2024-11

SECURING THE CONSTRAINED APPLICATION PROTOCOL FOR IOT USING AN ELLIPTIC CURVE ALGORITHM

Efrem, Belay Ayele

BAHIR DAR UNIVERSITY

BAHIR DAR INSTITUTE OF TECHNOLOGY

SCHOOL OF GRADUATE STUDIES

FACULTY OF COMPUTING

DEPARTMENT OF COMPUTER SCIENCE

MSc THESIS ON:-

SECURING THE CONSTRAINED APPLICATION PROTOCOL FOR IOT USING AN ELLIPTIC CURVE ALGORITHM

By

Efrem Belay Ayele

NOVEMBER 2024

BAHIR DAR ETHIOPIA

BAHIR DAR UNIVERSITY

BAHIR DAR INSTITUTE OF TECHNOLOGY

SCHOOL OF GRADUATE STUDIES

FACULTY OF COMPUTING

DEPARTMENT OF COMPUTER SCIENCE

By

Efrem Belay Ayele

Securing the Constrained Application Protocol for IoT using an Elliptic Curve Algorithm

A Thesis submitted to Bahir Dar Institute of Technology, Faculty of Computing Department of Computer Science for the Partial Fulfillment of Requirements of Master of Science.

Advisor: Seffi Gebeyehu (Ass. Prof)

NOVEMBER 2024

BAHIR DAR, ETHIOPIA

DECLARATION

I, the undersigned, declare that the thesis comprises my own work. In compliance with internationally accepted practices, I have acknowledged and refereed all materials used in this work. I understand that non-adherence to the principles of academic honesty and integrity, misrepresentation/ fabrication of any idea/data/fact/source will constitute sufficient ground for disciplinary action by the University and can also evoke penal action from the sources which have not been properly cited or acknowledged.

Efrem Belay Ayele _____ 04-Nov-2024

Name of the student: Signature Date:

APPROVAL SHEET

BAHIR DAR UNIVERSITY

BAHIR DAR INSTITUTE OF TECHNOLOGY

SCHOOL OF GRADUATE STUDIES

FACULTY OF COMPUTING

Approval of thesis for defense result

I hereby confirm that the changes required by the examiners have been carried out and incorporated in the final thesis. Name of Student Efrem Belay Signature _____ Date Nov 2024.

As members of the board of examiners, we examined this thesis entitled "Securing The Constrained Application Protocol For Iot Using An Elliptic Curve Algorithm". We hereby certify that the thesis isaccepted for fulfilling the requirements for the award of the degree of Masters of science in"**computer science**

Board of Examiners:

Name of Advisor	Signature	Date
Seffi Gebeyehu (Ass. Prof)	_____	Nov 12/2024
Name of External Examiner	Signature	Date
Dr. Sosina Mengistu (PhD)	_____	Nov.2024
Name of Internal Examiner	Signature	Date
Dr. Tesfa Tegegne (PhD)	_____	_____
Name of Chairperson	Signature	Date
Dagnachew Melesew	_____	NOV 12/2024
Name of Chair Holder	Signature	Date
Kedir M.	_____	_____
Name of Faculty Dean	Signature	Date
Tesfu T.	_____	NOV 12/2024

Acknowledgements

I would like to express my heartfelt gratitude to the following individuals and groups who have supported me throughout my journey in completing this thesis.

First and foremost, I thank God for granting me the strength and wisdom necessary to overcome the challenges I faced during this process.

I am profoundly grateful to my lovely wife, whose unwavering support and encouragement have been my anchor. Her patience and belief in me have been invaluable.

I extend my sincere thanks to my advisor, Seffi Gebeyehu (Ass. Prof), for his guidance, mentorship, and insightful feedback. His expertise and encouragement have greatly enriched my research experience.

Finally, I would like to acknowledge all of my brothers and sisters for their support and assistance in finalizing this thesis. Your contributions have been instrumental in bringing this work to fruition.

Thank you all for your kindness and support.

