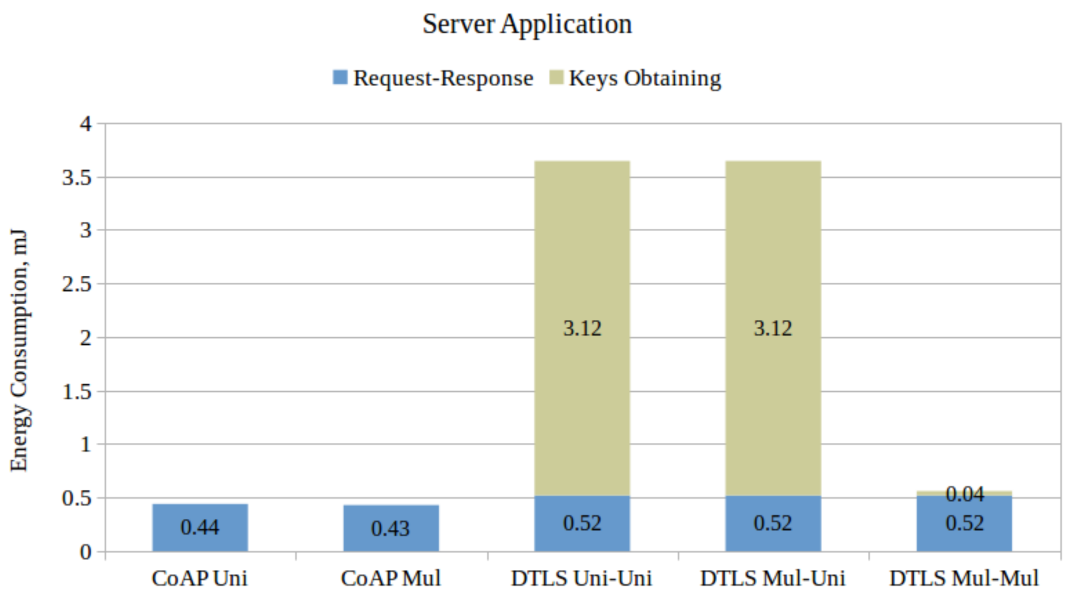


Figure 4.11. Energy consumption per transaction on server side for five examined settings. The Keys obtaining is DTLS handshake in Settings 3 and 4, and key hashing in Setting 5

Setting	Energy Consumption, mJ											
	Client						Server					
	Keys obtaining			Request-Resp			Keys obtaining			Request-Resp		
	CPU	Tran	Rec	CPU	Tran	Rec	CPU	Tran	Rec	CPU	Tran	Rec
CoAP Uni				0.65	0.67	0.40				0.10	0.15	0.19
CoAP Mul				0.46	0.21	0.40				0.10	0.15	0.18
DTLS Uni-Uni	2.09	3.91	3.14	0.77	0.61	0.56	0.78	1.26	1.08	0.13	0.22	0.17
DTLS Mul-Uni	2.09	3.91	3.14	0.54	0.20	0.56	0.78	1.26	1.08	0.14	0.22	0.16
DTLS Mul-Mul	0.13			0.54	0.20	0.56	0.04			0.14	0.22	0.16

Table 4.6. Energy consumption per component per transaction on client and server sides for five examined settings

An advantage of using multicast communication instead of multiple unicast ones can be clearly seen in Figure 4.10. The client application in the CoAP Unicast setting consumes approximately 50% more energy per transaction than in the CoAP Multicast setting. Radio transmission is the most power-consuming operation on a board with the cc2538 chip and similarly with other chips. In the multicast setting, the client transmits only one message instead of three separate ones and, moreover, a multicast packet can be more efficiently compressed on the 6LoWPAN layer. In our scenario, a size of the CoAP unicast request is 86 bytes, whereas a size of the CoAP multicast request is only 80 bytes. The difference in energy consumption between multicast and unicast communication will grow steadily with increasing number of listeners in a group. As it can be seen in Figure 4.11, there is almost no difference between the two insecure settings on the server application. The only slight distinction is caused by different size of incoming unicast or multicast request because this leads to difference in amount of energy spent on packet reception.

