Secure Multicast with Source Authentication for the Internet of Things

NIKITA MARTYNOV
Secure Multicast with Source Authentication for the Internet of Things

Nikita Martynov

Eindhoven, The Netherlands, 2014
Abstract (English)

The Internet of Things is a rapidly evolving field of high-end technology and research. Its security is vital to the reliability and safety of the future everyday communications. The DTLS protocol is a default protocol to assure security for unicast communication. A DTLS record layer extension for multicast in constrained environments is being designed to assure security for multicast. However, currently proposed DTLS-based multicast does not provide such an essential property as source authenticity for the transmitted data. Moreover, handshake layer is designed to establish pairwise keys only, and hence, there is no way to distribute and manage group keys either. The two aforementioned downsides become the primary objectives of the design for the thesis.

This thesis is conducted in collaboration with Philips. In the thesis, we formulate requirements to secure multicast in constrained environment based on the company’s outdoor lighting scenario with centralized trust model. We evaluate various source authentication schemes and key management protocols with regards to the formulated requirements. We select two authentication schemes and apply them to our scenario. As a result we design an extension of DTLS based multicast with support of ECDSA signature for source authentication and we develop a prototype implementation. Besides that, we determine cryptographic primitives for the TESLA scheme and adapt the scheme to be used for periodic communication pattern. Further, we design a lightweight and flexible group key management solution to distribute group keys and public keys by the trusted authority.

This thesis is a final project of the Erasmus Mundus Double Master’s Degree in Nordic Security and Mobile Computing (NordSecMob) with the study track: Royal Institute of Technology (KTH), Sweden and Technical University of Denmark (DTU), Denmark. The thesis was conducted in close collaboration with Philips Research Lighting, the Netherlands. The main supervisor of the thesis is Prof. Andrey Bogdanov at the Department of Applied Mathematics and Computer Science of DTU and the supervisor from KTH is Prof. Markus Hidell at the Department of Communication Systems. The advisors from Philips Research Lighting are Oscar Garcia-Morchon and Sahil Sharma.

The thesis was prepared during the internship at Philips Research Lighting, The Netherlands. The thesis deals with the secure multicast communication for the Internet of Things. It is focused on the scenario where an outdoor lighting system is controlled from a back-end server.

Eindhoven, 30-June-2014

Nikita Martynov
Acknowledgements

I would like to thank my supervisor Prof. Andrey Bogdanov at Technical University of Denmark for valuable discussions and comments, for constructive feedback and suggestions during my work on the thesis. His guidance helped me in time of research and writing of the thesis. I am also grateful to Prof. Markus Hidell at Royal Institute of Technology for being my supervisor and providing guidance with regards to the master thesis.

I would like to thank Oscar García-Morchon and Sahil Sharma, my advisors from Philips Research, for continuous guidance and thoughtful advice. Their expertise in the field, regular feedback and suggestions helped to improve the quality of the thesis.

I would like to express my sincere gratitude to my family and my precious girlfriend for their invaluable support throughout these 2 years of study.

Last but not least, I am thankful to the people involved in NordSecMob program for the opportunity to obtain exceedingly great education.

Eindhoven, 30-June-2014

Nikita Martynov
Abbreviations

6LoWPAN IPv6 over Low power Wireless Personal Area Networks
AES Advanced Encryption Standard
ASN.1 Abstract Syntax Notation One
BAS Bloom filter-based Authentication Scheme
BLS Boneh–Lynn–Shacham
CBC Cipher-Block Chaining
CCM Counter with CBC-MAC
CoAP Constraint Application Protocol
DICE DTLS In Constrained Environments
DLP Discrete Logarithm Problem
DSA Digital Signature Algorithm
DTLS Datagram Transport Layer Security
ECDLP Elliptic Curve Discrete Logarithm Problem
ECDSA Elliptic Curve Digital Signature Algorithm
EIBAS Efficient Identity based Broadcast Authentication Solution
GDOI Group Domain of Interpretation
GPS Global Positioning System
Abbreviations

**GSAKMP** Group Secure Association Key Management Protocol

**HMAC** Keyed-Hash Message Authentication Code

**HTTP** HyperText Transport Protocol

**IETF** Internet Engineering Task Force

**IoT** Internet of Things

**JSON** JavaScript Object Notation

**KDF** Key Derivation Function

**LKH** Logical Key Hierarchy

**LLN** Low-power and Lossy Network

**M2M** Machine-to-Machine

**MAC** Message Authentication Code

**MAF** Message Authentication Function

**MPL** Multicast Protocol for Low power and Lossy Networks

**NIST** National Institute of Standards and Technology

**RPL** IPv6 Routing Protocol for Low power and Lossy Networks

**RTT** Round Trip Time

**OS** Operating System

**PAN** Personal Area Network

**PBA** Polynomial Based Authentication

**SHA** Secure Hash Algorithm

**TCP** Transmission Control Protocol

**TLS** Transport Layer Security

**TESLA** Timed Efficient Stream Loss-tolerant Authentication

**UDP** User Datagram Protocol

**XML** Extensible Markup Language

**ZSS** Zhang–Safavi-Naini–Susilo
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Protocol stack</td>
<td>9</td>
</tr>
<tr>
<td>2.2</td>
<td>Packet format with minimal protocol header overheads</td>
<td>9</td>
</tr>
<tr>
<td>2.3</td>
<td>DTLS record format</td>
<td>12</td>
</tr>
<tr>
<td>2.4</td>
<td>DTLS handshake exchanges</td>
<td>13</td>
</tr>
<tr>
<td>2.5</td>
<td>DTLS multicast record format</td>
<td>14</td>
</tr>
<tr>
<td>2.6</td>
<td>TESLA protocol</td>
<td>17</td>
</tr>
<tr>
<td>2.7</td>
<td>GSAKMP’s join and rekey message exchanges</td>
<td>23</td>
</tr>
<tr>
<td>2.8</td>
<td>GDOI’s GROUPKEY-PULL and GROUPKEY-PUSH message exchanges</td>
<td>24</td>
</tr>
<tr>
<td>2.9</td>
<td>AMIKEY’s join and rekey message exchanges</td>
<td>27</td>
</tr>
<tr>
<td>2.10</td>
<td>AMIKEY GM’s message packet format</td>
<td>27</td>
</tr>
<tr>
<td>2.11</td>
<td>AMIKEY message sent by a GCKS</td>
<td>29</td>
</tr>
<tr>
<td>3.1</td>
<td>Use-case scenario</td>
<td>34</td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>-------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>4.1</td>
<td>Payload size per frame with ECDSA signature</td>
<td>49</td>
</tr>
<tr>
<td>4.2</td>
<td>Primitive storage optimization</td>
<td>53</td>
</tr>
<tr>
<td>4.3</td>
<td>Adaptation of TESLA to the main use-case</td>
<td>58</td>
</tr>
<tr>
<td>4.4</td>
<td>Payload size per frame with TESLA scheme</td>
<td>59</td>
</tr>
<tr>
<td>5.1</td>
<td>Options of key management setup with AMIKEY</td>
<td>73</td>
</tr>
<tr>
<td>5.2</td>
<td>Advanced scenario</td>
<td>74</td>
</tr>
<tr>
<td>5.3</td>
<td>Multicast Group ID payload</td>
<td>80</td>
</tr>
<tr>
<td>6.1</td>
<td>Packet format achieved at simulation</td>
<td>88</td>
</tr>
<tr>
<td>6.2</td>
<td>Experimental scenario</td>
<td>89</td>
</tr>
</tbody>
</table>
# List of Tables

2.1 Source authentication schemes comparison .................................. 21

3.1 Requirements covered by DTLS-based Secure Multicast for LLN .................. 38

4.1 NIST Security Level Recommendations ........................................ 44

4.2 Selection of TESLA parameters .................................................. 56

4.3 Comparison of the source authentication designs with frequent communication pattern (2 s periods) ................................................. 62

4.4 Comparison of the source authentication designs with the periodic communication pattern (12 hours periods) ............................................. 62

5.1 Key Management Protocols Comparison ........................................ 72

5.2 Security protocols list ............................................................... 79

5.3 Security policy parameters for DTLS multicast ................................ 79

5.4 Keying data types for Key data sub-payload .................................. 80

6.1 The memory footprint ............................................................... 90
Contents

Abstract (English) i
Abstract (Swedish) iii
Preface v
Acknowledgements vii
Abbreviations ix

1 Introduction 1
1.1 State-of-the-Art . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1
1.2 Problem Statement . . . . . . . . . . . . . . . . . . . . . . . . . . 3
1.3 Contributions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3
1.4 Structure of the Thesis . . . . . . . . . . . . . . . . . . . . . . . . 4

2 Background and Related Work 7
2.1 Resource-constrained IoT networks . . . . . . . . . . . . . . . . . 8
2.1.1 IP Multicast . . . . . . . . . . . . . . . . . . . . . . . . . . 10
2.1.2 Contiki Operating System . . . . . . . . . . . . . . . . . . . 10
2.2 Datagram Transport Layer Security . . . . . . . . . . . . . . . . . 11
2.2.1 DTLS Secured Multicast . . . . . . . . . . . . . . . . . . . . 13
2.3 Source Authentication Schemes . . . . . . . . . . . . . . . . . . . . 14
2.3.1 Elliptic Curve Digital Signature Algorithm . . . . . . . . . 15
2.3.2 Timed Efficient Stream Loss-tolerant Authentication . . . 16
2.3.3 Other Schemes . . . . . . . . . . . . . . . . . . . . . . . . . 18
2.3.4 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . 20
2.4 Key Management Schemes . . . . . . . . . . . . . . . . . . . . . . 20
2.4.1 Group Secure Association Key Management Protocol . . . 20
2.4.2 Group Domain of Interpretation .......................... 23
2.4.3 Multimedia Internet Keying .............................. 25
2.4.4 Adapted Multimedia Internet Keying ..................... 26
2.5 Related Work .................................................. 28
2.5.1 IPsec-based Secure Multicast ............................ 28
2.5.2 IPsec for IoT ................................................. 30

3 Problem Statement and Requirement Analysis 31
3.1 System and Security Model ................................. 32
3.2 Requirements to Secure Multicast over Constrained Networks 34
3.3 Requirements covered by DTLS-based Secure Multicast ........ 36
3.4 Motivation for Source Authentication at DTLS ............... 38
3.5 Problem Statement ............................................. 39
3.6 Design Goals .................................................. 40

4 Design of Source Authentication for DTLS-based Multicast 41
4.1 Selection of Prospective Candidate Schemes .................. 42
4.2 Source Authentication with ECDSA .......................... 43
4.2.1 Selection of Security Strength ............................ 43
4.2.2 Selection of the Curve ..................................... 45
4.2.3 Selection of the Encryption Mode for AES ................ 45
4.2.4 Combination of Signature and Encryption ................. 46
4.2.5 Applying ECDSA to the Main Use-case ................. 47
4.3 Source Authentication with TESLA .......................... 49
4.3.1 Selection of Parameters and Crypto Primitives .......... 49
4.3.2 TESLA Adaptation ......................................... 56
4.3.3 Combination of Authentication and Encryption .......... 57
4.3.4 Applying TESLA Scheme to the Main Use-case .......... 59
4.4 Comparison of the Design Options .......................... 59
4.5 Extension of the DTLS specification with source authentication 63

5 Design for Group Key Management 67
5.1 Formulation of the requirements for key management ........ 68
5.2 Fulfillment of the key management protocol requirements ...... 69
5.2.1 Selection of the protocols ................................ 70
5.2.2 Protocols Comparison ..................................... 70
5.3 Options of Key Management with AMIKEY .................... 73
5.4 Design of the AMIKEY based Key Management Solution ....... 76
5.4.1 Reduction of Superfluous Functionality .................. 76
5.4.2 Repurpose and Protocol Parameters Extension ........... 78
5.4.3 Node Join Discussion ..................................... 81
5.4.4 Node Eviction Discussion ................................. 82
5.4.5 Discussion .................................................. 82
5.4.6 Security Considerations .................................. 83
CONTENTS

5.4.7 Summary ................................................. 83

6 Experimental setup ................................. 85
  6.1 Point of Reference Prototype for DTLS Multicast ........ 85
  6.2 Authentication with ECDSA ............................. 86
    6.2.1 Experimental Packet Size ......................... 87
  6.3 Experimental Scenario ............................... 88
  6.4 Implementation in Hardware ........................... 88
  6.5 Memory footprint ...................................... 90

7 Conclusion and Future Work ......................... 91
  7.1 Future Work ........................................... 93

Bibliography ................................................ 95
Chapter 1

Introduction

The Internet of Things (IoT) is a fast growing area of business and research. Everyday objects become connected to each other and to the Internet via small computing devices attached to these objects. There is a number of large fields where IoT helps to improve our lives, e.g. wearable technologies, smart cities and building management systems. We start relying more on services provided by IoT. Lights in the streets can be remotely managed. In modern houses different systems like heating, ventilation air-conditioning and lights control systems are interconnected into IoT. Hence, the assurance of secure communication for the IoT is vital as well. Usually, IoT devices are small resource-constrained devices. These devices have a high energy constraint and may, often, be battery-powered which results in a low power processor, restricted memory capacity as well as, communicationally, IoT networks have low bandwidth and can suffer from losses.

1.1 State-of-the-Art

Such resource constraints dictate the differences of IoT from the regular Internet. The conventional stack of the Internet protocols is not applicable to the IoT. HyperText Transport Protocol (HTTP) [38] is widely used in the Internet, while in the IoT a Constrained Application Protocol (CoAP) is a widely used
Introduction

A web transfer application layer protocol designed to interface with HTTP for the compatibility with the Internet [80]. Importantly, CoAP is designed for constrained IoT environment, i.e. it is optimized for Machine-to-Machine (M2M), it is simple, implies low overhead and supports IP multicast [80]. Furthermore, HTTP often operates over Transmission Control Protocol (TCP) [38] which is too complex and unaffordable for IoT. Therefore, User Datagram Protocol (UDP) is used for IoT providing the "best effort" transport with small overhead. To provide security for HTTP the Transport Layer Security (TLS) protocol [33] is used which very much relies on TCP flow control mechanism. As far as TCP is not affordable in the IoT, Datagram TLS (DTLS) [77] is used to guarantee pairwise secure transport UDP datagrams. DTLS is a default security protocol for CoAP enabled IoT devices [80]. In order to make all the devices accessible through the Internet, the Internet Protocol version 6 (IPv6) is used as a network layer protocol. Primarily, the link and physical layers are represented by IEEE 802.15.4 standard [2] which has a low bandwidth and a short frame size. The IPv6 over Low power Wireless Personal Area Networks (6LoWPAN) standard is followed to compress the IPv6 and UDP header to increase a prospective per frame application payload [45, 62].

Due to the low bandwidth of the constrained IoT communication environment, support of multicast is indispensable. If numerous recipients need to receive the same message, in order to avoid transmission of the same message to each device, the recipients can form a group so that the sender can physically send a single message to the whole group of devices. Such a transmission is called multicast which results in a considerable saving in energy consumption for the sender device as well as consumption of bandwidth, especially, in the long run. In some IoT applications, multicast also helps to achieve to some extent synchronous processing of a message by the recipients, which is important, when a human is involved in the process. For example, when the lights in a room are turned on as a response to a human action. In our scenario the IP multicast is ensured by the network layer protocol. Since multicast is considered to be important for the IoT, the secure transport of the multicast messages is necessary as well. Recently, the DTLS-based multicast for constrained environments is proposed to support secure multicast for the IoT, which is motivated by the fact that DTLS is already supported to secure pairwise communication for CoAP enabled IoT devices [55]. The DTLS record protocol responsible for secure transmission of data is adapted to support multicast communication [55].
1.2 Problem Statement

DTLS multicast inherits all the security properties of the TLS/DTLS with respect to the group communication. Due to the fact that only symmetric cryptography is supported at the record layer by TLS/DTLS and thus by DTLS multicast, the confidentiality and authenticity for the DTLS secured multicast are assured on a group level. In other words, any of multicast listeners are able to impersonate a multicast sender if a listener wants to, due to the properties of symmetric cryptography. In particular, the same key is used to encrypt and decrypt message as well as to generate and verify Message Authentication Code (MAC). Since the keys are available to all group members, any of them have equal ability to generate a message and temper with the multicast messages as if they were originated from the legitimate sender. Even if the listeners are initially trusted, they could be later compromised by an attacker and the security of the network will be compromised as result, especially, if listener devices are located outdoor and cannot be protected constantly, as in the case of outdoor lighting systems. Therefore, a mechanism to unambiguously authenticate the origin of a multicast message should be provided to minimize consequences of a single multicast listener compromise, even though the DTLS multicast draft explicitly states that such an authentication is not the concern of the document [55]. Such an authentication in the context of multicast is called source authentication [24]. Hence, one of the two main objectives of this project is to enrich the DTLS-based multicast with a solution to provide source authentication applied to a use-case where outdoor lighting system is controlled by a back-end server and the back-end server needs to be authenticated.

Furthermore, handshake protocol used by the DTLS to establish pairwise keys can no longer be used for establishment of keying material for a group communication. The DTLS multicast draft does not cover key management at all, the draft only references that Group Secure Association Key Management Protocol (GSAKMP) can be adapted with any needed modifications for constrained networks [55]. However, key management is considered an important and indispensable part for the real system where the assumption of the pre-shared keys does not hold, since the keys should not only be installed but also managed. Therefore, the second main objective of the project is to design a key management solution for the DTLS multicast at the CoAP enabled IoT.

1.3 Contributions

The contributions of this master thesis are two-fold:
• We enrich DTLS multicast with source authentication scheme to improve security of multicast for the IoT;

• We present a solution for key management for the DTLS multicast with source authentication.

To extend DTLS multicast with source authentication scheme, we survey a number of source authentication schemes and compare them according to the resources that they consume. We present two solutions for the source authentication for DTLS. In particular, the solutions are based on: Elliptic Curve Digital Signature Algorithm (ECDSA) [50] and Timed Efficient Stream Loss-Tolerant Authentication (TESLA) [71]. The integration of the schemes with DTLS multicast is presented considering the application to the constrained environment and, in particular, focusing on the deployment scenario. We find a trade off between security level and the resource consumption. In case of the source authentication with TESLA, we enrich the scheme with the cryptographic primitives to be used for the solution and define the values of the parameters of the scheme suitable for our scenario. We determine a drawback of the TESLA scheme with regards to periodic communication pattern when a sender rarely sends messages out as well as we design an adaptation of the scheme to such a communication pattern. We present a comparison between the two designs according to different scenarios. Finally, we present a sketch of the extension of the DTLS multicast specification to support the selected source authentication.

Furthermore, we formulate the requirements for the key management for the DTLS secure multicast in our IoT scenario. Then, we present a comparison of 4 protocols with regards to the requirements. The best suitable protocol is selected and different options to integrate the protocol in our solution are presented. Afterwards, the protocol is adapted to our solution in order to minimize resource consumption. The protocol is extended with a new payload to suit our flexibility requirement. We propose a key management solution satisfying the requirements.

1.4 Structure of the Thesis

The rest of the thesis is organized as follows. Chapter 2 covers the necessary background information as well as the related work. The system model, requirements to secure multicast in the IoT and the problem statement are presented in Chapter 3. Chapter 4 contains the two designs of the source authentication for DTLS-based multicast and augmented discussion with final selection of the most efficient option. The key management design is presented in Chapter 5.
Chapter 6 shows the implementation of the source authentication with ECDSA for DTLS secure multicast. Finally, the conclusion is drawn in Chapter 7.
This section introduces some background information related to the topic of the thesis. First, we introduce the protocols used by a listener device in our resource-constrained network. Then, we give the background knowledge about DTLS and how the multicast communication is secured with DTLS in the IoT. Next, we introduce the different source authentication schemes taken into consideration as a solution for the extension of the DTLS secured multicast. The table of comparison according to the resource consumption including the communicational and computational overheads is presented as well. We also mention an IPsec multicast solution as a related work which is used for the regular Internet and recently becomes a subject of a standardization effort for constrained environments as well. Finally, we present an overview of the key management protocols widely used for management of keying materials, which also gives an insight into the possible services which can be provided with a key management protocol for secure multicast.
2.1 Resource-constrained IoT networks

Scarcity of the resources in the IoT is one of the primary concerns which is taken into account by the designers of the solution for the IoT. Therefore, the protocols proposed for the IoT are designed with the intention to minimize memory consumption, information sent over the air and computations required from a device and thus, the energy consumption is minimized. These networks tend to be lossy and not powerful, which limits the size of the packet transmittable without fragmentation, therefore, such networks are called Low-power and Lossy Networks (LLNs). The protocols used at the IoT protocol stack can differ but we introduce the ones which are widely used and considered at Philips as well.

The Constrained Application Protocol (CoAP) is a widely used application level protocol for constrained networks which is designed as a web protocol with the aim to minimize energy requirements [80]. The mandatory CoAP header consists of 4 bytes, which is extendable with optional 1 byte length fields, e.g. to represent the type of the content, e.g. Extensible Markup Language (XML), JavaScript Object Notation (JSON) or plaintext, as well as other applications can work on top of CoAP. It was designed to support M2M communication and multicast as described in [74]. CoAP is similar to HTTP with the client-server (request-response) model [80]. The protocol is designed to operate on top of a datagram transport such as UDP. Reliability is optionally supported with the possibility to set confirmable or non-confirmable message delivery, thus with no response arrived within a defined time interval a message is retransmitted. DTLS [77] is considered as a default security mechanism for unicast communication at the CoAP enabled devices. Furthermore, DTLS is extended to secure multicast communication as well [55]. DTLS multicast is explained in detail below. The UDP protocol is used as a "best-effort" transport for the Low-power and Lossy Network (LLN). The IPv6 is used to provide sufficient addressing space for devices. Multicast Protocol for Low power and Lossy Networks (MPL) is used to provide forwarding of the IPv6 multicast packets [44]. The support of IP multicast by CoAP is described in [74]. The IPv6 Routing Protocol for Low power and Lossy Networks (RPL) is the default routing protocol for constrained IP networks [95]. The link layer of such a Wireless Personal Area Network (WPAN) is composed as described in IEEE 802.15.4-2006 and it provides only 127 bytes long frames [2]. Figure 2.1 shows the protocol stack considered in this project as described above.