Table of Content

List of Abbreviations

CoAP	Constrained Application Protocol
DTLS	Datagram Transport Layer Security
ECC	Elliptic Curve Cryptograph
HTTP	Hypertext Transfer Protocol
6LoWPAN	IPV6 over Wireless Personal Area Networks
IEEE	Institute of Electrical Electronics Engineering
IoT	Internet of Things
IPsec	Internet Protocol Security
IPv4	Internet Protocol Version 4
IPv6	Internet Protocol Version 6
LLN	Low-power and Lossy Network
M2M	Machine to Machine
TCP	Transport Communication Protocol
TLS	Transport Layer Security
UDP	User Datagram Protocol
WSN	Wireless Sensor Network
CA	Certificate Authority
CPU	Central Processing Unit
ECDH	Elliptic Curve Diffie-Hellman
IP	Internet Protocol
PSK	Pre-Shared Key
IETF	Internet Engineering Task Force

List of Figures

List of Tables

ABSTRACT

Seamless communication with WSN devices has facilitated the development of various Internet of Thing (IoT) applications. The information communication technologies being developed for this purpose are currently centered around an adaptation layer, specifically IPv6 over Low Power Wireless Personal Area Networks (6LoWPAN), and are constrained by the Constrained Application Protocol (CoAP). A significant challenge in CoAP communications with internet-integrated wireless sensor networks is ensuring end-to-end security, particularly due to the high computational costs associated with elliptic curve cryptography (ECC) on resource-limited wireless sensing devices. Additional concerns include the incompatibility of end-to-end security with CoAP proxies and the limitations of wireless sensor nodes. The mechanism proposed in this research tackles these challenges effectively. It utilizes a DTLS-based security protocol that facilitates a transparent DTLS handshake with mutual authentication, aimed at reducing the computational load on constrained sensor nodes while offloading intensive ECC computations to more powerful devices. The implementation employs pre-shared key authentication for sensor nodes, along with a security protocol that guarantees both mutual authentication and confidentiality within the wireless sensor network (WSN) environment during end-to-end communications. The outcomes of this research have positively impacted both power consumption and memory usage in WSN devices. The proposed approach can be seamlessly integrated into applications running on internet clients (CoAP clients), and sensors node (CoAP servers). Overall, this system enhances end-to-end security for IoT applications while conserving resources in WSN nodes.

Keywords: DTLS, CoAP, ECC, IoT, 6LoWPAN, and WSN.

CHAPTER ONE
INTRODUCTION

1.1. Background

Over the past few years, advancements in technology have enabled small sensor devices to wirelessly connect with the broader internet. In wireless sensor networks (WSNs), IP technology has traditionally been seen as unsuitable due to its high demands on processing power and memory (Gao et al., 2012). An IP-enabled wireless sensor network (IP-WSN) is a specific type of WSN that utilizes the Internet Protocol (IP) for communication among devices. In this setup, each sensor devices are allocated a sole IP address, enabling direct communication with other nodes along with external devices connected to the internet. The use of IP in WSNs enables a variety of applications that necessitate data to be transmitted over long distances or across multiple networks. For example, an IP-enabled WSN could be used to monitor environmental conditions in remote locations, with data transmitted back to a central server over the internet.

There are several protocols that can facilitate IP communication in WSNs, including 6LoWPAN, ZigBee IP, and WiFi. To accelerate the adoption of these new emerging techniques, several key challenges need to be tackled, including routing, energy consumption, and security, in IP-enabled wireless sensor networks and their applications (Gao et al., 2012). The characteristics of the WSN channel makes the data vulnerable to being modified, injected and eavesdropped (Gao et al., 2012).

Therefore, security is often an important requirement. This study, focuses on securing the constrained application protocol for IoT by implementing Datagram Transport Layer Security (DTLS) protocols using a pre-shared key cipher suite with an Elliptic Curve Cryptograph (ECC) algorithm for end-to-end security with regard to saving resource of the constrained devices.

The study introduced a security scheme based on the ECC public key cryptography algorithm, designed to operate over normal communication stacks that provide UDP/IPv6 networking for Low Power Wireless Personal Area Networks (6LoWPAN).