Introducing security significantly increases energy consumption in Settings 3 and 4 on both the client and the servers mainly due to a highly consuming DTLS handshake process. The handshake alone consumes approximately 3.05 mJ on the client side and 3.12 mJ on the server side. Since the client has to perform three handshake in our scenario, the consumption triples and reaches approximately 9.14 mJ. It can grow up to critical values in case of, for example, 100 group members which is a plausible number according to documentation for CoAP group communication [16]. At the same time, key derivation in the Setting 5 based on our approach consumes only 0.04 mJ per listener! As a result, the client application in the secure DTLS Mul-Mul setting consumes even less energy than insecure CoAP Unicast! This means that with our approach introduction of security into a system may even not require any additional resources on client side in case if the system is also switching from unicast to multicast communication. A server consumes approximately 0.12 mJ more energy per transaction in DTLS Mul-Mul than in both CoAP Uni and CoAP Mul which means 20% increase.

Finally, the results demonstrate that security computations over one-request event (decryption of a request and encryption of a response on the server, encryption of a request and decryption of a response on the client) consume in a range from 0.07 to 0.09 mJ depending on a size of messages. These values include CPU consumption as well as transmission and reception consumption.

Chapter 5

Conclusions and Future Work

This thesis work aimed at creating a framework for efficient secure group communication which could be used out of the box, particularly, in the rapidly growing world of the Internet of Things. The DTLS protocol for secure end-to-end communication, its lightweight implementation in Contiki OS and proposals of adapting the protocol for group communication were chosen as a basis for our development.

Firstly, we implemented support for secure multicast communication which allowed presence of multiple listeners as well as multiple senders in one group and also provided reply protection. We then justified why we found an approach of establishing separate secure unicast sessions for listener's response messages to multicast requests inefficient. The existing approach for response protection was analyzed and several critical flaws were discovered. The approach was vulnerable to a replay attack due to identical DTLS headers of records from different group listeners and also did not protect group listeners from reuse of identical {key, initialization vector} pairs which might lead to complete disclosure of plaintext in case of authenticated encryption. We proposed a mechanism how improve the examined approach and, thereby, mitigate all the discovered attacks. Furthermore, the measures of how to make the approach resistant against Denial-of-Service attacks in the recourse-constrained networks of the IoT were proposed and other possible security considerations were discussed. The improved approach was also implemented and integrated into the framework.

Experimental setup, with one node, running a client application, as a multicast sender and three nodes, running server applications, as group listeners, was created. The communication scenario included transmission of a multicast request from the client and transmission of unicast responses from the servers. In order to evaluate the approach that we had proposed, we examined five different settings of the scenario: insecure unicast, insecure multicast, secure fully unicast, secure partially multicast, and secure fully multicast communication. The settings were examined in terms of memory occupancy, communication performance and energy consumption. Our approach demonstrated slightly higher ROM requirements but significantly lower RAM requirements compare to the traditional DTLS approach and the DTLS approach with only multicast extension. It also was considered to have good scalability. The communication performance of our approach surpassed the two other secure approaches considerably and was even comparable with insecure multicast setting. Finally, our approach demonstrated remarkable results in energy consumption. The consumption on the client was approximately 22% lower than for insecure unicast communication and about 33% higher than for insecure multicast communication. The servers consumed approximately 27% more energy with our approach than with insecure communication. Simultaneously, the two other secure approaches demonstrated more than 800% and 700% increase in energy consumption compare to insecure communication on client and server sides respectively.

5.1 Future Work

To begin with, group key management protocols can be analyzed and the most suitable one can be implemented and integrated in our framework as a future work. This would eliminate a necessity in direct distribution of keying material among group members by a group controller. Naturally, the enriched framework could be further evaluated then.

Moreover, the studied approaches for secure group communication could be evaluated with presence of Radio Duty Cycling. It would be valuable to examine if the proposed by us approach remain advantages of its communication performance and energy consumption in the environment with a high packet loss rate and frequent retransmissions.

Lastly, an algorithm for discovering and applying optimal delay time before transmission of unicast responses by group listeners can be designed. This could potentially increase performance of group communication by reducing response waiting time.

References

[1] J. Zheng, D. Simplot-Ryl, C. Bisdikian, and H. T. Mouftah, “The internet of things [Guest Editorial],” *IEEE Commun. Mag.*, vol. 49, no. 11, pp. 30–31, Nov. 2011.

[2] R. Khan, S. U. Khan, R. Zaheer, and S. Khan, “Future Internet: The Internet of Things Architecture, Possible Applications and Key Challenges,” in *2012 10th International Conference on Frontiers of Information Technology (FIT)*, 2012, pp. 257–260.