The transmission of IPv6 and UDP packets over IEEE 802.15.4 [2] is defined by the 6LoWPAN compression mechanism [45, 62]. Such a 6LoWPAN compression helps to considerably minimize the size of the IPv6 and UDP headers. The link layer overhead is 25 bytes at maximum: 23 bytes MAC header and 2 trailing bytes for the Frame Check Sequence (FCS) in the end of a frame [2]. In
general, the addressing fields comprise 20 bytes in total. Source and destination addresses have Personal Area Network (PAN) ID and MAC address, 2 and 8 bytes, respectively. If short addressing is used, then the frame overhead can be reduced to 13 bytes by shortening the MAC addresses to 2 bytes each [48]. If star topology is used, then PAN IDs can be elided [48]. The link layer security mechanism brings 21 bytes of overhead more if it is enabled. With 6LoWPAN compression, the size of the IPv6 header can be compressed from 40 to 7 bytes for the multiple hop transmission and the UDP header can be compressed from 8 to 2 bytes if the source and destination ports are set to the special range (0xf0b0 to 0xf0bf) [45]. Both compressed headers contain a dispatch information to inform about the compression options applied. Hence, considering the best optimization possible, 105 bytes per frame can be left for the protocols on top of UDP if link layer security is disabled. The DTLS record header comprises 13 bytes plus 8 bytes of authentication information if AES-CCM-8 mode [37] is selected. Furthermore, considering that the CoAP header does not have any option, then 80 bytes are left for the CoAP application data. Figure 2.2 shows the packet format as explained above.

If the size of the application data exceeds this maximum, the packet is fragmented at the link layer. However, fragmentation is not desirable in such LLN. Due to the use of UDP instead of TCP, no flow control is provided, in particular,
ordering of the messages is not ensured and messages can arrive corrupted or can be lost, and therefore, it is not efficient to send big packets and fragmentation is not desirable.

The IEEE 802.15.4-2003 and 2006 version is, in general, defined for low-power, low-data-rate and short range networks [2]. It specifies physical (PHY) layer as well as link layer. The data rate achievable with the IEEE 802.15.4-2006 PHY layer is up to 250 kb/s [2]. The IEEE 802.15.4g-2012 presents a set of amendments which allow to achieve data rate up to 400 kb/s with a special modulation as well as to increase the coverage area considerably [3]. The link layer frame size is extended up to 2047 bytes in IEEE 802.15.4g-2012 [3]. However, even considering the longer frame of IEEE 802.15.4g-2012, the pursuit of small message size is still relevant. The smaller the message, the fewer errors occur.

2.1.1 IP Multicast

In general, in the IP multicast there are two types of the nodes involved: a sender node and listeners. The sender initiates multicast traffic and the listeners subscribe to multicast feed. The sender of the multicast selects an IP address to dedicate the packets to from the special range of addresses allocated by the IP protocol specifically for multicast. The listeners who want to join the particular multicast just need to start listening to the particular address. In general, nothing prevent a listener from listening to a particular address or from sending multicast messages to a particular address. The listeners and the sender together form a multicast group. The group membership can change, its members can join and leave [31]. The sender needs to send only one packet and the number of listeners are able to receive it, which is very beneficial for the constrained networks. More details on support of IP multicast by CoAP are provided in [74].

2.1.2 Contiki Operating System

Contiki Operating System (OS) [5] is a widely used OS for the IoT with support for various constrained hardware platforms. Currently, Contiki OS is actively developed by Thingsquare company. Contiki support many protocol used for LLNs such as IPv6, RPL, 6LoWPAN and CoAP [5]. The OS is implemented in C language providing flexible environment for development of various applications. It is used for enabling communication of battery-powered sensors, smart power meters and lighting devices with the Internet.
Moreover, the powerful Cooja simulator is developed for Contiki which allows to simulate hardware properties of distinct devices. The simulator enables network communication of many devices. The signal strength, packet losses and time are also simulated by Cooja. A built in packet sniffer allows to conveniently observe packets sent out by the devices in the simulated network. Software can be compiled to native Cooja target or to any of supported targets, in order to create particular nodes and conduct simulation with the restrictions of the particular hardware.

2.2 Datagram Transport Layer Security

Transport Layer Security (TLS) [33] is a prevailing protocol used to ensure secure transmission of web traffic on the Internet. TLS is used to provide end-to-end security mechanism for pairwise communication. Confidentiality, authenticity and integrity is ensured with TLS. TLS is designed to work over reliable transport such as TCP, and thus, TLS does not deal with the losses and reordering of the packets. However, the growing number of the Internet applications operating over UDP formed the need in TLS over unreliable transport. Hence, Datagram Transport Layer Security (DTLS) is introduced specifying modifications to TLS to operate over unreliable transport, such as UDP, preserving the majority of the TLS mechanisms [77]. DTLS intentionally follows TLS as much as possible to increase code base reuse and avoid unnecessary security innovations [77]. DTLS preserves the two layers similarly to TLS: a handshake layer used to establish a connection with a keying material for two parties and a record layer used for secure data transmission.

There are two major challenges solved by DTLS to adapt TLS to unreliable transport:

1. TLS is not tolerant to losses of records – given a loss of a record, subsequent records are not able to be authenticated;

2. In TLS, handshake layer relies on reliability of the transport protocol, and therefore, does not handle any packet losses.

In order to deal with the first issue, the usage of stream ciphers for the encryption is prohibited at DTLS to enable independent processing of records. Furthermore, to prevent dependency between records during authentication, explicit numbering is included instead of including the record number to the MAC as it is done in TLS. The DTLS record format is represented on Figure 2.3.
Content type field is used to represent the type of the data transmitted in the record: handshake messages or application layer data. Epoch is a counter which is incremented every change of a cipher state. Ciphertext size varies as well as the size of MAC depends on the ciphersuite used.

To provide DTLS handshake with the mechanisms to deal with packet losses, timer and retransmission operation is introduced. Thus, the lost messages are retransmitted when the timer is expired. Subsequently, in order to prevent the reordering problem a sequence number field is introduced to the packet structure. Furthermore, given that the handshake message can exceed the UDP datagram size, fragmentation is designed to split the handshake message into several DTLS records which fit to a single datagram.

Moreover, the DoS countermeasure is added to DTLS for an attack where an attacker sends \texttt{ClientHello} message to the server with the tampered source address of a client. As a result of such an attack, the server will overwhelm the innocent client with the messages from the server. To secure against this attack, the server sends a \texttt{HelloVerifyRequest} which contains generated by the server stateless cookie and asks to repeat \texttt{ClientHello} with the cookie. The assumption is that it is difficult for an attacker to receive the cookie if the IP address was spoofed. The handshake protocol message exchanges are shown in Figure 2.4.

The bitmap window mechanism to detect replay attacks is borrowed from IPsec \cite{77}. The packets with sequence numbers out of this window or already received ones are discarded. The discussed features are summarized as number of bullets pointing out differences of DTLS from TLS:

- Sequence number and epoch fields at the record layer header;
- Prohibition to use stream ciphers;
- Timer and retransmission at the handshake layer;
- Sequence number field at the handshake layer;
- DoS countermeasure: stateless cookie at the handshake layer;
- Optional replay protection with a bitmap window.
During the handshake, the client sends a `ClientHello`, gets a reply from the server containing the cookie and repeats a `ClientHello` message containing the cookie. Then, the server sends a `ServerHello` message and optionally: a certificate and a keying material. Next, the client replies with the client’s keying material, optionally with the client’s certificate and notifies about the end of the handshake. Finally, the server finishes the handshake as well.

### 2.2.1 DTLS Secured Multicast

DTLS is extended to secure multicast in LLNs, since CoAP enabled devices support DTLS as a compulsory security mechanism [55]. There are few changes introduced to DTLS to provide secure guaranties to multicast communication. Since the multicast is performed at network level, the DTLS multicast draft does not change any multicast functionality but modifies the DTLS specification instead. DTLS multicast draft supports multicast communication with one sender and many listeners (1-to-M) and with many senders and many listeners (N-to-M).

DTLS-based multicast draft repurposes the use of the security parameters to derive the same keying material for all the devices in the group. Secondly, the default ciphersuit considered to be used by DTLS for constrained networks is Advanced Encryption Standard (AES) Counter with CBC-MAC (AES-CCM) mode [37, 59]. The AES-CCM is a mode where a single key is used for encryption and authentication. Encryption is provided with AES in Counter Mode (CM) and authentication is ensured with Cipher Block Chaining (CBC) Message Authentication Code (MAC) [59]. The CCM mode as other Counter Modes
Background and Related Work

is susceptible to nonce reuse [37, 59]. If a nonce value is reused for encryption with a given key, confidentiality can be compromised. In DTLS with AES-CCM mode, sequence number ensures that the nonce is not reused, however, in multicast with the N-to-M communication several senders within a group might have the same sequence numbers. Thus, security of the multicast can be compromised. Hence, the DTLS-based multicast proposes a unique sender ID for each sender in the same group which is represented by a single byte taken from a sequence number as shown in Figure 2.5.

<table>
<thead>
<tr>
<th>Content type</th>
<th>Version</th>
<th>Epoch</th>
<th>Sender ID</th>
<th>Seq_number</th>
<th>Length</th>
<th>Ciphertext</th>
<th>MAC</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Byte</td>
<td>Ma</td>
<td>Mi</td>
<td>2 Byte</td>
<td>1 Byte</td>
<td>5 Byte</td>
<td>2 Byte</td>
<td></td>
</tr>
</tbody>
</table>

Figure 2.5: DTLS multicast record format

2.3 Source Authentication Schemes

Authenticity of the data is an important requirement for secure multicast even in Low-Power and Lossy Networks. However, the mechanisms to achieve data origin authenticity for secure multicast are more complex than the ones for unicast transmission. If group members are not trusted, then receivers of a multicast message need to authenticate that the message has originated from the sender and not from any other group member. Thus, simple and fast solution used for unicast based on shared within the group authentication key used to generate a MAC does not achieve origin authentication (source authentication) in case of multicast. Therefore, in this section we give an overview of different source authentication mechanisms to be used for multicast authentication in constrained environments. In this section, we give some numbers in regards to the performance of the different scheme implementations. However, the performance numbers can vary a lot depending on the optimization goal preferences (RAM, ROM, speed) taken into consideration during implementation, and therefore, performance of different implementations of the same algorithm is different. Depending on the target platform for the implementation, the code size can also differ in the range on several kilobytes. Thus, the numbers summarized are only informational to show the range of resources required. Moreover, we mostly summarize the numbers for 80 bit security level solutions, since there is a lot of material on the Internet in regards to the source authentication schemes performance.
2.3 Source Authentication Schemes

2.3.1 Elliptic Curve Digital Signature Algorithm

The Elliptic Curve Digital Signature Algorithm (ECDSA) [50] is a Digital Signature Algorithm (DSA) version based on elliptic curve cryptography. The ECDSA security is based on Elliptic Curve Discrete Logarithm Problem (ECDLP), while the DSA security relies on Discrete Logarithm Problem (DLP). The ECDLP is harder than the DLP, and therefore, the length of keys for ECDSA is significantly smaller than the length of keys for DSA [50]. For example, the ECDSA keys are 160 bits for 80-bit security solution, while DSA keys have to be 1024 bits to provide the same level of security. The smaller keys result in faster computation and smaller certificates. In fact, ECDSA is used to sign only a hash of a message instead of signing the whole message, thus, a hash algorithm is used, e.g. Secure Hash Algorithm (SHA), SHA-2 family. The hash is selected in a way that the length of a hash is equal to the length of keys, otherwise, only the least significant bits are used where the number of the bits is equal to the key length. All the computation is done on a finite field which can be binary or prime.

Prior to use ECDSA both parties (signature issuer and signature verifier) should agree on certain parameters, such as a particular curve (curve coefficients \(a\) and \(b\)), number of points on a curve \(n\), a point of reference \(G\) and a value of the cofactor \(h\) parameter. The signature issuers generate a private key which is in the interval \([0..n − 1]\) and it can be generated randomly or pseudo-randomly. The public key is generated by point multiplication of a private key and a point of reference \(G\). Hence, the value of public key is tightly dependent on the private key and cannot be generated independently. A signature contains two parts: \(R\) and \(S\), each part has a size equivalent to the key length. In order to generate signature, the signature issuer calculates \(R\) value using the point of reference \(G\), and then, \(S\) value is calculated from \(R\) and a MAC of the message to be signed. Verification process involves MAC, both values \(R\) and \(S\) and the public key of a signer. Importantly, ECDSA algorithm is approved by the National Institute of Standards and Technology (NIST) [64].

Liu et al. implementation of ECDSA with 160 bits key at TinyECC library takes 2 s to sign a message and 2.4 s to verify a message with RAM and ROM usage 1.4 and 18 KB, respectively, on MICAz micro-controller with about 8 MHz ATmega128 8-bit processors [56]. Oliveira et al. achieve signature generation with ECDSA-160 on prime curve in 40 ms on ARM7 processor, however, the code size is about 49 KB [66]. On the MICAz Motes, signature generation is 0.68s in prime curve in 0.68s with 37 KB of ROM and 1.5 KB of RAM required [65].
2.3.2 Timed Efficient Stream Loss-tolerant Authentication

Perrig et al. proposed *Timed Efficient Stream Loss-tolerant Authentication (TESLA)* [71], a scheme based on symmetric cryptography achieving asymmetric properties. TESLA provides low communicational, low computational overhead and robustness to packet losses. The idea behind the scheme is that multicast receivers are loosely time synchronized with the sender where the time synchronization is in the range of the round trip time. The scheme used to achieve time synchronization involves sender signing a time stamp before sending it to the receiver. The time synchronization mechanism is beyond the scope of our project and the detailed description can be found in [71, 69]. The asymmetry is achieved by the sender revealing an authentication key with a certain key disclosure delay after the messages authenticated with this key supposedly arrived to receivers. So that none of the group members can generate a message using the same authentication key as the sender before the key is revealed. The delayed authentication requires receivers to buffer massages awaiting authentication. One-way key chain is used to generate authentication keys at the sender. A chain of values, each of which is used to derive a corresponding key, is generated with a cryptographic one-way function $F$. The first input value $s_{\text{init}}$ to the function $F$ is securely generated with some Pseudo Random Number Generator (PRNG). The function $F$ is applied iteratively to the output value of itself where each subsistent output of the function is a value used to derive a key. The generated values are used in reverse order, the last generated is used first. Hence, the currently revealed value $s_i$ can be authenticated at the receiver with the previously revealed one $s_{i-1}$ in a way that $s_{i-1} = F(s_i)$. A function used to derive a key from the one-way chain value is another one-way function $F'$, hence a key $K_i$ can be derived as follows: $K_i = F'(s_i)$. In case of packet losses, a missing authentication key can be reconstructed from the next received one-way chain value. Given the time synchronization with the sender, a receiver knows that $n$ intervals were lapsed between the missing and received keys $K_j$ and $K_i$, respectively: $n = i - j$. Thus, the value $s_j$ corresponding to $K_j$ can be recovered recursively applying the $F$ function $i - j$ times: $s_j = F^{i-j}(s_i)$. The key used for authentication of messages at interval $i$ is revealed with the messages at interval $i + d$, where $d$ is a key disclosure delay. Depending on the length of a time interval a variable number of messages can be authenticated with a key. Figure 2.6 shows the protocol as discussed above.

Furthermore, Perrig et al. propose $\mu$TESLA [72] which is a lightweight variant of TESLA. $\mu$TESLA targets very constrained devices, e.g. $\mu$TESLA authors conducted experiments on the devices with 8 bit CPU with 4 MHz frequency [72], and assumes further minimization of communicational and computational expenses as well as memory and energy requirements. Firstly, TESLA considers an
 authentication of bootstrapping either by a digital signature or by a pre-shared symmetric key, while \( \mu \text{TESLA} \) is restricted to the symmetric cryptography only. The assumption is that all \( n \) receivers share a pairwise authentication key with the multicast sender and the sender is supposed to store \( n \) pairwise keys. These keys are used to authentically bootstrap receivers, i.e. to perform time synchronization and to distribute an initial key chain value as well as disclosure schedule parameters. Next, in \( \mu \text{TESLA} \), releasing an authentication key for each message is considered too expensive in terms of communication overhead, thus in \( \mu \text{TESLA} \) a key is disclosed once per several interval. Perrig et al. show the numbers achieved for the \( \mu \text{TESLA} \) implementation evolution [72]. The target device has 8 bit CPU with 4 MHz frequency. The 16 byte long MAC is generated with AES-CBC-MAC operation mode and requires about 1.5 ms which varies slightly depending on the target optimization (Speed or ROM) [72]. \( \mu \text{TESLA} \) implementation took 574 B of memory and together with cryptographic libraries it comprised 2 KB [72].

Liu and Ning [57] extend the \( \mu \text{TESLA} \) scheme to increase scalability in case of late multiple joins of nodes. \( \mu \text{TESLA} \) relies on unicast for distribution of initial parameters from base station to joining nodes. Liu and Ning propose a multilevel key chains for extension for \( \mu \text{TESLA} \) allowing to distribute the initial parameters over multicast. Such efficiency is achieved by creating a separate key chain, called high-level chain, with longer intervals, where an initial commitment to this chain is pre-installed at every receiver device. A high-level chain is used to broadcast an initial commitment of the low-level chain which is used for authentication of frequent multicast messages. Furthermore, low-level chain a considered to be short, so that multiple chains are lapsed during the life time of a device. Thus, high-level chain is used for distributing new commitment for a low-level chain.

Figure 2.6: TESLA protocol
2.3.3 Other Schemes

Proposed by Perrig et al. [70], BiBa signature is a one-time signature scheme based on one-way functions, it has small overhead on verification and relatively small size of the signature [70]. In the BiBa signature, a signer precomputes many values which can be easily authenticated using public key but it is computationally infeasible to find these values given a public key. BiBa signature generation requires 5 times less time than generation of RSA signature, while verification of a signature requires 20 times less time than does RSA signature verification [70]. For 80 bit security solution, signature size is 128 bytes and the public key is about 10 KB [70]. Hence, the long public key of BiBa implies high communicational overhead and high memory requirements to store such keys.

Ren et al. [76] summarize drawbacks of symmetric cryptography based TESLA like authentication schemes and present an efficient, suitable for wireless sensor networks public key cryptography (PKC) based solution. Ren et al. [76] point out that TESLA-like systems are vulnerable to a wormhole attack due to the propagation delay of the authentication key transmission. Moreover, an adversary can trick receivers to store bogus packets, since the authentication is delayed. As a result, the attacker can overwhelm the network and cause energy-depletion at the receivers. As an alternative to the TESLA-like scheme, Ren et al. [76] propose public key Bloom filter-based Authentication Scheme (BAS) with partial message recovery and another scheme which extends BAS with Merkel hash trees in order to increase the number of users [76]. Computational and communication overhead parameters of the proposed mechanisms are comparable with ones achieved utilizing ECDSA [76]. In particular, BAS computational overhead is more than ECDSA, and signature verification requires $m/N\ln 2$ hash operations and one ECDSA signature verification, where $m$ is a message length and $N$ is a number of nodes. The communicational overhead is smaller than ECDSA. Even though the signature is of the same size, only a part of the message is transmitted and 10 bytes chunk is recovered from the signature.

Shim et al. [82] present an Efficient Identity based Broadcast Authentication Solution (EIBAS) for wireless sensor networks to reduce communication overhead, and therefore, energy consumption. The argument is that the computational capabilities of sensors increase way faster than communication data rate. The solution is based on ID based signature with message recovery proposed by Tso et al. [87]. The signature is generated on a paring friendly supersingular curve. The 80 bit security scheme requires 70 bytes for signature, it also provides partial message recovery. The signature is calculated on MICA2 Mote with ATMega128L processor over 2.8 s. However, EIBAS requires each node to store the identities together with public keys of other nodes in the network.
Zhang et al. [97] present a *Polynomial Based Authentication (PBA)* for lightweight sensor networks. The scheme assumes that all the nodes are preloaded with special unique identifiers (ID). It is assumed that the sender ID is known to the receivers. A sender is preloaded with authentication polynomial while receivers are preloaded with verification polynomial. All of the polynomials are distinct. In this scheme, the sender before transmission of a message compute a Message Authentication Function (MAF) to be transferred with the message, and then, the message is transferred. MAF is a polynomial used instead of a signature for authentication and it is derived from the authentication polynomial, hash of the message and the sender’s ID. A receiver instantiates the value of the polynomial variable with its own ID value, and then, solves it. Then, the receiver derives another value utilizing the verification polynomial, the hash of the received message and the ID of the sender. Next, the receiver evaluates both products, and based on the difference between the products, the decision to accept or to reject the message is made. This scheme scales towards a higher number of the nodes by increasing the length of the polynomials and other parameters not covered in this overview. With the increasing length of the polynomials the overhead parameters grow respectively. Hence, the Table 2.1 summarizes characteristics corresponding to $2^{15}$ number of nodes. This scale is relevant to our case, since current DTLS multicast internet draft is limited to $2^{15}$ nodes in case of multiple senders scenario [55]. The PBA scheme has low computational and communication overhead. The solution with $2^{15}$ nodes limit requires 6.4 ms to compute a MAF polynomial, 57 ms to verify that MAF polynomial is correct where the numbers are acquired from the MICA2 with ATmega128L [97]. However, the security properties of such schemes are not provable and in particular this scheme has been broken [7].