1.1.1. The Internet of Thing (IoT)

The Internet of Thing (IoT) is a rapidly growing field that involves joining unremarkable things for example sensors, appliances, and vehicles to the internet to enable new applications and services (A. Khan, 2016). IoT devices are typically small, low-power, and resource-constrained,

which presents unique encounters related to security and privacy (Mehdipour, 2020).

Cisco has projected that the IoT device will increase to 50 billion in 2020 (Cisco IBSG, 2011). IoT devices are isolated, communicating with one another and linked to the global network where they open themselves to attack (A. Khan, 2016). More sensitive data more risk to an individual or an enterprise. So, I need to protect this information's-based value to deliver security in IoT. The biggest concern is that to bring up the implementation of the Internet of Things but without Compromising privacy, security and integrity issues (M. Gamundani, 2014).This proposed work presents DTLS based on end-to-end security in CoAP communication for the Internet of Thing involving appropriate ECC cryptography algorithm and pre shared cipher suit.

1.1.2. Feature and Challenges' of IoT

A. IoT Features

The IoT is the interconnection of diverse networked entities, including sensors, actuators, mobile devices, and more. (Brachmann et al., 2012). The features that IoT needs to support are device heterogeneity, scalability, energy-optimized solution, ubiquitous exchange of data, self-organization capability, semantic interoperability, data management, embedded security, privacy-preserving mechanism and localization and tracing functionality. (Miorandi et al., 2012). One notable feature of the IoT is its focus on energy-optimized solutions. For various IoT devices, a key limitation is to minimize the power consumed during communication and computing tasks. (Miorandi et al., 2012). Subsequently, the go-to-plan arrangements that aim to optimize energy consumption (even at the expense of performance) will end up increasingly appealing (Miorandi et al., 2012).

B. Challenges of IoT

Due to the rise in Internet-enabled services and devices greening of IoT is also thought to be a key challenge to minimize the energy consumption i.e. to make the network devices more energy efficient(R. Khan et al., 2012). Additional challenges contain range allocation as devices requires dedicated spectrum to send data over the wireless standard, explicitly referring to and identity management to manage and assign unique identity.

One of the biggest challenges that prove to be a hindrance to IoT growth is security as shown in Figure 1.1 (A. Khan, 2016; Roman et al., 2013).