[3] D. Miorandi, S. Sicari, F. De Pellegrini, and I. Chlamtac, “Internet of things: Vision, applications and research challenges,” *Ad Hoc Netw.*, vol. 10, no. 7, pp. 1497–1516, Sep. 2012.

[4] J. Höller, V. Tsiatsis, C. Mulligan, S. Karnouskos, S. Avesand, and D. Boyle, “Chapter 2 - M2M to IoT – The Vision,” in *From Machine-To-Machine to the Internet of Things*, J. Höller, V. Tsiatsis, C. Mulligan, S. Karnouskos, S. Avesand, and D. Boyle, Eds. Oxford: Academic Press, 2014, pp. 9–37.

[5] R. Wittmann and M. Zitterbart, *Multicast Communication: Protocols, Programming, & Applications*. Morgan Kaufmann, 2000.

[6] R. Roman, P. Najera, and J. Lopez, “Securing the Internet of Things,” *Computer*, vol. 44, no. 9, pp. 51–58, Sep. 2011.

[7] M. Wu, T.-J. Lu, F.-Y. Ling, J. Sun, and H.-Y. Du, “Research on the architecture of Internet of Things,” in *2010 3rd International Conference on Advanced Computer Theory and Engineering (ICACTE)*, 2010, vol. 5, pp. V5–484–V5–487.

[8] N. Salman, I. Rasool, and A. H. Kemp, “Overview of the IEEE 802.15.4 standards family for Low Rate Wireless Personal Area Networks,” in *2010 7th International Symposium on Wireless Communication Systems (ISWCS)*, 2010, pp. 701–705.

[9] Deering, S. and Hinden, R., “Internet Protocol, Version 6 (IPv6), Specification.” Network Working Group, Dec-1998.

[10] Postel, J., “RFC 768. User Datagram Protocol.” 28-Aug-1980.

[11] G. Mulligan, “The 6LoWPAN Architecture,” in *Proceedings of the 4th Workshop on Embedded Networked Sensors*, New York, NY, USA, 2007, pp. 78–82.

[12]Z. Shelby, K. Hartke, and C. Bormann, “RFC 7252. The Constrained Application Protocol (CoAP).” Internet Engineering Task Force, Jun-2014.

[13]P. J. Leach, T. Berners-Lee, J. C. Mogul, L. Masinter, R. T. Fielding, and J. Gettys, “RFC 2616. Hypertext Transfer Protocol -- HTTP/1.1.” Network Working Group, Jun-1999.

[14]L. Atzori, A. Iera, and G. Morabito, “The Internet of Things: A survey,” *Comput. Netw.*, vol. 54, no. 15, pp. 2787–2805, Oct. 2010.

[15]L. Tan and N. Wang, “Future internet: The Internet of Things,” in *2010 3rd International Conference on Advanced Computer Theory and Engineering (ICACTE)*, 2010, vol. 5, pp. V5–376–V5–380.

[16]A. Rahman and E. Dijk, “RFC 7390. Group Communication for the Constrained Application Protocol (CoAP).” Internet Engineering Task Force, Oct-2014.

[17]Tim Dierks, “RFC 5246. The Transport Layer Security (TLS) Protocol Version 1.2.” Network Working Group, Aug-2008.

[18]“RFC 793. TRANSMISSION CONTROL PROTOCOL.” Sep-1981.

[19]Eric Rescorla and Nagendra Modadugu, “RFC 6347. Datagram Transport Layer Security Version 1.2.” Internet Engineering Task Force, Jan-2012.

[20]O. Garcia-Morchon, S. Kumar, A. Rahman, E. Dijk, and S. Keoh, “DTLS-based Multicast Security in Constrained Environments, draft-keoh-dice-multicast-security-08.” DICE Working Group, 03-Jul-2014.

[21]M. Tiloca, “Efficient Protection of Response Messages in DTLS-Based Secure Multicast Communication,” in *Proceedings of the 7th International Conference on Security of Information and Networks*, New York, NY, USA, 2014, pp. 466:466–466:472.

[22]SICS Network Embedded Systems Group, “Contiki OS.” [Online]. Available: http://www.contiki-os.org/. [Accessed: 20-May-2015].

[23]M. Kovatsch, S. Duquennoy, and A. Dunkels, “A Low-Power CoAP for Contiki,” in *2011 IEEE 8th International Conference on Mobile Adhoc and Sensor Systems (MASS)*, 2011, pp. 855–860.