Bohen et al. [20] propose *Boneh–Lynn–Shacham (BLS) short signature* scheme based on bilinear pairing. The idea behind pairing is to construct a mapping between two cryptographic groups which enables new cryptographic schemes. In BLS scheme bilinear pairing is performed on gap Diffie-Hellman group on special elliptic curves over a finite field. The gap Diffie-Hellman group is a group where computational Diffie-Hellman problem is intractable, while decisional Diffie-Hellman problem is solvable [20]. The signature generation operation requires a multiplication operation on a curve, while verification is performed by a bilinear pairing operation. The achievable length of the signature is equal to 161 bits [65]. Despite the short length of a signature a scheme is proven to be secure considering chosen-message attacks [20]. The BLS scheme requires a special probabilistic hash function to perform map to point operation ($\{1, 0\}^* \rightarrow G_1$). The signature can be generated in about as twice much time as required for ECDSA signature generation [65], while verification of the signature takes considerably longer time, 2.9 s are needed for 80 bit security level signature verification on 1 GHz Pentium III [20].
Zhang et al. [96] propose Zhang–Safavi-Naini–Sisilo (ZSS) short signature which is also pairing based signature similar to the BLS one. However, since pairing operations are the most expensive ones in such cryptographic systems, ZSS is constructed in a way to utilise fewer pairing operations [96]. The distinguishable feature of ZSS is the optimization towards lower computational overhead compared to BLS, while keeping the signature size the same short length. The scheme works with general hash function (such as SHA-1) which is more efficient than the probabilistic ones. Verification of the signature requires one pairing and one squaring against two pairings in case of BLS [65].

Oliveira et al. [65] compare the efficiency of BLS short signature [20], ZSS short signature [96], ECDSA [50] and Schnorr. Schnorr is a patented (U.S. Patent 4,995,082) simplest variant of ECDSA which generates signature without multiplication inverse. However, the Schnorr patent has expired.

### 2.3.4 Summary

Table 2.1 presents a summary outlining overhead characteristics of the broadcast source authentication approaches discussed above. However, the computational overhead numbers vary a lot depending on the particular implementation. Depending on a hardware limitations and system requirements a scheme can be programmed pursuing memory or processing time minimization goal. Often, both optimizations are important and in this case a suitable trade off between the required memory and the processing time is found.

### 2.4 Key Management Schemes

In this section we present an overview of the widely used key management protocols designed for the key management in multicast groups.

#### 2.4.1 Group Secure Association Key Management Protocol

The Group Secure Association Key Management Protocol (GSAKMP) [42] is designed to distribute and manage group security policies and SA for multicast group secure communication as well as to enforce that all the group member follow the policies. The protocol supports large groups with one or many senders.
<table>
<thead>
<tr>
<th>Scheme</th>
<th>Crypto</th>
<th>Comp. overhead</th>
<th>Comm. overhead (B)</th>
<th>Memory (KB)</th>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>TESLA [71]</td>
<td>Symmetric</td>
<td>Low (3 MAC / 4.5 ms on*)</td>
<td>Low (MAC + key)</td>
<td>RAM 0.14</td>
<td>-</td>
<td>Lowest overheads.</td>
</tr>
<tr>
<td>μTESLA [72]</td>
<td>Symmetric</td>
<td>Low (3 MAC / 4.5 ms on*)</td>
<td>Low (MAC)</td>
<td>ROM 2</td>
<td>-</td>
<td>Low comp.</td>
</tr>
<tr>
<td>BIBA [70]</td>
<td>One-time sig.</td>
<td>Low (gen. ~ RSA/5, ver. ~ RSA/20)</td>
<td>High (128)</td>
<td>RAM -</td>
<td>-</td>
<td>Lowest overheads.</td>
</tr>
<tr>
<td>ECDSA in [60]</td>
<td>PKC ECC prime</td>
<td>Mid (sig. 2s, ver. 2.4s on**)</td>
<td>Mid (40)</td>
<td>RAM 1.4</td>
<td>18</td>
<td>Mid comm., comp.</td>
</tr>
<tr>
<td>Schnorr in [60]</td>
<td>PKC ECC prime</td>
<td>Mid (sig. 0.68s on**)</td>
<td>Mid (40)</td>
<td>RAM 1.4</td>
<td>37</td>
<td>Mid comm., comp.</td>
</tr>
<tr>
<td>BAS [76]</td>
<td>PKC</td>
<td>Mid (ver. m/N ln 2 + ECDSA)</td>
<td>Low (30)</td>
<td>RAM -</td>
<td>-</td>
<td>Low comm.</td>
</tr>
<tr>
<td>EIBAS [62]</td>
<td>IDBS ECC binary</td>
<td>High (sig. 2.28s / on**+)</td>
<td>Mid (10)</td>
<td>RAM 2.6</td>
<td>22</td>
<td>Mid comm., comp.</td>
</tr>
<tr>
<td>PBA [97] (broken)</td>
<td>Polynomial</td>
<td>Low (sig. 0.6ms, ver. 2ms on**+)</td>
<td>Low (16)</td>
<td>RAM 1.2</td>
<td>37.6</td>
<td>Low comm., comp.</td>
</tr>
<tr>
<td>BLS in [60]</td>
<td>ECC pairing prime</td>
<td>High (sig. 0.18s / verif. expensive)</td>
<td>Low (20)</td>
<td>RAM 2</td>
<td>38</td>
<td>Low comm., comp.</td>
</tr>
<tr>
<td>ZSS in [60]</td>
<td>ECC pairing prime</td>
<td>High (sig. 0.78s / verif. expensive)</td>
<td>Low (20)</td>
<td>RAM 2</td>
<td>38</td>
<td>Low comm., comp.</td>
</tr>
</tbody>
</table>

Table 2.1: Source authentication schemes comparison

All the numbers are given for 80-bit security level versions of the schemes, the target platforms are denoted below, while all of them are 8 bit micro-controllers with 4 and 8 MHz frequencies. Verification of pairing based BLS and ZSS signatures is much more expensive than signature generation, e.g. verification of BLS signature requires 2.9 s on 1 GHz Pentium III for 80 bit security level [20].

Notation:
PKC - Public Key Cryptography
ECC - Elliptic Curve Cryptography
IDBS - Identity-Based Signature
* - SmartDust node CPU 8 bit, 4 MHz frequency
** - MICAz Mote with ATMega128 8-bit micro-controller with 8 MHz frequency
*** - MICA2 Mote with ATMega128L 8-bit micro-controller with 8 MHz frequency
GSAKMP is based on Internet Security Association and Key Management Protocol (ISAKMP) [58]. ISAKMP [58] provides a framework to pairwise security association management. ISAKMP offers two phases for security association (SA) negotiations. Phase 1 is used to achieve mutual authentication and to establish an ISAKMP SA (Phase 1 SA) to be used for protection of Phase 2 negotiation. Phase 2 is used to negotiate a SA for any protocol other than ISAKMP.

The functionality of the protocol is distributed among 4 entities: Group Owner (GO), Group Member (GM), Group Controller Key Server (GCKS), Subordinate-GCKS (S-GCKS). GO is responsible for creation and amendment of the group policy which identifies the GO and the group itself, defines who can access the group, who can manage the group as well as it defines the security mechanisms to be utilized for the protection of communication. GM has to follow the group policy and ensure that the GCKS managing the group is an authorized party. GCKS is responsible for generation and distribution of a Group Traffic Protection Key (GTPK). GCKS performs rekey operations as well as group membership management. S-GCKS is a GM who assigned to perform the responsibility of the GCKS for a sub-group. S-GCKS is only not allowed to generate a GTPK. S-GCKSs are introduced to achieve a distributed model of group management in large groups.

The protocol supposes utilization of a Logical Key Hierarchy (LKH) [90] or a similar method [81, 22] for forward and backward access control. Hence, GCKS manages the group with key array which can be done over multicast. If S-GCKSs take place, then GCKS manages an LKH for secure communication with S-GCKSs and each S-GCKS has a separate LKH for the sub-group it responsible for.

In order to form a group, a GO creates a policy token containing a group policy. GO signs the policy token and sends it to the GCKS. Through a discovery service potential group members are informed about the new group. A join operation is pulled by a potential GM willing to join the group. Figure 2.7 shows the message exchanges sequence between GCKS and GM. First, the GM requests a registration for the group from GCKS. If the GM is authorized to join the group, then GCKS sends back a key download message containing a keying material including GTPK and policy token signed by GO. Upon the receipt of the key download message, the GM checks if the GCKS is the authorized GCKS, and the policy token is legitimate (signed with the right GO). The GM also checks in the policy if it is allowed to be S-GCKS, if yes, it takes responsibility of S-GCKS. Then, the GM sends the Ack/False message informing the GCMS if the key download message was processed correctly. Each of three messages is signed by the initiating party. GSAKMP supports DSS, RSA-1024 and ECDSA-384 signature algorithms [42]. There are two optional messages which can be
sent by the GCKS: a join error message and a Lack of Ack message. As well, there is a possibility for protection from DOS attack via two additional message exchanges where GM needs to include a cookie downloaded from GCKS for the group registration request.

Figure 2.7: GSAKMP’s join and rekey message exchanges

In regards to the rekey operation, GSAKMP considers several distinct cases, such as key update due to life time expiration, eviction operation, and de-registration when a GM is willing to follow the protocol to be de-registered from the group. From the viewpoint of our project, the key update and member eviction appeal more to the context, thus they are considered. Both of the operations are pushed from GCKS to GMs and comprise half round trip, i.e. a single message transmission. If the group key is updated, then policy token is not necessary transmitted but only the key is updated with a rekey event message. If a GM is evicted, then the policy is updated and it is sent along with the rekey event message where the GM to be evicted is not included into LKH anymore.

Each mandatory message in GSAKMP is signed including the messages originated from a group member. Signature payload includes time-stamp to indicate the generation time.

2.4.2 Group Domain of Interpretation

The Group Domain of Interpretation (GDOI) protocol [93] facilitates group security associations management for secure multicast environments. The main application of GDOI is IPsec based multicast [92]. The protocol is utilized
GDOI is built upon the ISAKMP protocol [58] providing the group policy and key management based on the mechanisms used at ISAKMP for pairwise key management. GDOI reuses the Phase 1 of ISAKMP as described in [41]. Even though any protocol can be used to establish the Phase 1 SA, a special cookie should be established after Phase 1. The cookie is later used at Phase 2 to identify a particular session. At the Phase 2, two special exchanges GROUPKEY-PULL and GROUPKEY-PUSH are used at GDOI which make possible group security association management, i.e. group key management.

The GROUPKEY-PULL exchange is initiated by group member who wants to join or perform rekey operation as Figure 2.8 shows. It consists of 4 exchanges in total: 2 messages are initiated by the group member and other 2 are initiated by group control key server. Each message involves nonce transmission to detect replay attacks. After GROUPKEY-PULL exchange, a successfully joined node should have received a Key Encryption Key (KEK) and, possibly, Traffic Encryption Keys (TEKs) conforming to security policies. KEK is linked to Rekey SA and it is used to protect GROUPKEY-PUSH exchange. The GROUPKEY-PULL exchanges are protected with the Phase 1 SA.

With the first message of GROUPKEY-PULL exchange, a group member informs the group controller key server (GCKS) which group it wants to join. The second message is sent by the GCKS informing about cryptographic policies, such as security protocols used by the group. If the group member agrees on the policy, then it proceeds with the third message. The third message is an acknowledgment from the group member indicating acceptance of the group policies. This message can bear a request of group senders IDs. The forth message is initiated by GCKS and it contains the group keying material. This message may encapsulate as many as needed distinct keys.
2.4 Key Management Schemes

The GROUPKEY-PUSH exchange is initiated by the GCKS and it provides a half round trip mechanism to rekey as Figure 2.8 shows. Rekey operation is usually pushed over multicast address. With this GROUPKEY-PUSH exchange, the GCKS can either update keys or evict some of the group members from the group following the LKH mechanism [90]. If LKH is used, then rekey message contains a batch of keys to update the hierarchy of the keys. The GROUPKEY-PUSH mechanism achieves forward and backward access control by prohibiting a previous group member from reading new communication and by restricting a newly joint group member from getting access to an old key, respectively [93]. A GROUPKEY-PUSH contains security policy updates for the updated. The whole message is signed and the signature payload is appended in the end of the message. GDOI supports several signature algorithms, such as ECDSA-256, RSA, DSS, etc.

2.4.3 Multimedia Internet Keying

Multimedia Internet KEYing (MIKEY) [10] is a key management protocol originally designed to provide key distribution mechanisms for secure real-time multimedia application, in particular, Real Time Transport Protocol (RTP) [79] applications secured with Secure Real Time Transport (SRTP) [16]. MIKEY is supposed to primarily be used in peer-to-peer, one-to-many group communication or small groups with many-to-many group communication pattern. The protocol aims at simplicity, efficiency and flexibility as design objectives. It is designed to achieve low bandwidth consumption, low computational overhead, small code size and independence from the underlying transport protocol [10].

The protocol provides several ways of secure key establishment such as: using a pre-shared key, agreeing on a key through public-key encryption or following Diffie-Hellman key exchange. The Diffie-Hellman key exchange can be protected either with symmetric key encryption or with digital signature mechanism. RSA algorithm is intended to be used when public key encryption is selected. All 3 modes consist of two message exchanges. At the pre-shared key and public encryption modes, the keying material and security parameters are pushed by GCKS to the recipients. The second message in these modes, which is sent by a responder, is mainly used for achieving mutual authentication. The GCKS can choose if the second message is needed by setting up an appropriate flag in the first message header. Thus, these key establishment methods apply different communicational cost, as well as computational expenses are different. The pre-shared key mode is the most efficient both computationally and communicaionally. The flexible design of MIKEY is also affirmed by the number of provided extensions [25, 49].
The MIKEY protocol is used to establish Traffic Encryption Key (TEK) which is supposed to be derived from TEK Generation Key (TGK). The parties can agree on TGK using one of the methods mentioned above. A TGK can be used to derive multiple keys for several security protocol sessions. Furthermore, flexibility is achieved in a way that MIKEY can carry different payload types which perform distinct functions, as well as one packet may carry several distinct payloads. Each payload has a Next payload field and a field denoting the length of the payload. There is also a possibility to sign a packet with RSA signature. The Public Key Infrastructure (PKI) is not defined by MIKEY but PKI X.509 [43] is referenced to provide PKI services.

2.4.4 Adapted Multimedia Internet Keying

The Adapted Multimedia Internet KEYing (AMIKEY) protocol is designed with intention to modify the MIKEY [10] protocol in such a way that it can be used not only for real-time multimedia applications but also in the domain of Low-power and Lossy Networks (LLNs). The motivation behind the adaptation of MIKEY to LLNs is that MIKEY is efficient, simple, flexible and easily extendable. The fact that MIKEY is widely used, evaluated and repeatedly tested protects from reinventing new protocols and assuring its flawlessness.

AMIKEY reuses the crypto sessions structure but repurposes them from establishing session keys to establishing long term keys for distinct communication protocol layers on a device, e.g. routing level security for RPL and link layer security. AMIKEY requires a maximum of 2 exchanges for key management signaling and a minimum of 1 exchange. However, in AMIKEY both types of pull and push approaches are possible, while MIKEY allows only a GCKS to push keys to GMs. A (prospective) GM can act as a requester or responder, respectively, and GCKS is denoted as an initiator [10]. In AMIKEY following pull approach, a GM acts like a requester initiating a request message and gets the keying material with a message from the GCKS. If pull approach is used, then a key exchange always takes 2 exchanges as it is shown in Figure 2.9. If the GCKS pushes the key update, then a group member can be optionally required to send a response in order to achieve mutual authentication with GCKS as it is shown in Figure 2.9. AMIKEY also introduces several efficient cryptographic algorithms to be used for encryption and authentication like AES-CCM and AES-CBC-MAC. AMIKEY defines two new payloads to work with RPL: key index and key source identifier.

The GM's request and response messages preserve the same format. A message contains Common header, Timestamp, Verification and, optionally, Identification payload. The Common header payload bears the IDs of the security
2.4 Key Management Schemes

Figure 2.9: AMIKEY’s join and rekey message exchanges

protocols for which the keys should be established. The Timestamp payload is used to transmit a timestamp or a counter value for a replay protection. The Verification payload contains a MAC calculated on the whole message. The Identification payload is optionally used to provide an opportunity to select the correct key shared with a given GM in this case. Figure 2.10 shows the request packet format.

![AMIKEY GM's message packet format]

The GCKS message includes Common header, Timestamp, RAND, one or more Security Policy (SP) payloads, Key data transport payload (KEMAC) and, optionally, Identification payloads. Figure 2.11 shows the format of the packet. The RAND payload is only included in the messages sent by the server and it is used to transmit a random value to introduce freshness to the keys derived from a TGK [10]. It is used as a countermeasure to the offline pre-computation attacks [10]. The random value is transmitted only with an initial message from the server, i.e. during the join operation. Then, during key update operations, the same random value is reused. The Security Policy payload bears the security policies defined by the GCKS with regards to the keying material.
distributed with KE MAC payload. Several security policies can be included into a packet, they are linked to the specific protocol by the policy number. The KE MAC payload can also include several keys, each key is represented by the Key data sub-payload. Even though, there are more payloads described in the RFC3830 [10], we discuss only payloads which could be used for our needs: group key management and public key distribution.

2.5 Related Work

In this project, DTLS is considered as a main protocol to ensure security of unicast and multicast communication, however, there are other solutions to provide the end-to-end security which are discussed in this section. The Internet Protocol Security (IPsec) [54] is a widely used secure architecture to provide point-to-point, site-to-site or point-to-site secure channel on the Internet. IPsec contains two protocols to ensure security, namely: Authentication Header (AH) [52] and the Encapsulating Security Payload (ESP) [53]. The AH protocol can be used to provide authenticity for end-to-end communication and the ESP protocol is used to provide confidentiality and authenticity. The Internet Key Exchange Protocol Version 2 (IKEv2) is suggested as a key management protocol for IPsec [51]. The secure tunneling provided with IPsec is deployed on top of IP protocol, and hence, IPsec is transparent for the protocols on top including transport layer, i.e. payload of the IP protocol is protected and not available to the devices in the network on the way between two end-points. On the one hand, this feature is very convenient since application layer protocols do not need to care about security, unlike CoAP which describes DTLS to ensure security. On the other hand, this feature causes problems for on path services, e.g. for Network Address Translation (NAT) services, and therefore, the IPsec protected payload can be encapsulated to the UDP packet to avoid problems with devices providing on path services. Moreover, IPsec is successfully extended to support multicast communication.

2.5.1 IPsec-based Secure Multicast

The Multicast extension to IPsec provides an optional extension for the current IPsec standard [92]. Hence, the security mechanisms are to be applied to IP multicast traffic. The destination unicast IP address is substituted with the multicast IP address. Furthermore, the extension suggests to share a security association within a multicast group, and thus, it enables group members to perform cryptographic operations. Confidentiality is ensured with a shared en-
### Figure 2.11: AMIKEY message sent by a GCKS

The payloads depicted below KEMAC payload are encapsulated in other payloads. CS ID map info is a sub-payload of the Common header payload. SP TLV param shows the representation the Policy param field of the Security Policy payload. The Key data sub-payload is encapsulated at KEMAC payload.
cryption key. Authenticity and integrity on a group level are also provided with a shared authentication key, whereas source authenticity is suggested to be provided with the TESLA protocol [69], a digital signature or other mechanism. Moreover, group key management for IPsec multicast can be done with GDOI protocol as it is performed with Cisco secure multicast solution [27].

2.5.2 IPsec for IoT

With the development of IP IoT, IPsec becomes interesting for the IoT as well, however the overheads of IPsec are not acceptable for the constrained IoT devices [60]. Hence, recently in 2014 the Minimal ESP draft version 00 was proposed to specify minimal setting for ESP protocol to minimize overheads [61]. Furthermore, the Diet-ESP draft is also proposed this year to provide compression of mechanism of the ESP header to further minimize communicational overhead [60]. Diet-ESP proposes to use existing IKEv2 protocol to agree on the compression parameters. Compression of the ESP fields ensures that all the fields are as reasonably small as it is possible. The draft also assumes that the IoT constrained devices can be connected to a security gateway which enables interoperation of different IPsec versions, and thus, these devices will be accessible from the Internet [60]. Prospectively, this solution of IPsec for IoT can be used as a basics for the IPsec based secure multicast in the IoT, since the classical version of the IPsec supports multicast.
In this chapter we give motivation for the secure multicast for the Internet of Things with the use-cases where secure multicast with source authentication is an important feature. Furthermore, we present the requirements applied by Philips to secure multicast based on the deployment scenario. Next, we analyse the current version of the DTLS-based multicast draft [55]. Then, we outline the requirements which are not covered by the current DTLS-based multicast draft as well as we describe our security model. Finally, based on the not covered requirements (desired but missing features) we draw a problem definition and goals to be achieved with this project.

This master thesis is concerned with secure multicast communication for the constrained networks where CoAP [80] is considered as a primary communication protocol. Currently, the CoAP protocol supports multicast and secure unicast over DTLS [77].

Multicast is commonly required for constrained networks to facilitate an increase in efficiency and scalability of the networks, since in the Internet of Things the same message is often to be distributed to many nodes in the network. Due to the limited bandwidth, transmission of multiple duplicates of a single message is not affordable. In other cases, multicast is used to achieve synchronization
between devices, e.g. in lightning control systems where multicast is used to avoid a distinguishable by a human eye desynchronization. In other cases, a smart meter device might want to distribute the same information, e.g. the rates for electricity or to adjust a consumption at the devices it gathers information from. Thus, multicast is a must in such constrained environments. Furthermore, the information distribution over multicast is supposed to be performed in a secure way, e.g. lighting control commands or commissioning of the lighting devices by a back-end server. If an attacker could instantiate the commands sent by a legitimate controller, then he/she could seize the system control with crucial consequences, such as wrong network setup or, considering smart houses, an opening of a window during the night. Therefore, secure multicast is vital for the Internet of Things. Such Internet of Things constrained networks are called Low-Power and Lossy Networks (LLNs) since interconnected devices are resource-constrained devices with limited bandwidth. In such networks, CoAP is an application layer protocol and DTLS is enabled by default at CoAP, thus DTLS is selected to be used for secure multicast.