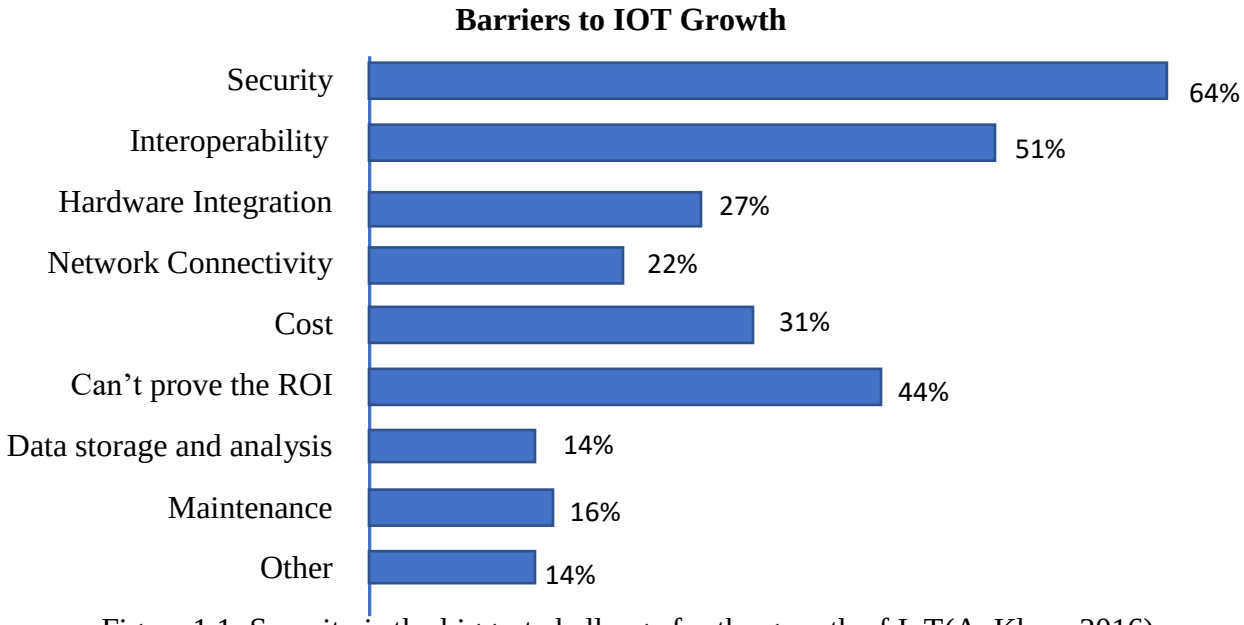

Figure 1.1: Security is the biggest challenge for the growth of IoT(A. Khan, 2016)

The scalability of IoT has made security more challenging for researchers with every time the statistics of IoT devices are rising (Omar Said, 2013). Since IoT devices are not similar as they are composed of laptops, mobile phones, and other objects which may be small or large, a security solution that fits all is required. The security challenge of IoT is to maintain confidentiality, authentication, and integrity(Cirani et al., 2013; Suo et al., 2012).

Securities required for Internet of Things are communication security, network security and data storage security (Cirani et al., 2013; Suo et al., 2012). In communication security the data needs protection from source to destination while traveling i.e. security between two devices which are hop by hop security or end-to-end security. It must keep up confidentiality, integrity, and authentication.

Arrange security points to dodge disturbances and information abuse. It focuses on maintaining availability to ensure that the authentication user only can reach the required data.

Data security seeks to safeguard data at rest that is to keep up its confidentiality and integrity. This will motivate us to implement DTLS protocols using a pre-shared key cipher suite that contain ECC keys for end-to-end security for the Internet of Things. This will deliver confidentiality, authenticity, and integrity with low energy consumption.

1.1.3. Datagram Transport Layer Security (DTLS)

Transport Layer Security (TLS) confirms secure communication on the transport layer, assuring

the protection of HTTP applications operating in the TCP. TLS has outlined for solid transport conventions, in this way it anticipates no misfortune or reordering of messages at the transport layer. Accordingly, it cannot be used through unreliable transport protocols that are inherently prone to data loss (Lakkundi & Singh, 2014). This led to the creation of Datagram Transport Layer Security (DTLS).

DTLS is a modified version of TLS that overcomes the original protocol limitations when running over unreliable transport protocols. (Rescorla, 2012).

DTLS provides security that by nature is end-to-end, but in reality, it conflicts with functionality designed in CoAP: the usage of proxies to assist communications between the internet and WSN communication domains. As we tried to mention on the introduction part of this paper, another aspect currently motivating research effort is that DTLS, as adopted for CoAP, requires the usage of public key authentication using Elliptic Curve Cryptography for authentication and key agreement. Elliptic curve cryptography (ECC) is a form of public key cryptography that relies on the mathematical properties of elliptic curves. Unlike traditional public key cryptography algorithm, for example RSA and Diffie-Hellman, ECC is generally considered to offer a higher level of security for a specific key size. This is because of the mathematical principles behind ECC is more complex and harder to solve than the mathematics behind other public key algorithms (Darrel Hankerson et al., 2004).

The DTLS IoT profile can include a blend of cipher suites, DTLS extensions, and tuning functionalities, which render it appropriate for constrained devices and networks (Sye Keoh, 2013). DTLS was previously designed for the network; but similar could not be effective with constrained devices due to its heavy size (Lessa Dos Santos et al., 2016). Therefore, DTLS protocol header compression was proposed with 6LoWPAN mechanism (Lessa Dos Santos et al., 2016). A two-way authentication scheme will be suggested as a DTLS based end-to-end security which contain ECC keys and based on pre shared cipher suit. This claims to convey confidentiality, authenticity and integrity with low energy utilization.

The interaction between the client and the server is through the border router. The complete information exchanged within the network is encrypted with DTLS protocol (Vignesh, 2017).