[24]Bergmann, Olaf, “tinyDTLS,” *SourceForge*. [Online]. Available: http://tinydtls.sourceforge.net/. [Accessed: 30-Mar-2015].

[25]“The Internet of Things (IoT) Starts with Intel Inside®,” *Intel*. [Online]. Available: http://www.intel.com/content/www/us/en/internet-of-

things/overview.html?cid=sem132p41839g-c&gclid=CLXgtZeQx8QCFVCVfgod3zYAsQ. [Accessed: 30-Mar-2015].

[26]“Internet of Things (IoT),” *Cisco*. [Online]. Available: http://www.cisco.com/web/solutions/trends/iot/overview.html. [Accessed: 30-Mar-2015].

[27]“GreenPeak - Building the Smart Home.” [Online]. Available: http://www.greenpeak.com/index.html. [Accessed: 30-Mar-2015].

[28]Dave Evans, “The Internet of Things. How the Next Evolution of the Internet Is Changing Everything,” Cisco IBSG, White Paper, Apr. 2011.

[29]“CEO to shareholders: 50 billion connections 2020,” *Ericsson.com*, 13-Apr-2010. [Online]. Available: http://www.ericsson.com/news/1403231. [Accessed: 01-Jun-2015].

[30]A. Håkansson, “Portal of Research Methods and Methodologies for Research Projects and Degree Projects,” presented at the The 2013 World Congress in Computer Science, Computer Engineering, and Applied Computing WORLDCOMP 2013; Las Vegas, Nevada, USA, 22-25 July, 2013, pp. 67–73.

[31]Salkind, Neil J., *Exploring Research*, 8th ed. 2011.

[32]W. M. K. Trochim and J. P. Donnelly, *The Research Methods Knowledge Base*, 3rd edition. Mason, Ohio: Atomic Dog, 2006.

[33]Tim Winter et al., “RFC 6550. RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks.” Internet Engineering Task Force, Mar-2012.

[34]M. R. Palattella, N. Accettura, X. Vilajosana, T. Watteyne, L. A. Grieco, G. Boggia, and M. Dohler, “Standardized Protocol Stack for the Internet of (Important) Things,” *IEEE Commun. Surv. Tutor.*, vol. 15, no. 3, pp. 1389–1406, Third 2013.

[35]C. Sammarco and A. Iera, “Improving Service Management in the Internet of Things,” *Sensors*, vol. 12, no. 9, pp. 11888–11909, Aug. 2012.

[36]S. Pollin, M. Ergen, S. Ergen, B. Bougard, L. Der Perre, I. Moerman, A. Bahai, P. Varaiya, and F. Catthoor, “Performance Analysis of Slotted Carrier Sense IEEE 802.15.4 Medium Access Layer,” *IEEE Trans. Wirel. Commun.*, vol. 7, no. 9, pp. 3359–3371, Sep. 2008.

[37]N. Kushalnagar, G. Montenegro, D. E. Culler, and J. W. Hui, “RFC 4944. Transmission of IPv6 Packets over IEEE 802.15.4 Networks.” Network Working Group, Sep-2007.

[38]J. Ko, A. Terzis, S. Dawson-Haggerty, D. E. Culler, J. W. Hui, and P. Levis, “Connecting low-power and lossy networks to the internet,” *IEEE Commun. Mag.*, vol. 49, no. 4, pp. 96–101, Apr. 2011.

[39]K. Egevang and P. Francis, “The IP Network Address Translator (NAT),” 1994.

[40]“How TLS/SSL Works: Logon and Authentication.” [Online]. Available: https://technet.microsoft.com/en-us/library/cc783349(v=ws.10).aspx. [Accessed: 04-Apr-2015].

[41]H. Krawczyk, R. Canetti, and M. Bellare, “RFC 2104. HMAC: Keyed-Hashing for Message Authentication.” Network Working Group, Feb-1997.

[42]National Institute of Standards and Technology, “FIPS PUB 180-4. Secure Hash Standard (SHS).” Federal Information Processing Standards, Mar-2012.

[43]T. Hardjono and B. Weis, “RFC 3740. The Multicast Group Security Architecture.” Network Working Group, Mar-2004.

[44]U. Meth, A. Colegrove, G. Gross, and H. Harney, “RFC 4535. GSAKMP: Group Secure Association Key Management Protocol.” Network Working Group, Jun-2006.