The DTLS is chosen to be used for securing multicast since it is enabled by default at Low-Power and Lossy Network (LLN) CoAP devices. The DTLS In Constrained Environment (DICE) working group of the Internet Engineering Task Force (IETF) performs efforts to standardize DTLS-based multicast [55], however, the Internet-Draft they are working on does not provide all the important features for secure multicast. Hence, we draw requirements to secure multicast in the following Section 3.2 and in Section 3.3 we analyze the DTLS-based multicast [55] and summarizes the requirements covered and not covered by the draft.

### 3.1 System and Security Model

In this master thesis, we take a smart outdoor lighting control system in the street as a main use-case scenario. In this system, lighting devices on the street establish mesh networks and interact with each other reacting on events which might happen, e.g. a coming car. However, all these devices are configured and controlled by a back-end server from somewhere on the Internet. The back-end server can periodically send various commissioning information to the devices in the network as well as centralized commands to turn on/off the light. There might be other devices in the network then lightning devices which can also be controlled by the back-end server but they are not mentioned in this project.

Figure 3.1 schematically shows the outdoor lighting use-case. The network established with the lighting devices is a constrained network with resource con-
strained devices. Each device has an IEEE 802.15.4 interface. Primarily, IEEE 802.15.4 2006 edition [2] is considered to be used, however, it might be considered to move towards a more powerful version of the standard which is IEEE 802.15.4g [3]. Devices establish a mesh network over the IEEE 802.15.4 interface. IPv6 is used as a network protocol and RPL protocol is used as a routing protocol [95] for this network and UDP – as a transport protocol. The transmission of the IPv6 and UDP over constrained environment is used as defined in [62, 45]. All the lighting devices form separate multicast groups are to be configured by the back-end server. Hence, the communication scenario with one sender and many listeners (1-to-M) is considered. All of the lighting devices have the same hardware resources, while one device per multicast group acts as a border router receiving the multicast message from a server and distributing it to the rest of the group members. A device acting as a border router has an Ethernet interface as well. The border router receives a multicast message through the Ethernet interface from the server and distributes the message through the IEEE 802.15.4 interface to the rest of the group. In this project, we assume that the infrastructure, which is used to send an insecure multicast from back-end server to the constrained network, is already in place. In our use-case, the back-end server is a powerful device with no restrictions on memory or computation resources. All the devices in the network are enabled with DTLS for pairwise secure channels establishment as well as DTLS is supposed to be used for secure multicast as proposed in the DTLS-based multicast draft [55]. The CoAP protocol [80] is used as an application layer protocol and it is used to generate multicast messages. Hence, the back-end server sends secured by DTLS multicast messages to each multicast group. The network can contain many devices which are split into distinct multicast groups, each group is about 100 devices.

Our security model is based on the main use-case where a back-end server sends commissioning data to the outdoor lighting devices. In this use-case, the server is assumed to be placed securely and it is assumed to be hard to compromise. The outdoor lighting devices are considered to be easily compressible. The probability of a listener device being compromised is equally distributed between lighting devices. With this security model, source authentication of the multicast originated by the server is important since it considerably improves the security of the system considerably. Moreover, in our scenario the server acts as a trusted party, commonly called, trusted party, i.e. the server can act as a group controller who takes responsibilities for the multicast group management. The server manages the group membership as well as it is responsible for the keying material generation and distribution.
3.2 Requirements to Secure Multicast over Constrained Networks

This section presents requirements applied to multicast communication for the Internet of Things. Perrig and Tygar [73] give a discussion on properties desired from multicast communication. Taken this discussion as one of reference points, we present a list of requirement to multicast for constrained environment justified with the needs and limitations of Philips:

- **Scalability** towards a large number of listeners is an important requirement since scalability is an original intention behind multicast. The limitation we put on our particular application is rather small. The multicast groups are considered to be up to 100 devices per group in order to manage multicast groups more efficiently. However, in other applications much more devices can be in a group, e.g. thousands of devices.

- **Diverse range of available resources** for listeners should be taken into account due to the fact that multicast is destined to multiple end-points.
Some of them can be computationally powerful with high bandwidth available, while others can have smaller resources. However, in our scenario with outdoor lighting, all the devices have equally restrictive bandwidth and computational resources.

- **No retransmissions** are anticipated since retransmission of the lost packets causes several problems in regards to multicast. Firstly, it might cause network congestion given the limited bandwidth in constrained networks. Secondly, the handling of the retransmitted packet harvests resources on the nodes which received the packet successfully. For example, multicast over TCP transport would not work for constrained environments.

- **Real time data.** A sender is assumed to have no knowledge of data in advance, as well as a receiver immediately processes received data. This applies certain limitations on performance characteristics: a sender is not able to prepare packets in advance and a receiver needs to pass the data to application as soon as possible. In our case, listeners process multicast messages as soon as they receive them.

- **Security: authenticity and confidentiality.** Two types of authenticity are distinguished: group authenticity and source authenticity [24]. *Group authenticity* assures that a multicast receiver is capable of detecting if a packet was sent by a group member or not, in this scenario a keyed MAC can be sent along with the message where the key is a secret key shared within a group. *Source authenticity* means that a multicast receiver is able to unambiguously recognize the identity of a sender, e.g. using a digital signature algorithm. Depending on a specific scenario, either *group authenticity* or *source authenticity* can be required. However, *source authenticity* provides stronger security guarantees, especially, if multicast listeners can be easily compromised which is the case in our security model, a detailed discussion of the security model is presented in Section 3.1. *Confidentiality* requirement implies that only multicast group members can read the content of a multicast packet dedicated to the group. Security mechanisms should comply with the following restrictions. The first four of them are derived from the aforementioned requirements:

  - **Low computational overhead** is implied by the assumption about computational power limitations of devices in a constrained network. The security mechanism should not take all computational resources of a device, since the device should be able to perform its primary functions as well.

  - **Low communicational overhead** comes from the wireless medium used for communication. The security mechanism should not introduce large overhead becoming an obstacle to transmit a cargo. In our case, IEEE 802.15.4 standard is primarily selected for MAC and PHY
layers which implies strict bandwidth and packet length limitations which are discussed in Section 2.1.

- **Robustness to packet loss** is implied by the absence of the retransmission. Hence, given a loss of an intermediate packet, the packet received afterwards should be processed anyway despite any losses. For example, in TLS, reliability and correct ordering of the packets are ensured with underlying TCP, and TLS is not robust to packet losses.

- **Scalability towards the large number of receivers** - the security mechanism should not significantly limit the scalability of the multicast.

- **Replay protection** mechanism is also required to prevent attacker from reusing eavesdropped messages. Commonly, a sequence number, nonce or timestamp can be used to protect from replay attacks.

- **Secure key distribution/update** is an important part of the secure multicast. Not only group members should be bootstrapped in a secure way but also a late join and leave operations should be handled. The specific requirements to group key management are four-fold:
  
  * **Group key secrecy** - group keys should be available only to the multicast group members.
  
  * **Backward secrecy** - a late joined group member should not be able to read previously sent packets.
  
  * **Forward secrecy** - a leaving group member should not be able to read packets sent to this group after he/she has left.
  
  * **Reliable key update**. The key update procedure is loss sensitive. If a packet containing a new key is lost, then a receiver will not be able to decrypt all the packets encrypted with this key. Therefore, the key update messages are usually retransmitted in case of any loss.

The requirements to group key management are discussed more in detail in Chapter 5.

### 3.3 Requirements covered by DTLS-based Secure Multicast

In this Section, we present a summary of the DTLS-based secure multicast [55] analysis in regards to the covered and missing requirements applied to secure
multicast for LLN which are listed in Section 3.2.

In current version of the DTLS-based secure multicast draft, symmetric encryption is considered to be used to provide confidentiality and a group level authentication is provided via keyed MAC [55]. Thus, the computational overhead is low due to the efficiency of the cryptographic primitives being used. In regards to the communication overhead, the actual overhead depends on a cipher suite used. For the DTLS-based multicast, AES-CCM with 8 bytes of authentication information is used, and therefore, we consider that DTLS-based multicast provides low communication overhead. Robustness to packet losses is achieved by the use of the UDP as a transport protocol for DTLS and by the modification applied to TLS in order to eliminate inter-record dependency. Considering scalability, even though the theoretical limitation is 32767 devices per group, the authors of the draft recommend to limit the size of a group to 100 devices with an intention to minimize the consequence of a device compromise by an attacker [55]. This precaution is implied by the use of the mechanism ensuring group authentication only. Replay protection is provided by tracking sequence numbers included into each record transmitted from a sender to receivers. Confidentiality can be ensured with a symmetric encryption mechanism, as previously mentioned, AES-CCM is used by default which also provides authenticity but on the group level only.

So far, the discussed above requirements applied to secure multicast are satisfied by the DTLS multicast draft. However, not all the requirements are covered, in particular, requirements to source authentication and key management. According to the authors of the DTLS-based secure multicast [55], source authenticity is not necessary since the devices in a multicast group are considered trusted and a multicast sender is compromisable as well as the listeners are. However, in our use-case, source authenticity is an important requirement since in our model a multicast listeners are believed to be easily compressible, unlike the sender. Group key management is an important part of the secure multicast e.g. devices can join and leave a group and, in order to fulfill forward and backward secrecy requirements, group keys should be renewed respectively. The DTLS-based secure multicast adapts the record layer of DTLS to support multicast communication, the group security keys are supposed to be pre-installed to all the group members and the key management is considered out of the scope of the Internet-Draft [55]. The Group Secure Association Key Management Protocol (GSAKMP) is referenced to be adapted for the DTLS-based multicast key management needs, however, no discussion regarding any modification required to adapt GSAKMP to the context of DTLS is mentioned.

The summary of the requirements covered by the current Internet Draft for DTLS-based secure multicast is compendiously drawn in Table 3.1. The clear-cut conclusion to the analysis of the DTLS-based multicast is that this draft is
not intended to cover source authentication and key management procedures, whereas such procedures are not yet designed for the DTLS-based multicast.

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low computational overhead</td>
<td>Yes, symmetric crypto primitives</td>
</tr>
<tr>
<td>Low communicational overhead</td>
<td>Yes, MAC size</td>
</tr>
<tr>
<td>Robustness to packet loss</td>
<td>Yes, datagram transport</td>
</tr>
<tr>
<td>Scalability</td>
<td>No, in theory max. 32767 nodes per group, but 100 is recommended</td>
</tr>
<tr>
<td>Replay protection</td>
<td>Yes, sequence numbers tracking</td>
</tr>
<tr>
<td>Confidentiality</td>
<td>Yes, record layer</td>
</tr>
<tr>
<td>Group Authenticity</td>
<td>Yes, record layer</td>
</tr>
<tr>
<td>Source Authenticity</td>
<td>No</td>
</tr>
<tr>
<td>Key distribution</td>
<td>Not defined</td>
</tr>
<tr>
<td>Key update group secrecy</td>
<td>Not defined</td>
</tr>
<tr>
<td>Key update backward secrecy</td>
<td>Not defined</td>
</tr>
<tr>
<td>Key update forward secrecy</td>
<td>Not defined</td>
</tr>
<tr>
<td>Reliable key update</td>
<td>Not defined</td>
</tr>
</tbody>
</table>

Table 3.1: Requirements covered by DTLS-based Secure Multicast for LLN

3.4 Motivation for Source Authentication at DTLS

Source authentication is indispensable for secure multicast. However, source authentication might be ensured at different layers from the architectural point of view, e.g. at an application layer, or at a transport layer. Considering the transport layer, source authentication is considered to be added as an extension to the existing DTLS-based multicast solution which is going to be available by default at all lighting devices in the network. Moreover, DTLS unicast protocol is tightly incorporated to the architecture\(^1\). Furthermore, the DTLS-based multicast is designed to provide protection of multicast communication, while it does not ensure such an important property for secure multicast as source authenticity. Such a downside seriously increases an impact of a single node on the group being compromised, and therefore source authenticity is considered indispensable for secure multicast. If not ensuring it with the DTLS multicast, then it should be provided at the application layer. However, addition of a separate protocol for source authenticity at the application layer introduces an additional header for this application layer protocol, unlike the extension of

\(^1\)The Lightweight Machine to Machine architecture [67] is considered as the architecture for the client-server communication, where DTLS is used to provide security for unicast communication
3.5 Problem Statement

Currently, the DTLS-based secure multicast for constrained environment is not the solution which can be utilized in a real world scenario. Given that the key management should be in place and that we consider a security model where group members are untrusted, i.e. easy to compromise, the main problem of the thesis is two-fold:

1. The DTLS-based multicast communication supports authentication on a group level only and it is fairly easy to forge, which results in getting a system under control of an attacker. A reasonable countermeasure to such a threat is a deployment of a source authentication scheme. Even though source authentication does not completely eliminate the threat of a system getting under control of an attacker by tampering with a group member, it definitely enhances resistance to a certain level, since only tampering with a multicast sender device an attacker can achieve the consequences achievable by tampering with any device given the group authentication only.

2. The secure multicast solution without a key management scheme is impractical and can not be utilized in a real life. As discussed in the Internet-Draft [55], forward and backward key secrecy should be supported to facilitate secure join and leave operations in case of the group membership changes. However, the DTLS-based multicast Internet Draft emphasises
that key management is out of the scope of the document and so far no work has been done in this area.

3.6 Design Goals

Therefore, the major goal of the thesis is an improvement of the DTLS-based multicast solution enabling the use of it for the use-case scenario presented in Section 3.1. The enhancement of the existing solution should be done by a design which fulfills the applied requirements (see Section 3.2) and which is secure in relation to the security model presented in Section 3.1. Hence, the design goal is to extend DTLS-based multicast solution to support and utilize suitable source authentication and key management schemes, in detail:

1. One sub-goal is to evaluate and select a source authentication scheme which is the most suitable for DTLS-based multicast for constrained environment according to the requirements presented in Section 3.2 and to the use-case scenario. Then, an extension to DTLS-based multicast has to be designed to accommodate the selected source authentication scheme.

2. From the perspective of the key management infrastructure, the sub-goal is to formulate requirements applied to the key management for the constrained environment in our case. Then, prospective key management protocols should be assessed according to the formulated requirement and the best one should be selected and adapted to be used in our system for distribution of group keying material for confidentiality and source authenticity protection by DTLS multicast.
Design of Source Authentication for DTLS-based Multicast

Our contributions in this Chapter begin with assessment of different source authentication schemes [71, 20, 57, 76, 97, 82, 96, 50] in order to select the most suitable one to be used for authentication of DTLS-based multicast in constrained networks. We select two most appropriate candidates, namely: ECDSA signature and TESLA scheme. We design the two solutions of source authentication for DTLS-based multicast. Both designs are optimized for minimization of computation load and bandwidth utilization by providing the minimal recommended by NIST security level. We apply both of the schemes to the DTLS multicast and evaluate the suitability according to our use-case. In case of TESLA scheme, we even define various parameters and select cryptographic primitives to be used for the solution. Based on the results of the comparison according to our use-case scenario, we propose to use of ECDSA signature for the DTLS-based multicast. Lastly, we give a sketch of the DTLS-based multicast extension to support ECDSA.
Design of Source Authentication for DTLS-based Multicast

4.1 Selection of Prospective Candidate Schemes

As it is shown in Section 2.3, there is a plethora of solutions discussed in literature to provide source authentication for constrained networks even for 8 bit low power micro controllers. Among those approaches, various schemes are based on public key cryptography and a solution based on symmetric cryptography. In order to proceed with the design phase we select two candidates to be considered in detail.

There are several criteria taken into account in order to select the best candidates. Three of them come from the requirements to secure multicast in constrained environment discussed in Section 3.2: communicational, computational overheads and memory consumption. Communicational overhead includes the information which is needed to be transferred regularly over the network in order to perform authentication of data. Computational overhead is represented with the computations required to perform the authentications. In our case, the multicast sender is a powerful device, while multicast listeners are constrained devices. Therefore, the low computational load on multicast listener devices is considered more important than the expenses applied on the multicast sender device. In regards to the memory consumption, the memory size can be greatly influenced with particular implementation goals, and therefore, assessment of this parameter is difficult in general. However, the key sizes which are required to be stored can be taken into account. We also consider an aspect of how well a scheme is examined to be secure. The supportive facts in this case are approval by some institutions and wide deployment. If a scheme involves new design concepts, especially, algorithms, then a wide use and standardization of such a scheme implies that it is less likely to have security breaches.

The first selected candidate scheme is ECDSA [50] signature algorithm because it holds a number of valuable characteristics. The scheme possesses medium communicational and computational costs as well as a key size is considerably medium as outlined in Table 2.1, while it is well standardized, widely deployed and it is shown to be secure. Moreover, ECDSA is one of the three in total and the only ECC signature algorithm approved by NIST [64], therefore the algorithm is supposed to be exhaustively verified to be secure. The second prospective candidate scheme is a symmetric cryptography based authentication scheme TESLA [71] which is also widely applied for source authentication, e.g. for multimedia applications [15] and for authentication of reliable multicast [78]. Moreover, TESLA provides lowest computational and communicational overheads since it, essentially, utilizes symmetric only cryptography only. Downsides of other candidate schemes are discussed below.

A one-time signature, called BiBa, has a very nice property of fast verification
4.2 Source Authentication with ECDSA

approximately 20 times faster than RSA, however, such a scheme has high signature generation overhead and it requires a very large public key in the order of 10 kilobytes and 128 bytes signature to provide 80 bit security [70]. Hence, such an approach is not affordable in our case. The Bloom filter-based BAS [76] signature approach implies almost the same communicational and computational expenses as ECDSA, while it provides partial message recovery from signature which helps to minimize the size of the message to be sent over the air by 10 bytes. This is a nice property which might reduce entropy of the signature and therefore, security can be compromised. The EIBAS [82] is identity based signature which requires computationally heavy pairing and the signature size is 70 bytes for 80 bit security solution. We do not consider identity based signature with such characteristics attractive to our project as well as it is now widely known and verified to be secure. Moreover, both of the aforementioned solutions are not standardized and widely used. The polynomial based PBA [97] scheme, implies very cheap overheads however security of such a scheme is not provable and in particular, PBA scheme is successfully broken [7]. Even though there are similar to PBA schemes, we do not consider such an approach due to unproven security properties. There is an attractive niche of ECC signatures, called short signatures, like BLS [20] and ZSS [96]. The signature size of such an algorithm is twice as less compared to ECDSA with the equivalent security strength, however signature verification operation is a computationally heavy operation requiring pairing, e.g. BLS signature with a little under 80 bit security strength requires 2.9 s to be verified on 1 GHz Pentium III [20]. Therefore, such an approach is considered unacceptable for our constrained receiver devices.

To select the best suitable option for the DTLS-multicast source authentication, the ECDSA and TESLA are applied to our main use-case specifying the necessary parameters, and then, the best option is selected based on how they comply with the requirements of our scenario.

4.2 Source Authentication with ECDSA

4.2.1 Selection of Security Strength

According to NIST, since the year 2010 it is recommended to use algorithms providing at least 112 bit security level in order to assure uncompromising data protection [13]. However, as NIST estimates, 112 bit security level will no longer be secure enough by the year 2031 [12]. Table 4.1 presents the NIST recommendations applied to different versions of ECDSA. According to the research conducted in the year 2009 by the researches from EPFL, Alcatel-Lucent Bell
Laboratories and Microsoft Research, the ECDSA-160 over prime field with 80 bits security level will still be secure against open community attacks till the year 2020 [21].

<table>
<thead>
<tr>
<th>Type</th>
<th>Security level (bits)</th>
<th>Key length (bits)</th>
<th>Signature size (bits)</th>
<th>Use recommendation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECDSA-160</td>
<td>80</td>
<td>160</td>
<td>320</td>
<td>Disallowed after 2013</td>
</tr>
<tr>
<td>ECDSA-192</td>
<td>96</td>
<td>192</td>
<td>384</td>
<td></td>
</tr>
<tr>
<td>ECDSA-224</td>
<td>112</td>
<td>224</td>
<td>448</td>
<td>Recommended</td>
</tr>
<tr>
<td>ECDSA-256</td>
<td>128</td>
<td>256</td>
<td>512</td>
<td></td>
</tr>
</tbody>
</table>

Table 4.1: NIST Security Level Recommendations

The ECDSA with 224 bit key provides a minimal security strength recommended by NIST. Therefore, as a design option we select ECDSA-224 to assure that data origin authentication is performed in a secure way which implies that there is no threat that an attacker can brute force a private key or break the algorithms in any other ways. On the one hand, selection of a recommended security level eliminates a need in frequently updating of a private key. On the other hand, we select the minimal acceptable security level which minimizes the key sizes as well as power, memory, computational overhead and bandwidth consumption [19]. Moreover, according to NIST, 112 bit strength signature applied now will be valid and trusted for the next 6 years and this will hold till the year of 2031 [12].

Since the 112 bit security strength is considered safe to be used until the year 2031, we emphasize that the system should be reconsidered from the viewpoint of security and appropriate security level should be used, e.g. 128 bit security strength. With the increase of the security level from 112 bits to 128 bits, ECDSA signature size increases by 8 bytes from 56 bytes to 64 bytes. Hence, if a system can easily tolerate an increase in the signature size by 8 bytes, then 128 bit level of security could be selected to provide security guaranties for longer period of time. This would mean that if no breaches will be found in the ECDSA algorithm in the future and computational abilities of the best possible attack increases as estimated by current research results, then the security of the designed system would last longer than till the year 2031.