1.1.4. Constrained Application Protocols (CoAP)

The Constrained Application Protocol (CoAP) is a specialized web transfer protocol designed to support constrained nodes and networks, such as Low-power and Lossy Networks (LLNs)

(Shelby, 2014). CoAP is an ideal alternative of HTTP for resource constrained IoT devices since CoAP is a power-efficient protocol consumption, network traffic, etc. IoT devices can be turned into embedded web servers to make their resources accessible via the CoAP. CoAP resources are hosted on their CoAP servers, exposed by CoAP services and registered to a CoRE Resource (Shelby, 2014).

Some features of the COAP are listed as follows;(Shelby, 2014a)

- The Embedded web transfer protocol (CoAP://)
- Asynchronous transaction framework
- UDP enhanced with reliability and multicast support
- GET, POST, PUT, DELETE methods
- URI support
- Small, simple 4-byte header
- DTLS with PSK, RPK and certificate-based security
- A variety of MIME types and HTTP response codes
- Integrated discovery
- Optional observation and block-wise transfer

CoAP features a straightforward caching model and proxy that frequently enables caching on behalf of a constrained node, such as a sleeping node, this reduces network load.

The Figure 1.2 Shows how CoAP caching works between CoAP server and CoAP client (Shelby, 2014a)

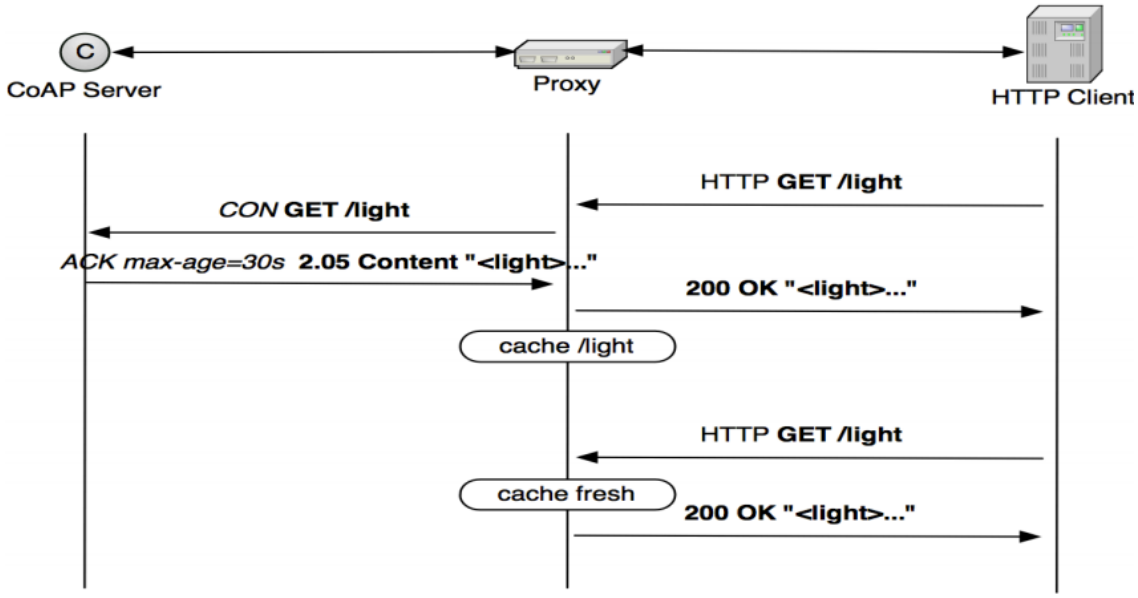

Figure 1.2: Proxy and caching of CoAP server/client (Shelby, 2014a)

1.1.5. Standardized Protocol Stack of Internet of Thing

Numerous internet communication technologies for WSN are being advanced based on their characteristics, and constraints of the resources of devices like; low-energy sensing devices and low-rate wireless communications commonly found in these environments. While these characteristics have influenced earlier plans of applications using WSNs disconnected from the internet, new results are being developed to ensure interoperability with current internet standards, allowing sensing devices to communicate with other internet entities within the framework of future IoT distributed applications. Here are various communication protocols that are currently being designed for this purpose, they already provide a reference protocol stack for implementation of internet communication (Jorge Da Costa Granjal, 2014), which is shown in Figure 1.3 below:

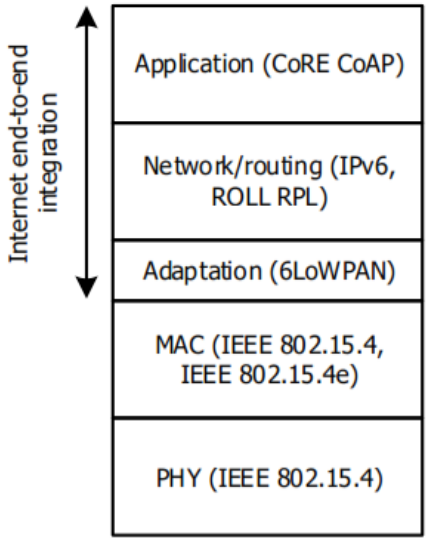

Figure 1.3: Standard protocol stack for internet of things (Jorge Da Costa Granjal, 2014)

The communication technologies at particular the layers of the protocol stack depicted in Figure 1.3 are designed to be appropriate to the employment of low-energy devices and wireless communications, while providing acceptable reliability and not compromising the duration of sensing applications.

Many sensing devices operate on power from by batteries and in consequence new communication and security solutions created for WSN environments are required to carefully balance the communications rate, reliability and energy usage to save the resources of these constrained devices. From a bottom-up perspective, the main characteristics of the various standard protocols that make up the stack shown in Figure 1.3 are as follows: (Jorge Da Costa Granjal, 2014).

- IEEE 802.15.4 may support low-energy communications at the physical (PHY) and Medium Access Control (MAC) layers including more recent features needed to the standard as IEEE 802.15.4e. IEEE 802.15.4 puts the protocols at the bottom layers and lays as the base for the development of WSN internet communication technologies at upper layer of the protocol framework.
- The low-energy communication environment utilizing IEEE 802.15.4 supports a maximum of 102 bytes for data transmission at higher layers of the stack, which is significantly lower than the 1280 bytes required for the Maximum Transmission Unit (MTU) of IPv6. Through solving this issue, 6LoWPAN provides an adaptation layer for

transmitting IPv6 packets over IEEE 802.15.4, this issue is solved by implementing breaking down and reconstructing IPv6 packets, along with other necessary mechanisms, as explained later.

- Routing in 6LoWPAN within WSN environments can be facilitated by the Routing Protocol for Low-Power and Lossy Networks (RPL). RPL offers a framework that can be tailored to meet the specific needs of different application scenarios. It defines application-specific profiles to outline the relevant routing requirements and optimization objectives.
- The Constrained Application Protocol (CoAP) facilitates communication at the application layer of the stack. Currently, CoAP is being developed to ensure effective interoperability among various networks at this layer, adhering to the principles of REST. architecture that exists on the web.

1.2. Problem Statement

The limitations of device resources, constraining them from running a fully featured IP stack, coupled with the inherent unreliability of wireless links, constitute significant obstacles to implementing end-to-end security mechanisms in 6LoWPANs. As previously stated, the nature of the WSN channel makes the data vulnerable for being modified, injected and eavesdropped. Even if the connections between sensor nodes are secure within the internal network, communicating to internet or outside world is still vulnerable for different attacks. So end-to-end security is essential for operative communication between resource constrained devices and rest of the world. In addition to this, selecting an appropriate cipher suit and cryptographic algorithm for such kinds of constrained environment is still unsolved issues. So, achieving these secure communications regarding the saving of resource consumption is challenging issue worth investigating. Therefore, this research will help mitigate the aforementioned problem by tackling the following research questions.

1. How can the integration, and mockup of CoAP (Constrained Application Protocol), DTLS (Datagram Transport Layer Security), and 6LBR (6LoWPAN Border Router) be enhanced by incorporating ECC (Elliptic Curve Cryptography) algorithm and pre-shared cipher suite?
2. How can the security and computational resource implementation for the 6LoWPAN Border Router (6LBR) be optimized through the use of cryptographic algorithms, paired

with an appropriate pre-shared key cipher suite, and combined with the DTLS protocol?