[45]D. Bailey and D. McGrew, “RFC 6655. AES-CCM Cipher Suites for Transport Layer Security (TLS).” Internet Engineering Task Force, Jul-2012.

[46]A. Choudhury, D. McGrew, and J. Salowey, “RFC 5288, AES Galois Counter Mode (GCM) Cipher Suites for TLS.” Network Working Group, Aug-2008.

[47]National Institute of Standards and Technology, “FIPS PUB 197. Advanced Encryption Standard (AES).” Federal Information Processing Standards, 26-Nov-2011.

[48]Sandeep Kumar, “Group Communication Security for Low-Power and Lossy Networks (LLNs), draft-kumar-dice-groupcomm-security-00.” DICE Working Group, 02-Jul-2014.

[49]K. Seo and S. Kent, “RFC 4301. Security Architecture for the Internet Protocol.” Network Working Group.

[50]Stephen Kent, “RFC 4303. IP Encapsulating Security Payload (ESP).” Network Working Group, Dec-2005.

[51]Stephen Kent, “RFC 4302. IP Authentication Header.” Network Working Group.

[52]Charlie Kaufman, “RFC 4306. Internet Key Exchange (IKEv2) Protocol.” Network Working Group, Dec-2005.

[53]N. Doraswamy and D. Harkins, *IPSec: The New Security Standard for the Internet, Intranets, and Virtual Private Networks, Second Edition*, Second. Prentice Hall, 2003.

[54]D. Ignjatic, G. Gross, and B. Weis, “RFC 5374. Multicast Extensions to the Security Architecture for the Internet Protocol.” Network Working Group, Nov-2008.

[55]S. Raza, S. Duquennoy, T. Chung, D. Yazar, T. Voigt, and U. Roedig, “Securing communication in 6LoWPAN with compressed IPsec,” in *2011 International Conference on Distributed Computing in Sensor Systems and Workshops (DCOSS)*, 2011, pp. 1–8.

[56]N. Modadugu and E. Rescorla, “The Design and Implementation of Datagram TLS.,” in *NDSS*, 2004.

[57]D. A. McGrew and J. Viega, “The Security and Performance of the Galois/Counter Mode (GCM) of Operation,” in *Progress in Cryptology - INDOCRYPT 2004*, A. Canteaut and K. Viswanathan, Eds. Springer Berlin Heidelberg, 2004, pp. 343–355.

[58]Jeff Preshing, “Hash Collision Probabilities,” 04-May-2011. [Online]. Available: http://preshing.com/20110504/hash-collision-probabilities/. [Accessed: 17-May-2015].

[59]“An Introduction to Cooja,” *GitHub*. [Online]. Available: https://github.com/contiki-os/contiki. [Accessed: 20-May-2015].

[60]Kovatsch, Matthias, Duquennoy, Simon, and Dunkels, Adam, “Erbium (Er) REST Engine and CoAP Implementation for Contiki.” [Online]. Available: http://people.inf.ethz.ch/mkovatsc/erbium.php. [Accessed: 01-Jun-2015].

[61]“CC2538 Evaluation Module Kit.” [Online]. Available: http://www.ti.com/tool/cc2538emk. [Accessed: 20-May-2015].

[62]“SmartRF06 Evaluation Board.” [Online]. Available: http://www.ti.com/tool/smartrf06ebk. [Accessed: 20-May-2015].

[63]“CC2538. A Powerful System-On-Chip for 2.4-GHz IEEE 802.15.4-2006 and ZigBee Applications.” [Online]. Available: http://www.ti.com/product/cc2538. [Accessed: 20-May-2015].

[64]Andrew Dodd, “CCSniffPiper. Live Packet Sniffer to Wireshark bridge for IEEE 802.15.4 networks.” [Online]. Available: https://github.com/andrewdodd/ccsniffpiper. [Accessed: 12-Jun-2015].

[65]“Wireshark.” [Online]. Available: https://www.wireshark.org/. [Accessed: 12-Jun-2015].

[66]Jelmer Tiete, “TI CC2538 Serial Boot Loader,” *GitHub*. [Online]. Available: https://github.com/JelmerT/cc2538-bsl. [Accessed: 20-May-2015].

[67]“Contiki Wiki. Radio Duty Cycling,” *GitHub*, 14-Dec-2014. [Online]. Available: https://github.com/contiki-os/contiki. [Accessed: 11-Jun-2015].

TRITA TRITA-ICT-EX-2015:62