Due to our constrained environment, i.e. low bandwidth and a possible packet size, currently, we select 112 bit security level to propose a secure solution with the minimized overhead on communication.
4.2.2 Selection of the Curve

In order to select a particular curve we refer to the list of the curves supported by TLS which can be used at the handshake protocol. There are a number of binary and prime field curves supported by TLS [19]: sect163k1, sect163r1, sect163r2, sect193k1, sect193r2, sect233k1, sect233r1, sect239k1, sect239r1, sect409k1, sect409r1, sect571k1, sect571r1, secp160k1, secp160r1, secp160r2, secp192k1, secp192r1, secp224k1, secp224r1, secp256k1, secp256r1, secp384r1, secp521r1. There are two binary curves and a prime curve supported by NIST [39] which provide 112 bit security level. The binary curves are a Kotblitz (sect233k1) curve and a random (sect233r1) curve with the size of the key equal to 233 bits. A prime curve is secp224r1 with the size of the key equal to 224 bits.

The deep analysis of the performance characteristics of the different curves is considered out of the scope of this project. However, there is research conducted on software implementation performance on all 3 possible elliptic curves [23]. According to the results of this research, the implementation of secp224r1 has the best performance among the rest of the candidates both in signature generation and verification [23]. However, Oliveira et al. achieves twice as less time required for generation of signature with binary curve as it is required to generate a signature with prime curve [65]. In regards to ECDSA implementation in hardware, the area and power consumption are shown to be similar for ECDSA over both prime and binary fields, however the run time of implementation over binary field is 1.5-2.8 times faster than the implementation over prime field [94]. Any of the officially approved by TLS curves can be selected for a particular implementation depending on the requirements, we select the secp224r1 prime curve.

4.2.3 Selection of the Encryption Mode for AES

In order to enrich the DTLS multicast with the source authentication, an encryption mechanism should be altered. Initially, AES in CCM mode is selected for encryption, however AES-CCM provides both confidentiality and authenticity. In CCM mode Counter Mode (CM) is used for encryption and CBC-MAC is used for authentication. In case of AEC-CCM-8, 8 bytes of authentication information are added to the trail of a message. Hence, addition of the signature directly in combination with the CCM mode is not efficient, since superfluous computations are performed to compute CBC-MAC as well as 8 bytes of superfluous information are attached as a trailer to data. Thus, we propose to use AES-CM for encryption and ECDSA for authentication. Counter Mode is
sensitive to repetitions of an initialization vector. Any plaintext encrypted with the same secret key should be encrypted with distinct initialization vector values, otherwise, confidentiality of any plaintext encrypted with the same key and the same initialization vector can be compromised if the Counter Mode is used for encryption [36]. The same requirement for initialization vector uniqueness applies to CCM mode as well [37]. DTLS-based multicast proposes a solution to ensure uniqueness of the initialization vector for the CCM mode which is previously mentioned in Section 2.2.1. Therefore, we apply the same approach for the Counter Mode, composing an initialization vector from client/server salting value, unique sender ID, sequence number and epoch which ensures that initialization vector is used only once for encryption with a given secret key.

Even though the encryption algorithm provides 128 bit security strength, the overall level of security in the system is 112 bit since selected signature mechanism version provides only 112 bit security. Thus, the system security is derived from the weakest component of the system. Hence, one may suggest to reduce the level of the security of the encryption algorithm to 112 bit level as well. AES with 128 bit key provides the weakest security possible with AES and it already persists in the system since AES is as well used for unicast communication encryption with DTLS which helps to keep the memory requirement to be minimal. Furthermore, three-key Triple DES is the only algorithm approved by NIST providing 112 bit security [13] and it is considered to be 3 times slower than AES-128 [68]. Therefore, AES is reused with combination with 112 bit security strength signature.

4.2.4 Combination of Signature and Encryption

This section discusses how to apply signature at the DTLS record layer. First, we consider a case when no confidentiality requirement is applied, therefore, no encryption is needed. In this case, the DTLS standard suggests to generate MAC not only on data but also concatenate it with epoch and sequence number to prevent replay attack [77]. Hence, to apply the signature we also suggest signature generation on concatenation of epoch, sequence number and data, analogically to MAC generation. However, signature can be applied to a whole record header, since the header of a record should be delivered intact to the recipient due to the fact that DTLS is located on top of the transport protocol.

If confidentiality is also required, then a right combination between encryption and signature generation should be selected. Nonetheless, there is no particular recommendations for encryption and signature generation as it is proposed for encryption and MAC combination [1, 17]. Assuming that encryption and signature mechanisms are implemented in a secure manner, we propose to en-
crypt plaintext and then, to generate a signature on the selected DTLS record header fields and ciphertext. Such combination, apart from confidentiality and authenticity of the plaintext data, also provides us with authenticity of ciphertext. On the contrary, generation of a signature on plaintext first and then, encryption of the plaintext does not provide us with the authenticity of the ciphertext. There is also an option to generate signature on plaintext and then, to encrypt both the plaintext and the signature. Since the signature algorithm is secure and it is believed that no information can be derived from a signature itself, the option with encryption of the signature is not considered valuable.

In order to select the best suitable option of the two prospective ones, let us show a situation applicable to our main use-case when authentication of the ciphertext is beneficial. Let’s assume that the back-end server uses the same public-private key pair for many distinct multicast groups and there is an attacker who compromised two multicast listeners from distinct groups. In such a case, the attacker has in his/her possession group encryption keys of the groups the compromised devices belong to. Hence, if we generate a signature on plaintext, then the attacker can tamper with a command, encrypting the command destined to one group with the group key of another multicast group. Since the signature is generated not only on plaintext but also on truncated sequence number, sender ID and epoch, the attacker has to ensure that the epoch and the sequence number are acceptable by the other group\(^1\). If they are acceptable, then the tampered message will be processed by the multicast listeners as if this message was originally dedicated to this group. Hence, generation of the signature on ciphertext protects from such an attack, since signature is no longer verifiable if the ciphertext changes, i.e. the plaintext is encrypted with another group key. However, if separate public-private key pairs are used by the back-end server for each group, such kind of attack is no longer possible.

### 4.2.5 Applying ECDSA to the Main Use-case

For now we assume that the group key and public key are in place at all the devices. The signature should be added instead of the MAC to the DTLS record. Currently IEEE 802.15.4 the year 2006 standard [2] is considered as it is proposed with the 6LoWPAN compression mechanism from 2011 [45].

The IEEE 802.15.4 provides a maximal MAC frame size which is up to 127 bytes, where the maximal frame overhead is 25 bytes and it optionally can be reduced to 13 bytes if short addressing (4 bytes per address) is used which yields 102 bytes and 114 bytes for payload [2]. The IEEE 802.15.4 security mechanisms take 21 bytes more, however, we assume that secure data transmission is assured

\(^1\)A sequence number and an epoch should be considered valid, e.g. a message with such a sequence number and the epoch should not be previously received by the multicast listeners.
at the transport level with DTLS. The IPv6 header without extensions and UDP packet headers all together comprise 48 bytes, 40 and 8 bytes, respectively which leaves only 54 spare bytes for higher layers protocols. Therefore, the IPv6 and UDP headers compression mechanism is introduced [45, 62], which is able to considerably compress the headers.

However, considering an IEEE 802.15.4g [3], the upper bound on MAC frame size is up to 2047 bytes. Even though the use of the 802.15.4g standard implies a different hardware segment choice, selecting 802.15.4g standard fragmentation will not be needed in most cases. While, it can be considered as an option, the minimization of a packet size is still required since smaller packet results in smaller probability of errors.

It is desirable to fit regular commands to a single frame in order to avoid fragmentation. There might be longer messages bearing commissioning data which do not fit to a single 802.15.4 frame. The regular commands are assumed to be in the range of several bytes, e.g. containing information to turn on/off lights. Hence, we estimate the available space per frame. The exact size of available payload calculation using compression mechanisms is not the topic of this project. The IPv6 and UDP headers compression mechanisms [45, 62] are complex and require consideration of various details at MAC and IP level to determine which compression options can be applied in a specific case. These details are not available now and therefore, we rely on an approximate number which might be different at the deployment network setup. In this project we consider the best case: that at MAC layer the short addressing feature is used, and thus, the MAC overhead is 13 bytes only which includes the 11 bytes header and 2 trailing bytes for Frame Check Sequence (FCS) [2]. Also, we consider that the best possible compression is used for IPv6 header. As far as it is going to be transmitted over the multiple hops, the size of the best possible compressed header is equal to 7 bytes [45]. In regards to the UDP header compression, in this project, we assume that the compressed header comprises 7 bytes, since the destination and source ports can be compressed to 4 bits each, only if they are set in a specific 16 ports range. Most likely, the range of the ports is going to be different.

Next, the DTLS and CoAP headers should be considered. The DTLS multicast header takes 13 bytes plus length of a MAC [55] and the CoAP header requires 4 bytes with 1 byte thr length option fields according to IETF draft [80]. MAC can be excluded from calculations since it is substituted by signature in this case. There is also a DTLS header compression mechanism introduced for unicast communication. The proposed draft reduces overhead implied by the DTLS record header from 13 to 10 or maximum 4 bytes [73] depending on the epoch and sequence number use. This compression mechanism is not applicable directly to DTLS multicast since 1 byte of sequence number is allocated for the
sender ID but it can be adapted if it is needed. The minimal size of the DTLS header in this case will be 5 bytes, however, we do not consider application of this compression in this project.

If CoAP protocol is supposed to be used directly to send the commands, then our approximation results in 82 bytes available for the application data and the signature, 26 bytes for the application data respectively as presented in Figure 4.1. The 26 bytes space should be enough for a regular command.

![Figure 4.1: Payload size per frame with ECDSA signature](image)

### 4.3 Source Authentication with TESLA

We select the original TESLA scheme and not the μTESLA scheme to further elaborate on our design since our receiver devices are considered powerful enough to handle a disclosure of a key at each interval as well as TESLA is easier to start with due to the fact that a key is disclosed in each packet. Furthermore, in our use-case a multicast sender is a back-end server which can create a one-way chain of sufficient length to avoid a need in subsequent commitments. The description of the TESLA scheme is presented in the Section 2.3.2.

#### 4.3.1 Selection of Parameters and Crypto Primitives

To consider the TESLA scheme as a design option for the DTLS-based multicast source authentication, a number of aspects have to be specified, such as:

- Time synchronization mechanism;
- Key length and one-way chain value length;
- Time interval length;
- Key disclosure delay;
- One-way key chains length;
• Selection of a Pseudo Random Number Generator (PRNG) to generate an initial value for one-way chain;
• Selection of HASH function;
• Selection of a function to derive a key;
• MAC length;

The specificity of applications takes a great role in the design for the TESLA. In our case, a back-end server sends a commissioning data to the outdoor lighting devices via multicast. In this scenario, the multicast sender is always online. Other use-cases might assume that the multicast sender can be temporarily unavailable, e.g. human controlling the light in the room. In this case, a human directly observes the result of a triggered action. The aforementioned distinction is especially important to be considered while selecting the length of the key disclosure interval. Furthermore, in all these use-cases, multicast messages are sent periodically rather than constantly. These aspects are taken into account while specifying parameters for TESLA design option:

• Time-synchronization mechanism

In our case, the outdoor lighting devices are considered to be GPS enabled. The assumption is that GPS is used to provide time synchronization, therefore, time synchronization mechanism is out of the scope of the project. However, in order to ensure that the time synchronization is performed securely, one would need to consider GPS spoofing attacks. GPS spoofing attack is an attack when an attacker tricks a GPS receiver to calculate its position and time different from the real ones. Such attacks are possible especially on civilian GPS receivers which do not have neither authentication nor encryption of the messages [86]. An attacker can mount a GPS spoofing attack by locating a single antenna which sends a set of satellite signals with certain position and time settled by the attacker. Civil GPS receivers in the area of reachability of the signal will be tricked to believe that they are in a spoofed position and will set the time distributed by an attacker [86]. However, the position at receivers will be set to the same value distributed by the attacker, while the time will be different depending on the time receivers get a signal from the attacker device. If the distance is in the range of several meters, the time difference will be in the range of nanoseconds [86]. The countermeasure against such an attack would be to provide communication between the devices with the aim to check the relation of their locations, i.e. if they have the same location, while they are physically located in different places, and the time is not synchronized. Given that one of these statements is true, an alarm can
be raised to warn that something suspicious happens. More sophisticated attacks can be mounted by increasing the number of spoofing devices, however, with the increasing number of communicating civil GPS receiver devices the attack becomes more and more difficult to mount [86]. The practical implementation of the GPS spoofer is presented in [46].

- **Key and One-Way Chain Value Length**
  
  In order to achieve the recommended by NIST minimal security level (112 bits) [13], we select a key length equal to 112 bits. Each key will be valid for at most several seconds, and therefore, the key length could be reduced to a smaller value. To further minimize the key length one needs to estimate the security level which can be broken within a time interval of key validity given state-of-the-art technologies. However, this is considered out of the scope of the project.

  Furthermore, to provide claimed 112 bit security strength for the authentication keys, the mechanisms utilized to derive a key should provide at least the same security level. One-way chain values should have at least the same security level, and therefore, they should be at least 112 bits long. The discussion below takes into account that the key and one-way chain values should provide 112 bit security strength.

- **Time Interval (t)**
  
  Different application scenarios apply distinct requirements for the time interval for the same key to be used for message authentication. The length of the interval influences the authentication delay and the number of keys in a chain for the same time period. The main use-case, when outdoor lighting devices are controlled with back-end server via multicast, does not apply strict requirements to responsiveness comparing to the case when a human controlling the indoor light observes the result of an executed action. Furthermore, the key disclosure interval should not be less than the upper bound of the round trip time of a message transfer to any receiver, otherwise, messages might arrive to the receiver at the next time interval, and thus, be rejected [69]. The suggested by the RFC 4082 [69] way to minimize authentication delay is to take a maximum value from RTT and the expected period of time that the sender will multicast messages. To give an idea, an upper bound on RTT on the Internet is estimated to be often not more than 500 ms given absence of problems with routing [69].

  In our main use-case the back-end server’s communicational pattern is rather periodical than persistent. Nonetheless, in order to provide higher responsiveness, the key disclosure interval can be settled in the range of 1-5 seconds, assuming that the RTT is lower.

- **Key disclosure delay (d)**
The key disclosure delay specifies the number of time intervals $t$ to be lapsed before the key utilized in the current interval can be disclosed [71, 69]. RFC 4082 recommends that the key disclosure delay should be set to a minimum of 2 intervals, since the key disclosure delay equal to 1 can result in rejection of the message. If the sender send a message in the end of the interval $i$, a receiver might receive it in the interval $i+1$. Hence, if the key disclosure delay is set to 1, then the key for the interval $i$ is disclosed in the interval $i+1$, and the receiver can receive it in the same interval as the message authenticated with this key was received. Nonetheless, $d=2$ is the minimal key disclosure delay, the upper bound on the authentication delay rises to $2 \times t$ in the worst case.

### One-Way Chain Length

According to the assumptions about computational power of the devices, a multicast sender device is powerful, while multicast receiver devices are constrained devices with limited resources. Hence, we assume that the server is capable of generating a one-way key chain of sufficient length to avoid problems with redistribution of a new commitment value.

For example, with the time interval equal to 2 seconds and a single value of a one-way chain equal to 112 bits, it will be 43200 intervals in a day or 15768000 intervals in a year which is approximately $2^{24}$ intervals, i.e. 2216 GB of storage for a year. However, we can store only intermediate points and the rest can be computed as soon as they are needed, e.g. storage of the values for 3 hours (approximately $2^{12}$) comprises approximately 79 MB which is depicted on Figure 4.2. Furthermore, there are solutions to dramatically increase storage and computation efficiency of one-way hash chains. The solution proposed by Jakobsson optimizes storage and computational requirements to $\log(n)$ where $n$ is the number of elements in the chain [47]. The high level idea is that in this approach only $\log(n)$ intermediate values are stored and the rest is computed. As soon as the intermediate value is reached, it is moved further dividing next interval into two smaller intervals. More detailed information can be found in the original paper [47]. Following the aforementioned approach, in our example, the storage required for one year chain with 15768000 values of 112 length comprises 480 bytes and 24 hash function computations per value, while for 10 years 588 bytes and 27 hash function computations per value are required. Moreover, Coppersmith and Jakobsson further optimize the solution achieving slight improvement in storage overhead, while twice decreasing computational expenses per chain value [29].

### Generation of an initial value for the one-way chain

The initial value for the one-way chain should be generated in a way that there are no shortcuts in deriving this value other than bruteforce, since such a shortcut lowers down the security level of the whole system. If an
4.3 Source Authentication with TESLA

The circles are representing the values which are stored. In the beginning, all values from 0 to 3 hours are stored, which is approximately $2^{12}$ values considering the one value for 2 seconds time period. When the time starts approaching 3 hours, these values will be rewritten with the corresponding values computed for the next 3 hours intermediate interval.

attacker is able to derive this value, then he/she can forge any message with authentic MAC. Therefore, a reasonable attention should be paid to generation of the initial value. A Cryptographically Secure Pseudorandom Number Generator (PRNG) should be used. There are some standardized suggestions for generation of a random value [9]. However, we assume to reuse a PRNG from existing secure system, such as OpenSSL [6] which is an implementation of TLS v1. We further assume that the seeds for the PRNG are securely generated and already stored at the sender device.

- **Hash Function Selection**

One-way hash function is used in three places in TESLA:

1. To generate values of self-authenticating one-way chain;
2. To compute a keyed MAC;
3. To derive a key from a value of one-way chain.

To select a hash function, widely used, approved by NIST hash functions are considered, such as: SHA-1, SHA-2 family [30]. As reported by NIST, collision resistance for the approved by NIST hash functions comprises half of the length of a hash value [30]. Exceptionally, collision resistance of SHA-1 is considerably lower than half of the hash value. Output of SHA-1 is 160 bit, however, according to cryptoanalysis, SHA-1 collision resistance strength is lower than 69 bit level [91]. Stevens, with his recent cryptoanalysis of SHA-1, argues that there are attacks on SHA-1 with complexity equivalent to 61 bit security [83]. Thus, in order to perform such a collision attack, one will need to collect $2^{61}$ pairs of plaintext and corresponding SHA-1 output values. Even though such a scenario is considered impractical, SHA-1 function is vulnerable and it is not selected in order to avoid any further issues if the cryptoanalysis will make a step further and this hash function will be considered even weaker. The SHA-2
family hash functions do not have any vulnerabilities found yet, e.g. SHA-224 has 112 bit security strength in resistance to collisions [30]. Thus, the SHA-224 function is selected to be used for one-way chain values generation. As discussed above, the self authenticating values should pertain 112 bit security as recommended by NIST [13], and therefore, 112 left most bits are taken from the SHA-224 output to present a value.

For the MAC generation, Keyed-Hash Message Authentication Codes (HMAC) based on SHA-224 is selected to be used. Hence, each MAC is generated using an authentication key.

To derive an authentication key, we propose to follow NIST recommendation for key derivation [26]. There are 3 distinct modes proposed by NIST to derive a keying material for a required number of keys, all of the methods are based on a PRF. As a PRF we reuse the HMAC function and we select a counter mode for key derivation. In counter mode of key derivation recommended by NIST, HMAC is used as many times as it is needed to generate a keying material of the required length. Each time HMAC is used, it takes 3 input parameters: key derivation key, counter and fixed input data. The key derivation key in our case is represented with a one-way chain value which is generated in a secure way. The value of a counter is incremented with each time HMAC is applied during one key derivation procedure. In our case, only one key of 112 bits length is to be derived, thus, a single HMAC produces keying material of sufficient length for an authentication key. The fixed input data parameter is represented by a Label, a separation indicator 0x00, a Context and length (L) of a required output represented in bits. The Label can be represented as a string constant and it indicates the purpose of the derived keying material to be used for. In our case, the derived keying material is used as an authentication key, thus the Label should encode such a meaning. The Context is an identifier of the parties who derive or use keying material. We propose to put into the Context field an identifier of a sender. The length (L) of a required output in our case is set to 112 bits. Since the HMAC produces an output value bigger than the size of the key, the 112 left most bits are taken as a key. A derived key inherits the security strength of the cryptographic primitives utilized to generate it, and hence, an authentication key has 112 bit strength.

- **MAC Length**

In order to minimize communicational overhead of our source authentication design using TESLA, we use truncated HMAC value of length \( l \) to be transmitted for the message authentication purposes. Only \( l \) of left most bits of the HMAC output are transmitted as authenticating information. NIST recommends to transmit at least 64 bits value to make neglectable the probability of an attacker being able to generate a correct truncated
HMAC without knowing the key [30]. The probability of the successful attack is calculated by subtracting the number of the bits in the truncated MAC by the allowed number of failed MAC authentications utilizing the same key.

For example, NIST recommendation is made considering that the HMAC key is renewed before $2^{20}$ failed authentications by the same key. In this case, if the length of truncated MAC is 64 bits, then the probability of successful attack is $2^{-44}$. In our use-case, the HMAC key is renewed much faster, in the range of few seconds, and hence, not many messages will be authenticated with the same key. Thus, the number of acceptable failed authentications by the same key can be set to $2^8$. Furthermore, in order to maintain the selected 112 bit level of security we set the length of truncated HMAC to 120 bit, so that the probability of a successful attack becomes $2^{-120+8} = 2^{-112}$.

Since the truncated MAC is 120 bits, the collision resistance drops to 60 bits. However, collision attack is considered impractical due to the fact that only several messages are authenticated with the same key. Therefore, an attacker is not able to collect even a small part of the required $2^{60}$ value pairs of corresponding plaintext and HMAC values authenticated with the same key.

Preimage resistance strength in this case is 120 bit and, since messages to be authenticated are going to be small in our system, the second preimage resistance is also considered 120 bit in our case [30]. However, we are not concerned about preimage attacks. An attacker does not know the HMAC key, and hence, he/she is not able to generate an authentic HMAC from a plaintext which is required to mount preimage attack.