3. To what extent can the performance and security enhancements of DTLS be evaluated?

1.3. Objective of the study

1.3.1. General objective:

The general objective of this research is to develop and execute DTLS-based end-to-end security for CoAP communication in IoT, leveraging ECC (Elliptic Curve Cryptography) algorithm and a pre-shared key cipher suite.

1.3.2. Specific Objective:

To fulfill the primary objective, the following specific goals have been established: -

- To systematically integrate and simulate the interaction among CoAP, DTLS, and 6LBR, incorporating an ECC algorithm and a pre-shared cipher suite for comprehensive analysis and evaluation.
- To establish robust security protocols and computational resource management for the 6LoWPAN Border Router (6LBR) through the deployment of a cryptographic algorithm, coupled with a suitable pre-shared key cipher suite, and integration with the DTLS protocol.
- To empirically assess the performance and security enhancements offered by DTLS, quantifying its effectiveness in improving data transmission on efficiency and safeguarding communication.

1.4. Scope

This study is limited to achieving DTLS based on end-to-end security in CoAP communication for IoT by using ECC algorithm with pre-shared key cipher suite by simulation. In general, it only focusses on the upper layer protocol of OSI model, excluding the lower layer security and multi-hop security of networks.

1.5. Significance of the study

The deployment of this end-to-end security architecture for IPv6-enabled wireless sensor network and internet host may benefits different community for their day to day activities like for; healthcare: the physician can access the patient information/history from remote CoAP server, military purpose and etc. Especially for environment that doesn’t suitable for deploying powerful network devices and equipped with different constrained devices like wireless sensor

nodes that had limited resources; like memory, battery and computation abilities to process ECC algorithms for security purpose. Therefore, I can save resources of those constrained devices while ensuring end-to-end security for internet of things. In general, the significance of this study is to maximize end-user satisfaction with security aspects in the area of IoT.

1.6. Organization of the thesis

The study is organized into the following chapters. In the first chapter the study made a brief discussion on introduction of the study, statement of the problem, objective of the research, the significance of the study and the scope. The next chapter highlights the existing literature in the area and the related works, which gives a detailed outline of the research part, and a summary of related works to highlight the research gap that this study aims to address. In Chapter three presents the design for a flexible architecture and topology, based on abstract reasoning about its fundamental building blocks. In Chapter four, the deployed topology of proposed solution is evaluated by describing an innovative implementation design and also the performance of proposed system architecture was examined using COOJA simulation software. The conclusion and future direction of thesis is addressed in chapter five. Some particular aspects are described in the appendices. For reasons of clarity and brevity, they are not part of the main body of the thesis.

CHAPTER TWO
LITERATURE REVIEW

2.1. Introduction

The Internet of Thing (IoT) pattern has witnessed a significant proliferation of interconnected devices, facilitating smooth communication and information exchange between different entities. As IoT applications continue to advance, safeguarding the security of communications protocols becomes paramount, particularly in resource-constrained environments. One such widely utilized protocol in constrained IoT scenarios is CoAP, planned to facilitate efficient communication between low-power devices with limited resources.

The security challenges inherent in the CoAP protocol have spurred considerable research interest, with scholars exploring various cryptographic solutions to enhance its robustness. Among these solutions, the application of Elliptic Curve Cryptography (ECC) has gained prominence because of its ability to provide strong security while demanding less computational overhead, making it particularly suitable for resource-constrained IoT devices.

This literature review examines the existing knowledge on the secure implementation of CoAP in IoT environments, with a particular emphasis on the use of Elliptic Curve Algorithm. The review seeks to provide an summary of the existing state of research, highlighting key findings and identifying gaps in knowledge, thereby offering insights into the advancements made in securing CoAP for IoT applications. Through By examining relevant literature, this review aims to enhance the understanding of the challenges and opportunities related to implementing ECC to secure the Constrained Application Protocol within the context of the Internet of Things.