Table 4.2 gives a summary on selection of the parameters for source authentication with TESLA.
Design of Source Authentication for DTLS-based Multicast

### Table 4.2: Selection of TESLA parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Selection</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time-synchronization mechanism</td>
<td>GPS</td>
</tr>
<tr>
<td>One-way chain value length</td>
<td>112 bit</td>
</tr>
<tr>
<td>Key length</td>
<td>112 bit</td>
</tr>
<tr>
<td>Truncated MAC length</td>
<td>120 bit</td>
</tr>
<tr>
<td>Generation on initial value for one-way chain</td>
<td>PRNG from OpenSSL with pregenerated seeds</td>
</tr>
<tr>
<td>Function to for one-way chain</td>
<td>HMAC-SHA224</td>
</tr>
<tr>
<td>Function to derive a key</td>
<td>HMAC-SHA224</td>
</tr>
<tr>
<td>MAC computation</td>
<td>HMAC-SHA224</td>
</tr>
<tr>
<td>Time interval length</td>
<td>1-5 sec</td>
</tr>
<tr>
<td>One-way chain length</td>
<td>10 years</td>
</tr>
<tr>
<td>Key disclosure delay</td>
<td>2 time intervals</td>
</tr>
</tbody>
</table>

In our design, we select the parameters for TESLA to provide the security level assured with ECDSA. However, the length of the HMAC key and truncated MAC length can be minimized to a security level of a particular application.

#### 4.3.2 TESLA Adaptation

The key disclosure interval length affects the computational overhead implied by the use of the TESLA scheme. In the TESLA scheme, a new key is used to generate message authentication codes at each time interval. The key used in a particular interval is revealed only after $d$ intervals, in our case, after 2 intervals. The revealing is done by sending one-way chain value corresponding to the key via multicast. Even though, technically, a key is not sent directly, the value used to derive a key is disclosed instead. We refer to this event as a key disclosure.

Our use-case considers that the multicast messages are sent in a periodical manner. Each message carries a one-way chain value to be disclosed for a previous interval as discussed above. A device receiving a message at the time interval $i$ needs to check if the MAC for this message was generated with the key which was not yet revealed at the point in time when the message was received. In order to check this, a device has to compute the one-way chain values disclosed in between the time interval $i$ of this message and the time interval $j$, when the last one-way chain values were received. If, applying the hash function $i - j$ times to currently received one-way chain value, a device gets an already verified value received at the interval $j$, it means that the device can utilize this value to verify the value to be received in the interval $i + d$. Then, the value received in the interval $i + d$ can be used to derive a corresponding key and to authenticate the message received in the interval $i$. 
4.3 Source Authentication with TESLA

If the period when no messages are exchanged comprises several days, e.g., given that the time interval is equal to 2 seconds, after two days a device will need to perform 86400 calculations of the function used to derive a next key-chain value, in our case: 86400 calculations of SHA-2. Since the lighting devices are constrained devices with limited computational power, such a scenario might further increase an authentication delay. Therefore, we propose an adaptation of the protocol to periodically force disclosure of a key based on a timeout. The adaptation is shown on Figure 4.3. The timer could countdown from the moment of the last key disclosure sent over the network. Depending on the processing power of a particular hardware to be utilized for a receiver device in the deployed system, the countdown timer value can be adjusted accordingly, so that a receiver device would require at most the time equal to one time interval to perform verification of the disclosed one-way chain value and the corresponding key, respectively. This forced key disclosure facilitates that the time required to verify the chain value does not exceed the length of a time interval, which is close to a minimal possible key disclosure delay\(^2\). Hence, assuming there are no losses in the channel\(^3\) the process of verification cannot introduce additional delays to the protocol. Even if a sender initiates a multicast message at the interval \(i\), right before the timeout for the forced key disclosure is lapsed. The one-way chain value from the previous interval \(i-d\) is disclosed with this message. This message is buffered at the recipient to be authenticated with the key disclosed in the time interval \(i+d\), meanwhile, the verification of the one-way value disclosed with the buffered message can be started. Since the timeout has not lapsed the verification of the one-way value is guaranteed to be finished before the next key disclosure appeared. Hence, making the timeout for the periodical key disclosure less or equal to the time required to perform the computation of the missed one-way chain values keeps a delay in the range of one time interval.

4.3.3 Combination of Authentication and Encryption

In order to add TESLA scheme to DTLS-based multicast, we need to combine encryption with DTLS and authentication with TESLA. The TESLA scheme requires buffering of awaiting for authentication messages at the receiver side and, at the same time, a key disclosed with the message should be used directly upon the receipt of the message.

\(^2\)If a message is sent in the end of the time interval \(i\), then a key for this message will be disclosed in the beginning of the time interval \(i+d\). Since, in our case, \(d\) is equal to 2 intervals, then, in fact, the key disclosure time delay can comprise almost the length of a single time interval.

\(^3\)If there are losses, then, due to those losses, verification can take more time than a time interval.
The forced key disclosure message is red coloured. If no messages are sent within the forced key disclosure interval ($df$), the key disclosure is enforced by the sender. The $df$ time interval is defined by $n \times t$, where $n$ is a number of time intervals corresponding to the time required to compute the number of hash operations computable by the receiver within an interval. The key disclosure delay ($d$) is represented by a number of intervals, in the example $d = 2$.

We propose to first encrypt, and then compute MAC on DTLS record fields to detect any alteration during transfer and on the encrypted payload as recommended in [1, 17]. As Bellare and Namprempre [1, 17] discuss encryption, then authentication with MAC generally works securely, providing authentication of both the cipher text and the plaintext. However, one needs to be careful while following generic approach with the specific case. Since Bellare and Namprempre [1, 17] consider only probabilistic encryption schemes and not initialization vectors (such as CM, CBC modes) or nonce based, there might be flaws. Namprempre et al. [63] disclose a flaw in ISO 19772. The choice of encryption-MAC in the standard is made by referencing to a generic composition "encryption then MAC" advertised in [1]. Due to the fact that in [1] the probabilistic encryption is considered, no reasonable attestation has been paid to the starting value which can be nones or initialization vector for different encryption modes. Therefore, we pay reasonable attention to initialization vector for CM mode as discussed in Section 4.2.3.

In our case, utilization of "encrypt then MAC" composition also prevents unnecessary computation, i.e. decryption of unauthenticated messages, since we do not need to decrypt messages in order to verify MAC. However, if we want to avoid decryption, we need to send a disclosed key in plaintext to allow the use...
of this key for the authentication of previously buffered messages. Since the key to be disclosed is no longer secret, we assume that there is no harm to reveal it in clear, even though it can still be authenticated by MAC.

### 4.3.4 Applying TESLA Scheme to the Main Use-case

Figure 4.4 shows the possible size of payload available for a command using a short addressing at MAC layer with IPv6 and UDP header compressions as previously discussed for design of authentication with ECDSA in Section 4.2.5.

![Figure 4.4: Payload size per frame with TESLA scheme](image)

Considering the proposed design of source authentication with TESLA for DTLS-based multicast, the communication overhead comprises 29 bytes: 14 bytes for a key disclosure represented by a one-way chain value and 15 bytes for the truncated MAC. DTLS record message is initially authenticated with CCM mode of AES where the MAC length is 8 bytes. The length of the truncated HMAC designed to be used for authentication with TESLA is 15 bytes. The length of the authentication key is 14 bytes. The possible command for a payload is supposed to be 53 bytes.

### 4.4 Comparison of the Design Options

This section presents a comparison of both design options. The security level is not a concern of the comparison, owing to the fact that both options are aligned to provide formally the same security level.

In general, signature mechanisms are known to provide a non-repudiation property. In particular, ECDSA provides us with the non-repudiation property which might find an application where a multicast recipient can claim that the signed data are originated from the back-end server and no one else. On the contrary, the source authentication with TESLA which does not provide non-repudiation [69]. In current scenario, received data is considered to be processed as soon as possible but not stored for a long time since such a storage increases memory expenses.
Considering the computational and communicational overheads on a high level, the source authentication with ECDSA performs worse than the one making use of TESLA. The ECDSA signature verification requires considerably more computational efforts than computation of two HMACs and SHA2 for TESLA. In order to make the comparison more obvious, let us exemplify the performance characteristics with the numbers acquired by Aydas et al. [11] on 80 MHz, 32-bit ARM microcontrollers. The authors present performance characteristics for different ECDSA versions but they do not present the ECDSA-224. However, the proportional increase of time required for the signature verification is clear, e.g. verification of ECDSA with 192-bit key takes 148 ms, ECDSA-208 takes 194 ms and ECDSA-256 takes 313 ms [11]. Therefore, we assume that verification of ECDSA signature with 224 bit key requires approximately 235 ms. In regards to SHA2 performance, e.g. hardware implementation on System on Chip (SoC) with embedded 32-bit ARM7 processor has a throughput 98 KB/s [34]. Hardware implementations are believed to be faster than software implementations. However, let us assume that, in our case, calculation of SHA-224 block requires approximately 0.25 ms as it is presented in [34]. Furthermore, let’s assume that calculation of HMAC requires 0.5 ms, since the most heavy computation in HMAC is calculation of a hash two times. In case of the TESLA scheme, let us take the key validity time interval $t$ to be equal 2 s.

Furthermore, the minimal signature size complying with the NIST recommended security level is 56 bytes as discussed in Section 4.2.1, while considering TESLA version which provides the same security level, the key and truncated MAC comprise 29 bytes of communicational overhead. Thus, in case of ECDSA signature, the additional communicational overhead in comparison with the initial group authentication with CCM is 48 bytes and, in case of TESLA scheme, it is 21 bytes. Taking into account the desirable feature to fit a command into a single IEEE 802.15.4 frame [2], the payload left for the command itself is important. Applying ECDSA signature with the best compression possible, approximately 26 bytes of payload are left for a command (see in Section 4.2.5) and, in case of TESLA, 53 bytes are left (see in Section 4.3.4), respectively.

To give a more specific comparison, 2 use-cases with distinct communication patterns should be discussed, assuming there are no losses in the channel. The first use-case is when messages frequently arrive to a multicast receiver with the period equal to the length of a single TESLA key validity time interval, i.e. each 2 s in this case. The second use-case is when messages arrive to a multicast receiver rather rarely, so that several forced key disclosures can happen during that time period. Let us take this period to be equal to 12 hours, since it is very common for the lighting systems to be turned off during the time, when it is enough daylight outside. Hence, 21600 key validity time intervals are lapsed during this time. In both use-cases, there are the same 3 parameters to assess: communicational overhead, authentication delay and computational overhead.
Since the use of ECDSA signature implies higher overhead on communication, a signature leaves little space for a command and the command size is not yet defined, it might occur that a command will not fit into a single frame and therefore, fragmentation will be needed. However, it is assumed that the communicaitional expenses implied by TESLA will unlikely cause fragmentation of a regular command. Hence, we discuss 3 options:

ECDSA$^{fr}$ - when a DTLS record with a regular command authenticated with ECDSA signature does not fit in a single IEEE 802.15.4 MAC frame;

ECDSA$^{nofr}$ - when a DTLS record with a regular command authenticated with ECDSA signature fits in a single MAC frame, i.e., either a regular command is small enough to fit into the available payload space in IEEE 802.15.4 MAC frame or IEEE 802.15.4g (which has frames up to 2047 bytes long) is used;

TESLA - TESLA scheme is applied for authentication.

Considering the system with first communicational pattern, the ECDSA$^{fr}$ option is the worst one. Each command requires 2 packets to be sent; authentication delay consists of a delay between two packets and the time required for the signature verification; the computational delay comprises the time required for the signature verification. In regards to the ECDSA$^{nofr}$ and TESLA options, both of them require only one packet to be sent but the packet sent with TESLA option is 27 bytes smaller. TESLA option requires one packet since each packet disclosing a key carries useful cargo. However, the authentication delay is longer in TESLA, since the key disclosure delay is equal to 2 time intervals ($t$) which is equivalent to a time period $x$, where $t < x < 2 \times t$, i.e., $2 < x < 4$ (s). Let us take $x$ to be the median, i.e., 3 s. Computation wise, as discussed above, the ECDSA verification is 235 ms. With this communicational pattern, TESLA scheme does not require additional hash function computations since there is no intervals for which the key is not disclosed. Hence, in this case, message source authentication with TESLA requires computation of two HMACs (one for key derivation and one for MAC computation) and one SHA, which takes much less computational efforts than ECDSA verification, as exemplified above, we assume it takes about 1.25 ms. Based on computational and communicational overhead, the TESLA option is more efficient than the ECDSA$^{nofr}$ option, while authentication delay is longer for the TESLA option. Hence, if the system can tolerate the authentication delay introduced with TESLA scheme, the TESLA option is preferable for such a communicational pattern. The summary of the discussion is presented in Table 4.3.

Considering the system with second communication pattern (periodical communication), the option ECDSA$^{fr}$ again has all the drawbacks discussed for
Design of Source Authentication for DTLS-based Multicast

Table 4.3: Comparison of the source authentication designs with frequent communication pattern (2 s periods)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>ECDSA&lt;sup&gt;fr&lt;/sup&gt;</th>
<th>Mark</th>
<th>ECDSA&lt;sup&gt;nofr&lt;/sup&gt;</th>
<th>Mark</th>
<th>TESLA</th>
<th>Mark</th>
</tr>
</thead>
<tbody>
<tr>
<td>Communication overhead</td>
<td>2 packets</td>
<td>-</td>
<td>1 packet</td>
<td>+</td>
<td>1 short packet</td>
<td>+, +</td>
</tr>
<tr>
<td>Authentication delay</td>
<td>Time to receive second packet and verify a signature</td>
<td>-</td>
<td>Signature verification (235 ms)</td>
<td>+</td>
<td>Key disclosure delay (3 s)</td>
<td>-</td>
</tr>
<tr>
<td>Computational overhead</td>
<td>Signature verification (255 ms)</td>
<td>-</td>
<td>Signature verification (255 ms)</td>
<td>-</td>
<td>2× HMAC + SHA (125 ms)</td>
<td>+</td>
</tr>
</tbody>
</table>

Table 4.4: Comparison of the source authentication designs with the periodic communication pattern (12 hours periods)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>ECDSA&lt;sup&gt;fr&lt;/sup&gt;</th>
<th>Mark</th>
<th>ECDSA&lt;sup&gt;nofr&lt;/sup&gt;</th>
<th>Mark</th>
<th>TESLA</th>
<th>Mark</th>
</tr>
</thead>
<tbody>
<tr>
<td>Communication overhead</td>
<td>2 packets</td>
<td>-</td>
<td>1 packet</td>
<td>+</td>
<td>2 key disclosure packets + 2 short packet</td>
<td>--</td>
</tr>
<tr>
<td>Authentication delay</td>
<td>Time to receive second packet and verify a signature</td>
<td>-</td>
<td>Signature verification (235 ms)</td>
<td>+</td>
<td>Key disclosure delay (3 s)</td>
<td>-</td>
</tr>
<tr>
<td>Computational overhead</td>
<td>Signature verification (255 ms)</td>
<td>-</td>
<td>Signature verification (255 ms)</td>
<td>-</td>
<td>2× HMAC + 21601×SHA (5.4s)</td>
<td>--</td>
</tr>
</tbody>
</table>

Summarizing the comparison, if the communication pattern of the system corresponds to the description of the first pattern (frequent communication), then the source authentication with TESLA is more efficient than the one with ECDSA. If the communication pattern of the system corresponds to the description of the second pattern (periodic communication with long periods of silence), then the source authentication with TESLA consumes too much resources. Thus, the source authentication with ECDSA is preferable. Considering the discussed above example with 12 hour periods between messages, in case of TESLA, to
provide the same computation overhead as the one implied by ECDSA, the computational cost of hash operation should be 23 times smaller, i.e. 11 $\mu$s, which is unlikely possible.

If command is such that signature does not fit into a single frame, then efficiency of the source authentication with ECDSA drops considerably. In TESLA scheme, with regards to periodic communication pattern, there is a dependency between the authentication delay and the communicational and computational overhead. The longer authentication delay, the smaller communicational and computational overhead.

In our main use-case communication is periodic with long periods about 12 hours and longer as well as long authentication delays are considered not desirable due to specificity of the lighting application. The design option with ECDSA is more efficient in this case, and thus, it is selected. Therefore, if a regular command authenticated with ECDSA signature will cause fragmentation, then IEEE 802.15.4g [3] might be considered to be used to avoid it, if it is needed. Of course, the choice towards the IEEE 802.15.4g implies a change of hardware and it should be made in advance.

4.5 Extension of the DTLS specification with source authentication

The design of source authentication with ECDSA is considered to be the most appropriate to be used for the DTLS-based multicast. Therefore, in this section we present an extension of the DTLS-based multicast draft [55] with support of ECDSA signature algorithm for DTLS records authentication. In order to enrich DTLS multicast with the ECDSA signature in a way that is logical and formally correct in terms of standardization efforts, the DTLS record layer data structure is modified to support the extension for the multicast source authentication. In regards to the data structures, the DTLS-based multicast draft follows the DTLS record protocol [77] which follows the TLS standard [33]. TLS specifies the SecurityParameters data structure used to link the cryptographic information to the connection state. Security parameters by default are not able to represent authentication with signature, since the data structure explicitly mentions MAC parameters. Therefore, we suggest to enrich the data structure with the parameters for signature support. Thus, depending on the ciphersuite used, either MAC parameters or signature parameters have to be set to null value. The extended data structure is presented below, the new fields are marked with the appropriate comments:
Design of Source Authentication for DTLS-based Multicast

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

All the fields pertain to the purpose as explained in the TLS standard [33], only 3 new fields are added to represent signature, namely: algorithm type, signature length and key length. SIGAlgorithm type is presented below and is used to represent a signature algorithm type in a common for TLS manner:

    enum { null, ECDSA_224, ECDSA_256 } SIGAlgorithm;

The security parameters are used to derive the set of items represented by keying materials which are linked to the connection state to perform cryptographic operations [33]. The extended set of items is presented below:

    client write MAC key
    server write MAC key
    client write signature key  // New item
    server write signature key  // New item
    client write encryption key
    server write encryption key
    client write IV
    server write IV
The DTLS-based multicast draft suggests that multicast listeners are assigned the "client" role, while multicast sender is assigned the "server" role [55]. Hence, the multicast listeners use "server write encryption key" to decrypt received multicast data and so on. In regards to the new fields, the "server write signature key" item contains server's public key at client side to verify signatures generated by the server, while, at server side, the "server write signature key" item contains server's private key to generate signatures. Such asymmetry is achieved due to the fact that there is a pair of the keys, unlike it is with symmetric cryptography. The items not applicable to a particular ciphersuite used for a specific connection are assigned with a null value as it is proposed with TLS [33]. Moreover, we propose to assign the signature keys directly to the values from the keying material distributed by a group key management protocol, unlike the rest of the parameters which are derived via a PRF function as specified with TLS [33].

The appropriate ciphersuite is proposed following the pattern of the DTLS-based multicast draft [55]. A ciphersuite for encryption with AES counter mode and authentication with ECDSA as well as a ciphersuite without encryption are presented below:

MTS_WITH_AES_128_CM_ECDSA_224

MTS_WITH_NULL_ECDSA_224

The DTLS multicast ciphersuits have a special preamble "MTS" and they contain no information about the handshake, unlike the TLS or DTLS unicast, because the keys for multicast are supposed to be distributed by some other protocol which is not yet clarified.

There is an RFC 4492 proposed to support authentication with ECDSA at the handshake layer of TLS [19]. All the needed data structures to represent ECDSA parameters can be reused from the RFC 4492. In particular, $ECParameters$ data structure represents all the parameters for both binary and prime curves depending on the curve used. Since we use the curve from the list specified by this RFC document, no changes are needed even to the name of a curve. The curve parameters should be linked with the curve name. In case of prime curve, $ECParameters$ structure contains [19]:

- A prime number $p$ which specifies a finite field $F_p$;
- Coefficients $a$ and $b$ of the curve;
- A base point $G$;
- Order $n$ of the base point;
• A value of the cofactor $h$ parameter.

The proposed extension to DTLS multicast supports only one-to-many communication pattern, i.e. one multicast sender and many multicast listeners per multicast group. Therefore, the use of a ciphersuit supporting authentication with signature does not allow to use many-to-many communication pattern, which suits our use-case. If many-to-many communication pattern has to be supported, then another extension has to be done, e.g. "server write signature key" should be represented by an array which contains the keying material of all possible senders. However, this scenario is considered out of the scope of this project and no further details are provided.
Chapter 5

Design for Group Key Management

Our contributions to this Chapter begin with formulation of requirements that we apply to key management, selection and analysis of 4 candidate protocols. Three of those are widely used, well known standardized solutions. Then, we select the best suitable candidate and discuss which of the requirements are still not satisfied. We present 3 different options how the protocol can be applied as well as we discuss a possible advanced system setup which might take place in the future. Then, we propose a customized and extended solution satisfying our requirements. In particular, we extend and modify the candidate protocol to satisfy our flexibility requirement and small memory requirement as well as we further optimize a solution for a low communicational overhead. We base our solution on the best candidate protocol with security assured via DTLS established secure session.

Our design is mainly targeted for the key management of the DTLS based multicast, in particular, we are concerned with the group key management to provide confidentiality for group communication and server’s public key management to provide source authenticity for multicast messages send from the back-end server. However, we consider that lower level security protocols can also be managed with the same key management protocol.
5.1 Formulation of the requirements for key management

The DTLS-based multicast draft is not intended to cover the key management part. There is no work yet done in regards to group key management which means that even the requirement document is not yet proposed. Hence, in this section, we formulate the requirements which we apply to key management protocol. Although some of the requirements are inherited from the general requirements from Section 3.2, they are still discussed from the viewpoint of the key management protocol to draw the complete representation:

- **Low computational overhead** is a requirement implied by the constrained multicast receiver devices which have limited computational resources. Hence, the setup should take into account the restricted computational possibility, e.g. many computationally heavy public key operations on such a device are not affordable. The fewer resources required for the key management services the better.

- **Low communicational overhead** is a requirement implied by the constrained multicast receiver devices and low bandwidth capacity of LLN. Thus, the size of the messages exchanged and respectively the number of the messages exchanged for the key management should be kept minimal.

- **Small storage** requirement is implied from the constrained memory resources in a device. The less memory is required for the security the better. Thus, it is desirable to reuse as much functionality as possible to reduce the footprint size and to store minimum number of keys. Furthermore, the protocol itself can have a complex or simple structure, thus, a simple protocol requires less memory for its implementation.

- **Replay protection** mechanism is required to prevent the attacker from reusing eavesdropped messages.

- **Key distribution/update group secrecy** implies that group keys should be available only to the multicast group members, i.e. keys should be distributed/updated confidentially. With regards to the initial key distribution, this requirement also implies authentication of a joining node by the key distribution server.

- **Key distribution/update source authenticity** is important to assure that the keying material is genuine and distributed by the trusted authority. Since in our scenario, the back-end server is a multicast sender, it acts as a trusted authority as well. According to our security model discussed in Section 3.1, the back-end server is invulnerable, while the other
receiver devices are easy to compromise. Such devices do not trust each other in regards to the group keying material to join multicast, therefore, they should trust only the keying material originated from the server.

- **Key update backward secrecy** implies that a late joined group member should not be able to read past group communication. For example, the previously distributed old commissioning data should not be available to the new group member.

- **Key update forward secrecy** implies that an evicted group member should not be able to read packets sent to this group after it has left.

- **Scalability** towards the number of receivers - the key management for the security mechanism should not significantly limit the scalability of the multicast.

- **Reliability** is important since the key update procedure is loss sensitive. If a packet containing a new keying material is lost, then a receiver will not be able to read all the packets protected with this keying material. Therefore, the key update messages are usually delivered reliably.

- **Flexibility** - there are several desirable features to be provided by the group key management protocol. One such desirable feature is to enable a prospective group member to initiate a join operation as well as to inform about the group it wants to join. It is important for convenience of the system that new devices initiate the join operation and not the server pushes the keys. There are also some scenarios where a joining node could get the group identifier through the local discovery service and therefore, the new node can request a join to a specific group. However, the back-end server takes a final decision to grant a permission to join the requested group only if the requester is authorized.

However, the pursuit of the above requirements should not compromise or reduce security level of the system. That is, the general security principals should be followed, e.g. key separation principle. The *key separation principle* states that distinct keys should be used for different algorithms and modes of operation [40].

## 5.2 Fulfillment of the key management protocol requirements

In this section, we first discuss motivation and then present the comparison of the selected candidate group key management protocols.
5.2.1 Selection of the protocols

First of all, the existing DTLS handshake protocol cannot be used to establish group keying material, since it can only be used for generation of pairwise keys. Moreover, a public key should be generated by the back-end server and distributed as it is, since the keys for the ECDSA signature are generated in a special way and cannot be derived, for example, from Diffie-Hellman exchange.

The current draft version [55] only references Group Secure Association Key Management Protocol (GSAKMP) [42] as a prospective candidate for amendment to fulfill the requirements of the DTLS multicast. Therefore, we first of all select this protocol to be analyzed. GSAKMP is a protocol which provides a framework for secure group key management with various options on the Internet. Next, GDOI [93] protocol is the protocol designed with the special focus on support of the group key management for IPsec multicast [92]. The Cisco secure multicast implementation of IPsec multicast utilizes the GDOI as group key management protocol [27]. IPsec and TLS are both comparable secure tunneling protocols. Therefore, IPsec multicast group key management protocol is selected to be considered as well. Multimedia Internet KEYing (MIKEY) [10] is another group key management protocol utilized for management of secure real-time application keys. MIKEY is typically used for SRTP [16] session keys management. We select it since MIKEY is designed with the aim to be simple, to minimize memory use, communicational overhead and provide flexibility. There is also a proposal of an Adapted Multimedia Internet KEYing (AMIKEY) for LLNs which is very appealing to our case since we are working in the constrained environment, and thus, it is selected to be considered. Even though the protocol is in the status of expired draft in the year of 2013, the standardization can be resumed.

5.2.2 Protocols Comparison

These protocols are selected to be compared according to our requirements for multicast key management presented in Section 5.1. Table 5.1 presents comparison of the selected protocols. For certain requirements such as computational and communicational expenses, scalability and flexibility, we distinguish between join and rekey events handling. Rekey is considered either a Group Member (GM) eviction operation or a key update. Regarding the computational overhead, we compare the number of messages required as well as the total overhead in bytes to distribute a server's public key for ECDSA-224 (28 bytes) and a group encryption key (16 bytes) to a single group member with each protocol. Even though GSAKMP and GDOI have an option to perform a rekey
5.2 Fulfillment of the key management protocol requirements

operation over multicast to a whole group, we do not take this into account in the communicational overhead comparison. The multicast rekey would require a reliable multicast which is considered to be not affordable for our constrained network. We calculate overhead preserving the minimal mandatory setup allowed for each protocol. Considering computational overhead, we compare the computation required by a GM device, since in our setup, GM is a device with constrained resources. The primary focus is put on how many public key cryptography operations are required due to their high cost. We also show which replay protection mechanism is used by the protocols.

As the result of the comparison, all of the protocols in one or another way satisfy most of the requirements. GDOI and GSAMKMP apply at least twice higher communicational overhead to distribute the keying material as well as the number of exchanges required is bigger than the one of MIKEY/AMIKEY. Also, only MIKEY/AMIKEY does not involve public key cryptography as compulsory part. However, MIKEY/AMIKEY does not have an option of rekey via multicast and it requires unicast communication with each GM which is also reflected in the key update backwards and forward secrecy parameters. All of the protocols do not discuss the reliability of transmission which is left out to be dealt with the other protocols in a protocol stack. AMIKEY mentions that reliability should be provided but does not discuss it any further. Replay protection in GSAKMP can be provided by means of sequence numbers or nonce, while GDOI relies on nonce only and MIKEY/AMIKEY relies on timestamp or, optionally, on a counter. Regarding the flexibility requirement, MIKEY does not provide a possibility for a GM to initiate a join operation, while other three protocols support this important for us property. Overall, MIKEY/AMIKEY is a much more simple protocol which is important for the low bandwidth networks with constrained devices. As well, it implies much less expenses computationally and in terms of per message traffic. Given that in our application group keys are supposed to be used for a long period, the groups are considered to be about 100 GM and, once formed, groups are not supposed to change membership frequently. The lack of an option to perform a rekey via multicast using a mechanism such as LKH does not overweight all the benefits discussed above. Moreover, MIKEY/AMIKEY is designed to generate multiple pairwise keys from a single key, called TGK. Such a feature is also valuable for us since it improves efficiency of pairwise key management needs of our system, i.e. management of pairwise keys for link layer and routing layer security. Therefore, MIKEY/AMIKEY is the best candidate to be selected. However, AMIKEY is tailored to comply with LLNs restrictions as well as it also allows the keying material to be pulled by a GM unlike MIKEY. Thus, this protocol is selected as the best suitable candidate for key management.
Table 5.1: Key Management Protocols Comparison

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Push</th>
<th>Pull/Push</th>
<th>Rekey</th>
<th>Join</th>
<th>Protocols</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements</td>
<td>❌</td>
<td>❌</td>
<td>❌</td>
<td>❌</td>
<td>GSAMP, MIKEY, AMIKEY, GDOI</td>
</tr>
<tr>
<td>Computational Overhead</td>
<td>1/180</td>
<td>1/121</td>
<td>1/89</td>
<td>1/121</td>
<td>1/180</td>
</tr>
<tr>
<td>Communication Overhead</td>
<td>0/624</td>
<td>0/624</td>
<td>0/624</td>
<td>0/624</td>
<td>0/624</td>
</tr>
<tr>
<td>Reliability</td>
<td>Complex</td>
<td>Simple</td>
<td>Complex</td>
<td>Simple</td>
<td>GSAMP, MIKEY, AMIKEY, GDOI</td>
</tr>
<tr>
<td>Simplicity</td>
<td>Complex</td>
<td>Simple</td>
<td>Complex</td>
<td>Simple</td>
<td>GSAMP, MIKEY, AMIKEY, GDOI</td>
</tr>
<tr>
<td>Scalability</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>GSAMP, MIKEY, AMIKEY, GDOI</td>
</tr>
<tr>
<td>Repair Protection</td>
<td>None</td>
<td>TimeStamp counter</td>
<td>None</td>
<td>TimeStamp counter</td>
<td>GSAMP, MIKEY, AMIKEY, GDOI</td>
</tr>
<tr>
<td>Communication Overhead</td>
<td>1/216</td>
<td>1/216</td>
<td>1/216</td>
<td>1/216</td>
<td>1/216</td>
</tr>
<tr>
<td>Computational Overhead</td>
<td>0/192</td>
<td>0/192</td>
<td>0/192</td>
<td>0/192</td>
<td>0/192</td>
</tr>
</tbody>
</table>

- The number of exchanges required to join and rekey a single device is presented in the table. The numbers are calculated for the minimal setup with mandatory messages and payloads. The numbers presented in the table are the least possible. In GSAMP, the variables have variable length, and thus, are not counted during the calculation.
- Data is presented in the following format: number of exchanges / number of bytes required for these exchanges.
- Computational overhead is compared according to the number of public key operations involved in the protocol.
5.3 Options of Key Management with AMIKEY

From Table 5.1 it is clear that AMIKEY does not ensure reliable data transfer. However, reliability is required, in order to ensure that the keying material reaches the destination. It is suggested that AMIKEY protocol can be used on top of other message signaling protocols [8]. Therefore, we suggest to send AMIKEY messages over CoAP since it is available by default. In this case, the main purpose of CoAP is to enrich the AMIKEY with reliability. Next, receivers of such a CoAP message containing an AMIKEY packet have to be informed that they should handle payload in an appropriate way. The CoAP header could have an option field denoting the content of the payload. Thus, we can introduce a special content type value to inform that an AMIKEY message is encapsulated.

Considering that reliability for AMIKEY packets is ensured with CoAP, there are 3 possible options of setup which are depicted in Figure 5.1.

![Figure 5.1: Options of key management setup with AMIKEY](image)

In general, the AMIKEY protocol provides all the necessary mechanisms to perform the key management securely with no need in any other additional means, i.e. a separate secure channel can be established to exchange keying material. In this case, the protocol stack used for key management includes AMIKEY on top of the CoAP and UDP as a transport without DTLS between CoAP and UDP. However, in our system, DTLS is always available as a building block for security, thus, there is a second option, when security is ensured with DTLS and security guaranties are not provided with AMIKEY. With this option, the protocol stack includes AMIKEY without security on top of CoAP, then DTLS and UDP are considered as a transport. Hence, the already established DTLS session can be reused for providing secure key management exchanges for AMIKEY.

Even a third scenario is possible, such a scenario assumes a larger setup where
several back-end servers are present which belong to different companies. These servers manage their devices, while all of the devices are still in the same network. Hence, a separate network management entity manages routing of all the devices in the network without trusting the back-end server, i.e. RPL secure routing should be configured by the network management entity protected from the back-end servers. Depending on the infrastructure, the traffic might have to pass through back-end server. Assuming that a direct DTLS connection from lighting devices to network management entity is not possible, the keying materials for secure routing still have to be securely transferred from the network management entity to the lighting devices ensuring that no one except the lighting devices can read it. Thus, the devices in the network can have an AMIKEY application establishing the end to end channel with the network management entity. This traffic can be transmitted transparently through the back-end servers, and back-end servers at the same time might maintain secure channels with the managed devices and with the network management entity. Hence, in this case, the back-end servers can authenticate the network management entity and let the encrypted traffic pass to their devices. This option is depicted in Figure 5.2. Such a scenario can be applicable in the future when the system will involve different companies to provide outdoor lighting to a city. But for now we assume all the devices in the network belong to a single company. Thus, a back-end server manages devices and performs the responsibilities of the network management entity at the same time. Therefore, a separate secure channel at the higher level is not needed.

![Figure 5.2: Advanced scenario](image)

Considering the two possible options from the viewpoint of architecture, the
5.3 Options of Key Management with AMIKEY

Lightweight Machine to Machine (LWM2M) [67] architecture is taken as a reference architecture for client-server communication, where CoAP and DTLS are the key protocols for applications and securing the UDP transport as well. The key management is considered to be added as a part of a device management functional interface in the scope of the LWM2M architecture. In LWM2M architecture, security is ensured with DTLS which means that all the device management operations are secured with DTLS. The key management, in this case, is considered as a part of management traffic and therefore, in order to follow a single common architecture for the complete device management, it is unsuitable to maintain a separate secure channel for every type of communication. It is reasonable to secure key management traffic with the same DTLS session which is used to secure the rest of the device management data. Hence, the option where the AMIKEY is secured with DTLS satisfies the architecture, unlike the option where security is ensured separately with AMIKEY.

Furthermore, the reuse of the DTLS supplementary minimizes the memory requirement, since in this case no security mechanisms are maintained solely for AMIKEY but instead existing mechanisms provided with DTLS are reused. As far as the device management is performed securely, a DTLS handshake is carried out to establish secure DTLS session for the device management, therefore, all the management including key management is to be performed over the same DTLS session. Thus, there is no need to maintain separate key(s) to ensure confidentiality and authenticity for the key management exchanges. The key separation principle mentioned in Section 5.1 is not violated in this case since the session key is not reused for the different algorithm but instead additional traffic is transmitted over the same session. However, if a session key is a long term key which is the fact for our constrained network and since this key is frequently used for data protection, then the probability to break this key increases. Therefore, session key update should be enforced regularly.

Considering compromise of a device, when an attacker physically accesses a device in the group, it implies the same level of harm as if separate keys would be used for the key management exchanges and other device management traffic protection. In both cases, an attacker gets access to the secret keys stored in the compromised device and hence, the attacker can read messages previously secured with the keys in his/her possession.

Reduction of the AMIKEY security not only implies smaller memory requirement but also some of the payloads are to be removed from the message. As a result, a communicational overhead can also be reduced which is discussed in detail in the following section.
5.4 Design of the AMIKEY based Key Management Solution

In general, on the architectural level in our system we consider that the join operation is requested (pulled) by a joining device (client) while the rekey is usually pushed from the back-end server and, optionally, the server can wait for a response from the group member if mutual authentication is required. However, it is also possible that a group member triggers the rekey operation and the back-end server takes a decision to perform a key update. Hence, the key management procedures at AMIKEY are half round trip or single round trip exchanges. Both pull and push exchanges are depicted in Figure 2.9. The joining node requests message informing for which security protocols the device wants to establish keys. If no other services previously performed identification of the node, then a unique identifier is also sent with the request, which is needed to provide the server with the information to select an appropriate key shared with the node. Then, the server replies with the keying material for the requested protocol including the corresponding security policies to be used. The detailed description is presented in Section 2.4.4. The rekey message from the server contains the new keys and/or updated policies.

With our design we leave a potential compatibility of our version of AMIKEY with a generic AMIKEY, e.g. if in the future the complexity of the system will grow to the level of the scenario with several back-end servers and a separate network management entity as discussed in Section 5.3. Furthermore, in our scenario a back-end server acts as a trusted party as well as the back-end server distributes only its own public key. Hence, there is no need in Public Key Infrastructure (PKI) like X.509 [28] and public keys can be transmitted the same way as the group key. However, in a more complex case where a separate trusted third party is responsible for the public key distribution to several distinct parties, PKI like X.509 [28] should be in place. In the following subsections, we define a set of payloads which must be implemented in our system and we discuss why other payloads are not necessary for our system. As well, we discuss the use of the protocol’s fields.

5.4.1 Reduction of Superfluous Functionality

The reuse of the DTLS for security implies that certain functionality of the AMIKEY is no longer needed, and therefore, we propose to eliminate the unnecessary payloads to minimize the size of the message. Below we present the discussion, the arguments and decisions in regards to Common header, Times-
5.4 Design of the AMIKEY based Key Management Solution

tamp and Identification payloads apply to all messages sent from server and client devices. The decisions regarding the Verification payload apply to the client, while RAND, Key data transport and Security policy payloads apply only to messages initiated by the server.

Regarding the Common header payload, we leave it untouched as shown in Figure 2.10. However, we repurpose the use of the fields as discussed in Section 5.4.2.

The Time Stamp payload is no longer needed to be sent with any message since DTLS provides a replay protection mechanism via counters included into each record. Therefore, all the key management messages inherit the replay protection from the DTLS secure channel.

The Identity payload is considered optional in AMIKEY but it still might be needed, if no other identification is performed before the key management exchanges. Considering the use of the established DTLS session for protection of the key management, the Identity payload is not needed at all. Prior to the key management exchanges, the identification is performed with the DTLS handshake as a part of the secure session establishment.

The Verification payload carries a MAC calculated on the whole message for authentication of the client device. This payload is superfluous as well, since authentication of the whole message is ensured with DTLS, e.g. AES-CCM mode. Hence, we propose not to implement it in our version of the protocol.

The RAND payload is only included in the messages sent by the server and it is used to transmit a random value to introduce freshness to the keys derived from a TGK [10]. It is used as a countermeasure to the offline pre-computation attacks [10]. The random value is transmitted only with an initial message from the server, i.e. during the join operation. Then, during key update operations, the same random value is reused. However, in this project we are concerned with the management of keys required to facilitate group communication which are not supposed to be generated from TGK, and thus, we also propose that if the group communication keying material is to be distributed with the protocol, then the RAND payload should not be added even to an initial message of the server.

The Security Policy payload is included into the server initiated message during the join operation and it can optionally be present in the key update message if the policies require an update.

Similarly to the Verification payload, the Key data transport payload is not needed any more since it is used by the server to provide confidentiality of
keying material and authentication of the whole message. The DTLS secure channel provides these properties and DTLS even ensures confidentiality of the whole message as far as the key management message is a payload of the record layer. Instead of being encapsulated in the Key data transport payload, the keying materials presented with a Key data sub-payload instance per a key are included directly after the last Security policy payload. The Key data sub-payload format remains intact as well as the salt data and key validity data remain optional. However, we suggest that if not predefined by default, the key validity is highly likely going to be included as defined in [8], in order to facilitate enforcement of the key update procedures. If the server strictly follows the key update schedule, the multicast listeners can trust the server that the key update will be enforced when it is needed. In this case, the key validity data can be skipped.

5.4.2 Repurpose and Protocol Parameters Extension

In the Common header payload, we repurpose the usage of CSB ID field and the verification (V) field. The Version field should contain the version of AMIKEY which informs that our custom version is used. The verification field is set only by the server. If the server requires mutual authentication which implies a reply from the client, then the verification bit is set. We consider that the join operation is initiated by the client, and the rekey is half round trip initiated by the server, thus this flag is not going to be used. The CoAP header already has an instrument to signal that the reply is needed. However, the verification field is only 1 bit long and there is no strong motivation to remove it from the packet. The CSB ID field in AMIKEY is only used to detect duplicate messages which could appear due to retransmissions in case of message losses. Even though the CoAP header contains the Message ID field which provides the same functionality, we leave the CSB ID field due to the fact that in our system it is going to have the same functionality as in MIKEY but to identify multiple virtual networks at the RPL level. However, this part is out of the scope of this project, and thus, no further discussions are provided. In this project, we assume that we just keep this field for further use explained in another project and for compatibility with general version of the protocol. The Common header payload should also contain the IDs of the security protocols for which keys are requested (in case of a request message) or distributed (in case of a server initiated message). AMIKEY has a list of predefined protocol IDs to which we add an ID for the DTLS multicast protocol to the list. Thus, the ID of the DTLS multicast protocol appears in the Common header payload of the server message and informs that the keys for the DTLS multicast protocol for it should be expected in the corresponding payloads as well as policies can appear if mentioned. The extended list is presented in Table 5.2.
In regards to security policies, we add a security policy parameter, namely, Group ID to identify a multicast group for which the keying material is distributed. As well, we propose to minimize the number of policies to be distributed during the join operation by setting up in each node a default security policy parameters set for each protocol, in particular, we are interested in the DTLS multicast protocol. This optimization further improves the low communicational overhead requirement. Table 5.3 shows a set of the default parameters which can be reconfigured if needed. The set of parameters is presented in AMIKEY and we only define the default values for them, and we add a Group ID parameter. Taking into account that key length and the encryption and authentication algorithm types are set by default as well as encryption is enabled, only a Group ID parameter has to be distributed.

<table>
<thead>
<tr>
<th>Type</th>
<th>Meaning</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Encryption alg</td>
<td>AES-CM-128</td>
</tr>
<tr>
<td>1</td>
<td>Encryption key len</td>
<td>128</td>
</tr>
<tr>
<td>2</td>
<td>Authentication alg</td>
<td>ECDSA-224</td>
</tr>
<tr>
<td>3</td>
<td>Authentication key len</td>
<td>224</td>
</tr>
<tr>
<td>5</td>
<td>Encryption on/off</td>
<td>On</td>
</tr>
<tr>
<td>6</td>
<td>Group ID</td>
<td></td>
</tr>
</tbody>
</table>

Table 5.3: Security policy parameters for DTLS multicast

In regards to the Key data sub-payload, we need to convey information that a group key or public key is contained in the payload. Therefore, the Type field is enriched with new possible values, such as a group traffic encryption key (GTEK) and a public authentication key (PAK). However, we assume that salt data is not going to be needed in this case. According to the protocol [8, 10]
salting data is included if the mode of operation requires it and, in our case, we assume AES-CM for encryption and ECDSA for authentication. As stated in NIST SP800-36a [35] the initialization vector for AES-CM does not need to be secret, thus salt is not needed. Table 5.4 shows possible values refined.

<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>TGK</td>
<td>0</td>
</tr>
<tr>
<td>TGK+SALT</td>
<td>1</td>
</tr>
<tr>
<td>TEK</td>
<td>3</td>
</tr>
<tr>
<td>TEK+SALT</td>
<td>4</td>
</tr>
<tr>
<td>GTEK</td>
<td>5</td>
</tr>
<tr>
<td>PAK</td>
<td>6</td>
</tr>
</tbody>
</table>

Table 5.4: Keying data types for Key data sub-payload

5.4.2.1 Addition of Missing Functionality

According to our flexibility requirement, a joining device might be advertised in the Group ID through some neighbor discovery services which are out of the scope of the project. Hence, it is desirable that this device could inform the back-end server with the Group ID which it wants to join. However, a final decision to authorize the device to join the group is taken by the back-end server. AMIKEY does not provide this functionality, thus, we design a new Multicast Group ID payload in a generic way to provide a possibility to select a number of groups to request a keying material for. For each distinct protocol, a separate payload should be included. The payload is presented in Figure 5.3. It can be added after the Common header payload in the client’s request message.

Figure 5.3: Multicast Group ID payload

If the client is authorized to join the selected group, the server includes the corresponding Group ID in the policy. Otherwise, the server includes the ID of a Group, the client is authorized to join and corresponding keying material is included into the message.
5.4 Design of the AMIKEY based Key Management Solution

5.4.3 Node Join Discussion

Considering the join operation of a new node, we assume that a pairwise DTLS session is already established. The new node could be informed through local discovery service about the multicast group ID it can join. Then, the node assembles the AMIKEY packet placing the DTLS multicast protocol ID in the Common header payload. If only DTLS multicast keys are to be requested, then no other protocol IDs are included in the header. The Data type in the header is selected as a request packet. If the node already knows the group it wants to join, the group ID is added with the Multicast Group ID payload. Then, AMIKEY packet is appended with the CoAP header which contains a Content-type option denoting that the payload should be handled as an AMIKEY packet. The assembled CoAP message secured with DTLS using a previously established session key and DTLS record is sent to the server.

Upon receipt, the server decrypts the record and authenticates that the record was generated by the joining node. After authentication, the server checks which protocols the keying materials are requested for. If a multicast group ID is present in the packet, the server verifies that the requester is authorized to join the group. Otherwise, the server just looks up the group ID the requester is supposed to join. Assuming that only a DTLS multicast group keying material is requested, the server assembles AMIKEY packet containing the Common header, Security policy payloads and two Key data sub-payloads. The Common header payload Data type field is set to inform that the message is initiated from the server. It also contains the ID of the DTLS multicast protocol which is mapped to an ID of the policy for the protocol. The policy itself is included in the Security Policy payload. Since the majority of the parameters are predefined, the policy contains a single parameter denoting the multicast group ID. Next, the two Key data sub-payloads are included subsequently. One payload has a type GTEK which bears the group traffic encryption key and another payload has a type PAK which bears ECDSA public key of the server used for source authentication in the particular group. Then, the AMIKEY packet is appended with the CoAP header including the Content-type option which informs the receiver about the content type. Afterwards, the CoAP message is secured with the same session and DTLS record is sent to the joining node.

Upon receipt, the joining node decrypts the record and authenticates that the record was generated by the server. After authentication, the node retrieves the group ID and keying material for the group communication which includes group encryption key and public key. As a result, from now on the node is authorized and able to listen to the particular group communication.
5.4.4 Node Eviction Discussion

Node eviction is a specific case of the rekey operation, key update is also specific type of a rekey as well. In node eviction operation, the new keying material is distributed to all the members except for an evicted one. In this case, the server distributes keying material via unicast to each group member secured with the established pairwise DTLS session.

Considering the node eviction operation, the server assembles AMIKEY packet containing the Common header payload and the Key data sub-payload. If security policy is changed or if the ID of the multicast group is needed to be mentioned, the Security Policy payload is added as well. The Common header payload Data type field is set to inform that the message is initiated from the server. Next, the Key data sub-payload is included which bears the group traffic encryption key. The public key may remain the same, since node eviction does not imply the update of a public key. Then, the AMIKEY packet is appended with the CoAP header including the Content-type option which informs the receiver about the content type. Afterwards, the CoAP message is securely transmitted within the same DTLS session to the joining node.

Upon receipt, a group member decrypts the record and authenticates that the record was generated by the server. After authentication, the node retrieves new encryption key for the group communication and, optionally, the group ID. From now on the group member should use the updated keying material.

5.4.5 Discussion

Such a node join and eviction operations are not efficient since they require an unicast exchange with each group member for a group key update in order to comply to backward and forward secrecy requirement. However, in our scenario group membership changes rarely as well as multicast groups are supposed to be about 100 nodes to manage them easier.

A more efficient solution involves the use of LKH [89] over multicast which is used in well known group management protocols like GDOI [93] and GSAMKMP [42]. Due to our requirement for the key exchanges reliability, the multicast communication has to be reliable which is not the case for the constrained networks. In our case, the constrained network cannot afford reliable multicast due to the limited bandwidth capability and lossy channel. Thus, the group key management solution with the use of LKH over multicast is not applicable to our case. Moreover, for the constrained networks with frequent membership variable key
validity periods can be introduced as discussed in [22, 88].

5.4.6 Security Considerations

The fact that we do not use the security mechanisms of AMIKEY in our key management solution does not weaken the level of security assured with the solution. The DTLS record layer is used instead to protect the key management exchanges. The key management protocol is placed at the application layer and it is used as a payload of the DTLS record protocol as well as it does not alter any functionality of DTLS. Therefore, the key management protocol inherits the security of the DTLS record layer which is well standardized, widely used and shown to provide trustworthy security over the time. We propose to reuse an already established pairwise DTLS session for the key management. Taking into account that the key management information is very important, the session key update should be strictly enforced by the server. There is also a possibility to maintain a separate DTLS session exclusively for key management needs, however, maintenance of the separate session requires an additional DTLS handshake to be performed for the session establishment. The DTLS handshake is assumed to be expensive and it is desirable to avoid it, since it is not crucial for the security in this case.

5.4.7 Summary

Our key management solution for DTLS secure multicast is based on the AMIKEY protocol placed on top of the CoAP. Both join and rekey operations are performed over unicast. The join operation is initiated by a joining device with an option to inform the group controller about the multicast group ID the device want to join. The join request message contains the Common header payload and, optionally, the Multicast Group ID payload which is not presented in basic AMIKEY version. The response message in the join operation is initiated with the group controller. It contains the following payloads: Common header, Security Policy, Key data sub-payload. The RAND payload is not required to be sent if DTLS multicast is managed only. The rekey operation is generally initiated by the group controller and it contains the following payloads: Common header, Security Policy (if it was changed) and Key data sub-payload.

The reliability is ensured with CoAP through the retransmission of not acknowledged packets. Security is ensured with DTLS secure unicast channel, which is supposed to be already established to facilitate device management in general.
In this section, we present the developed for Philips prototype for demonstration of source authentication with ECDSA for DTLS multicast. This prototype is developed as a proof of design shown in Section 4.2. We develop the prototype on top of Contiki OS v2.7. The starting point was a prototype of DTLS multicast provided by Philips where AES in CCM mode is used for confidentiality and authentication. In this project we enrich the existing prototype implementation of DTLS multicast with ECDSA for authentication of DTLS records. The implementation of ECDSA is out of the scope of the master thesis, therefore, a publicly available ECDSA implementation is taken for development of prototype. The server and clients are both represented with equally constrained devices in our prototype.

6.1 Point of Reference Prototype for DTLS Multicast

The prototype of the DTLS multicast, which is our starting point, is implemented for Contiki OS v2.6, and therefore, it is possible to run simulations on the Cooja simulator, the Cooja target should be selected at compile time. The devices can be simulated including LEDs and the communication between the
devices can be observed. The prototype allows sending multicast turn on/off the light commands over CoAP secured with DTLS based multicast. The LEDs light up if lights on command is received and they are turned off if lights off command is received. The prototype is based on the publicly available implementation of TinyDTLS 0.4 [18] by Olaf Bergmann. TinyDTLS 0.4 is a DTLS implementation for constrained environment to secure unicast communication. The ciphersuite supported by the implementation is TLS_PSK_WITH_AES_128_CCM_8. Respectively, the multicast prototype implementation supports the AES_128_CCM_8 ciphersuite which means that the same key is used for confidentiality and authenticity protection of DTLS records. The keying material is preloaded to all the multicast group members. A multicast address and a UDP port number are preconfigured. The multicast listeners join the multicast group by starting to listen to the multicast address on the particular port. The prototype accepts multiple listeners in a group. The sender device has an option of sending a lights on or lights off command which can be triggered from the serial interface. In order to send a command, first CoAP packet is assembled. Then, DTLS record header is assembled and the CoAP packet is authenticated and encrypted with AES-CCM-8, thus, 8 bytes of authentication information are added to the encrypted text. Next, the UDP datagram is sent to the multicast address. Upon its receipt, all listeners decrypt the DTLS record, analyse the CoAP packet and execute the command. As a result, lights are on or off, depending on the command. Both multicast sender and multicast listeners are implemented on DTLS client software.

6.2 Authentication with ECDSA

In our prototype, we assume that the public keys are preloaded as well as the shared key in the initial prototype. In order to enrich the prototype with the support of the ECDSA signature for authentication, a publicly available implementation of ECDSA is used. To follow the easiest path, we select the ECDSA implementation incorporated in TinyDTLS 0.5 [32] which is the 0.4 version upgraded by Cetic company [4] with support of TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8 ciphersuite. In this ciphersuite, ECDSA is used to protect authenticity of Diffie-Hellman exchanges at handshake layer. Since it is already integrated to work with Contiki it is supposed to be easier to integrate it to the record layer. Even though this implementation is ECDSA-256 but no3 Kt 224 as it is proposed by design, we still select it since this fact does not influence the demonstration aim of our prototype. The secp256r1 prime curve is used at the implementation. For the real deployment the appropriate implementation should be chosen with regards to the requirements, since different optimization goals might be pursued with distinct implementations, which makes big difference to usage of
RAM, ROM and execution speed. In our case, the signature size is expected to be 64 bytes. However, in the implementation, which we have taken from TinyDTLS 0.5, the signature is transmitted including Abstract Syntax Notation One (ASN.1) fields. Moreover, signature itself has variable length of 64-66 bytes, owing to the fact that sometimes the implementation generates $R$ and/or $S$ signature parts with an extra leading zero. In total, signature element is represented with 70-72 bytes instead of 64 bytes: 1 byte for total signature length, 2 bytes to represent length of $R$ and $S$ (one byte for the length of each signature part, respectively), 3 bytes for ASN.1 fields and 64-66 bytes for signature. Therefore, we modify the implementation to minimize signature up to 64 bytes by eliminating the ASN.1 fields and by deleting the extra leading zeros.

Furthermore, in order to reuse the encryption algorithm provided with the available ciphersuite TLS_PSK_WITH_AES_128_CCM_8, we alter the CCM mode to avoid sending the MAC generated with the CBC-MAC mode. In CCM mode plaintext is encrypted with Counter Mode (CM), and then, MAC is generated on the plaintext together with the authentication information. Finally, MAC is also encrypted. In case of DTLS multicast, the authentication information consists of epoch, sender ID and sequence number. Therefore, to minimize the modifications, using preprocessor definitions we disable the MAC verification in CCM mode, when the authentication is provided with ECDSA. Hence, MAC is not anymore transmitted with the record and the encryption is ensured with the AES-CM. As it is designed in Section 4.2, we implement signature generation on the header fields (epoch, sender ID and sequence number) and on the ciphertext to provide authenticity of ciphertext as well. Thus, first the plaintext is encrypted, and then, signature is generated. The signature is appended in the end of the record. Hence, we achieve the AES_128_CM_ECDSA_256 cipher-suite from AES_128_CCM_8. The input for signature verification consists of the header fields and ciphertext. If verification fails, then processing of the received message is terminated.

6.2.1 Experimental Packet Size

In this section we discuss the observations of the packet size achieved during the implementation, however, the sizes of the packets of the real deployment might be different. At least, the size of the signature is supposed to be smaller. During our simulation the packets are sniffed by Wireshark. The sizes of the header in the sniffed packets are presented in Figure 6.1.

The MAC layer header is 15 bytes, owing to representation of the destination and source by 4 bytes and 8 bytes, respectively. IPv6 header is represented by 6 bytes, where dispatch field is 2 bytes and the destination address is 4 bytes,
the rest of the IPv6 header fields are elided. The UDP is compressed to 7 bytes, since only the length field is elided, while a byte of dispatch is added. The DTLS record header is 13 bytes and CoAP packet in total is 11 bytes: 5 bytes header and 6 bytes light control command. In total, length of the frame is 118 bytes, while the maximum size is 127 bytes. Hence, a frame does not need to be fragmented to send a light control command. However, in Contiki OS v2.7 there is a flaw in file contiki/core/net/sicslowpan.c at the line 1450 which does not allow to send 118 byte long frames without fragmentation. The flaw is caused by deducting MAC header twice from the frame size at the conditional statement responsible for a decision in regards to fragmentation. We fix this bug and enable transmission of frames up to 127 bytes without fragmentation.

6.3 Experimental Scenario

In our experimental scenario, we set up 4 nodes with the DTLS client software. All the nodes are equally constrained nodes which run 6LoWPAN. The simulation is performed over a single hop only. One of the nodes acts as a back-end server, hence, this node takes the responsibility of the back-end server to send light control commands to the multicast listeners. The remaining 3 nodes act as multicast listeners, however, only 2 of them have the correct group keying material preloaded, while the third node just listens to the multicast address without having the correct keying material. As a result, only the two client nodes having in possession the keying material are able to execute the commands and to turn on or off the LEDs correspondingly. Figure 6.2 represents the aforementioned scenario which is successfully simulated in Cooja simulator.

6.4 Implementation in Hardware

For the experimental run in hardware, the STM32W108C-SK constrained devices are used with the board MB851 rev D [85]. Such a device has a 32 bit ARM family Cortex-M3 processor [84]. The STM32W108C-SK device has ROM
256 KB, RAM 16 KB and predefined 1 KB for stack memory. The device also has one functional button and two LEDs.

We program the button press event to trigger the function to send multicast turn on and off the light commands. Hence, a device can act as a back-end server, sending the light control commands. Furthermore, the LEDs are programmed to turn on or off during the CoAP packet processing, depending on the received command. Thus, the reaction of the listeners on the received command can be observed.

To compile the software for the STM32W108C-SK device in Contiki 2.7 the "mbxxx" target should be selected. The command to compile and flash the software to a device is presented below:

```
make TAGRET=mbxxx STM32W_CPUREV=CC dtls-client.upload
```

However, 1 KB of stack memory, as it is initially set up in Contiki for STM32W108, is not enough for the execution of ECDSA signature. The device reboots because of the stack pointer getting out of the physically accessible memory bounds. Therefore, we extend the stack size up to 2.5 KB in the settings of the OS in cpu/stm32w108 directory which leaves only 13.5 KB of RAM for all global and static variables.
In our experimental run on STM32W108C-SK constrained devices, the sender successfully sends the DTLS protected multicast with AES_128_CM_ECDSA_256 ciphersuite. The client devices are able to successfully receive, verify signature and decrypt the CoAP packet as well as successfully execute the light control command. The experimental scenario explained in Section 6.3 is successfully tested in hardware as well.

6.5 Memory footprint

In this section we present the code size of the prototype. In order to reduce the RAM usage, we eliminate the buffer allocated by DTLS implementation for HMAC context information. The buffer comprises 2520 bytes. Due to the fact that the HMAC is not used with our prototype, we save this RAM area to avoid shortage of memory. Table 6.1 shows the ROM and RAM memory consumption by the DTLS multicast compiled with Contiki OS. The RAM consumption is represented with the memory usage by global and static variables in the program. The Table does not reflect the usage of stack, while the total size of the memory segment allocated for stack is 2.5 KB. One can assume that the DTLS multicast with source authentication via ECDSA requires about 2 KB of extra stack memory, comparing to the DTLS multicast without source authentication. The numbers in the Table were acquired from the memory mapping file generated during the build of the software. One can observe that the source authentication with this particular implementation of ECDSA increases the ROM size by 4.7 KB only. The influence on RAM is almost not reflected, owing to the fact that this ECDSA implementation does not contain static and global variables. There are mainly local variables allocated on stack. However, this is only an example and the deployment version will highly likely have different numbers.

<table>
<thead>
<tr>
<th>Code</th>
<th>ROM (bytes)</th>
<th>RAM (bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DTLS Multicast</td>
<td>63464</td>
<td>11456</td>
</tr>
<tr>
<td>DTLS Multicast with source authentication</td>
<td>68152</td>
<td>11476</td>
</tr>
</tbody>
</table>

Table 6.1: The memory footprint
This thesis work deals with the secure multicast for the IoT, in particular, the focus is on the IP IoT with CoAP as an application layer protocol and security of multicast is ensured with DTLS. We build our work around a real world business scenario for control of outdoor lighting system by a back-end server to clearly outline the motivation. First, we formulate requirement applied to secure multicast in IoT and pinpoint the missing functionality of the current DTLS secured multicast draft, namely, the absence of the source authentication and the solution for the key management in a multicast group. These two aspects are taken as goals of the thesis.

To enrich the DTLS secured multicast with source authentication, we survey a number of source authentication schemes present in the literature. Then, we select the ECDSA signature and TESLA scheme as the best candidates to proceed with the design of the source authentication for DTLS. We apply both TESLA and ECDSA to the DTLS multicast solution and evaluate the efficiency of both solutions with regards to different communication patterns, namely: frequent and periodic communication. Both solutions are designed providing the minimal recommended by NIST security level in order to minimize energy consumption and overheads without compromising the security. The efficiency of TESLA is better than ECDSA with the frequent communication pattern, while considering a particular periodic communication pattern, ECDSA is more efficient with
 Conclusion and Future Work

In regards to communication and computational overheads. Since in our scenario the communication pattern is periodic, as well as, we assume, it is the case for many scenarios, the ECDSA is selected to be used as an extension for source authentication for DTLS secured multicast. Moreover, we design a "forced key disclosure" adaptation of TESLA scheme to be able to work in the IoT with long downtime period. Since the TESLA scheme is not as rigorously standardized as ECDSA, we enrich it with the cryptographic primitives and select the protocol parameters suitable for our IoT scenario. Finally, we present an extension of DTLS multicast draft with support of authentication with ECDSA at the record layer, in particular, data structures are extended to specify ECDSA and its parameters as authentication mechanism. We also implement a prototype of source authentication with ECDSA for DTLS multicast to provide a proof of concept for the presented design.

In regards to the key management, we present a solution aligned with our architecture to distributed keying material to the multicast group members. First, we formulate the requirements for the DTLS secure multicast implied by our use-case scenario including a special requirement for flexibility of key management protocol, such as a possibility of a joining device to initiate the protocol and propose a group it wants to join. Then, we compare 4 protocols according to the key management requirements and select the AMIKEY protocol as the most suitable one for constrained networks. The AMIKEY implies the least overhead on communication and memory since it is the most simple one. We describe 3 different scenarios how the AMIKEY protocol can be applied to the systems taking into account architectural view point. We minimize the functionality of the protocol to decrease memory requirement as well as overhead on communication. We also describe how the group key and public key can be distributed and specify the default security policies for the protocol to minimize the number of policies required to be transmitted to the devices. The security mechanisms of AMIKEY are omitted since DTLS can be used to ensure security for key management. The reliability of the message exchanges is ensured with CoAP. Furthermore, the protocol is extended to support the flexibility requirement.

The main contributions of the thesis can be summarized as follows:

- Comparison of different source authentication schemes in regards to overheads implied;
- A design of source authentication with ECDSA for DTLS secured multicast including extension of the DTLS data structures;
- A design of source authentication with TESLA for DTLS secured multicast including selection of the cryptographic primitives, protocol parameters as
well as a design of adaptation of the scheme to the periodic communication pattern;

- Comparison of the group key management protocols in regards to the IoT requirements;

- A design of the solution for the key management in the IoT multicast group;

- Implementation of the source authentication with ECDSA for DTLS secured multicast

In regards to the social and ethical aspects, the research is conducted correctly. No social interests are violated during the research. From the perspective of ethics, the contributions and results of the project are not biased, tampered or mutilated. The sources of information are cited to denote the origin. We base our work on the results from the papers published in conferences which can usually be acquired from the publisher. The DTU and Philips digital libraries were mainly used to acquire access to the papers which were not in free access. Considering economical and environmental aspects, the thesis deals with constrained networks where bandwidth is low and devices are low-power. In the project, the minimization of energy consumption is pursued through minimization of computations implied with the solution as well as the overhead on communication is minimized. Energy consumption is directly related to economical and environmental aspects, since energy costs money and prevalent ways to produce energy are not environmental friendly. Hence, the thesis brings benefit to the environment and the economy. Moreover, sustainable energy resources can be used to power IoT devices, e.g. solar panels can be deployed to accumulate energy during the day time.

7.1 Future Work

As a further work, a simulation could be conducted to experiment with the length of the downtime periods to detect when TESLA scheme starts to consume more processing power than ECDSA and compare the communication overheads of the schemes at that moment.

Moreover, there might be a scenario when group members are not trusted, and therefore, multicast without source authentication is not acceptable. Hence, it could be interesting to compare an energy consumption and overheads implied by unicast transmission to each node versus one multicast but with source authentication by ECDSA.
In this thesis, DTLS is considered as a main security building block both for unicast and multicast due to architectural reasons. However, recently IPsec adaptation was proposed for the IoT pairwise communication. The original IPsec supports multicast with source authentication. Therefore, one can be interested in considering the drafts of IPsec for constrained environments from the viewpoint of multicast and source authentication. Furthermore, both solutions IPsec and DTLS can be compared to determine the most efficient one.