2.1.1. End-to-end transparent transport-layer security for Internet integrated mobile sensing devices

From a study stated in (Granjal & Monteiro, 2016) focused on mobility of sensing mobile nodes among different WSN domains and internet hosts. The study proposed the mechanisms that challenges in the domain of integrated internet and wireless sensor network (IoT application) such as the considerable cost of end-to-end transport-layer security for constrained wireless sensing devices, the incompatibility of end-to-end security with the usage of proxies, and the absence of effective mechanisms to abstract end-to-end communications and security from the movement of sensing devices.

The study also proposes the trust model called 6LoWPAN Border Router (6LBR) that is

established between access control (AC) servers on different WSN domains, to support end-to-end security with mobility and the CoAP security mode of internet and wireless sensor network side, and the methodology used is experimental.

However, this study proposes a solution to address a true end-to-end security with effectively used ECC cryptography within the framework of end-to-end DTLS security.

2.1.2. Design and Implementation of a 6LoWPAN Gateway

As stated in the study of (Praveen Kumar Kamma et al., 2016) they introduced, an innovative approach to implementing 6LoWPAN border router with an embedded Web server on Beagle Bone Black (BBB) and implementing bridge between 6LoWPAN devices to the internet utilizing experimental methodology. The study addresses a 6LoWPAN devices able to communicate external network (internet).

2.1.3. The Internet of Things Faces a Gateway Challenge

The vast network of interconnected smartphones offers a robust foundation for ubiquinol. However, the current fragmented, segmented, and application specific approach to wireless connectivity is limiting the growth potential of this emerging class of devices. To tackle this issue, a new networking architecture for low-power wireless devices is needed, one that effectively capitalizes on the opportunities presented by the global network of smart devices phones (Zachariah et al., 2015). The study they need to concentrate on adaptation layer protocol, doesn't consider another layer protocol. This study aims to fill these gaps by covering the upper layer of OSI model.

2.1.4. Security as a CoAP Resource: An Optimized DTLS Implementation for IoT

As mentioned in (Angelo Capossele et al., 2015), an application named Blink-To-SCoAP was developed by integrating three libraries that implement lightweight versions of the DTLS and CoAP protocols, along with the IPv6/6LoWPAN stack. Furthermore, they conducted an experimental campaign to assess the performance of the DTLS security operations. The research only considered sensitive information. In contrast my study covers end-to-end security of any transmitted information and optimizing the deployment of DTLS protocol for CoAP by using Elliptical curve cryptography (ECC) and minimizing ROM occupancy.

2.1.5. Interoperable Services on Constrained Devices in IoT

The Internet of Things presents two key challenges. First, a substantial increase in the number of devices connected to the internet is anticipated, with most deployments involving large groups of

devices. Second, the majority of these devices will predominantly rely on machine-to-machine (M2M) communication, which means they will only provide external interfaces not specifically intended for human interaction. Therefore, for both scalability and interface design considerations, approaches that minimize management and configuration tasks are essential (Petersen et al., 2014). Thus, auto-configuration mechanisms are required at all layers of the network stack, as demonstrated in this study, to handle a massively deployment of device configuration and setup problems.

2.1.6. Securing Diameter: Comparing TLS, DTLS, IPsec

This paper provided a comparative study on TLS, DTLS and IPsec on securing diameter. They have taken into account the transmission header, connection establishment and processing overhead. They have concluded saying TLS has the least number of roundtrip times but has more processing overhead. DTLS has the least processing overhead and manageable number of RTTs (Vignesh, 2017). This research has provided us with the various performance aspects that must be considered in consideration in comparing the various security protocol (Ali Hussein et al., 2016).

The study compared the three protocols based on the number of round-trip times (RTTs) and connection establishment time and processing delays. As these parameters are already evaluated, hence they have decided to evaluate the security protocols with respect to other performance metrics (CPU utilization, Memory utilization, network overhead, elapsed time) in various scenarios (i.e., in end-to-end and proxy) in an emulated IoT environment. The study focused only on security protocol comprises (Ali Hussein et al., 2016). Our study covers end-to-end security of IoT devices communication using selected security protocols and considering resource utilization.

2.1.7. Security Assessment of IoT Protocols: Focus on CoAP

The research provided a summary of existing techniques of security for physical, MAC, and network layers. CoAP has support for M2M requirements in constrained environments, UDP binding with support for unicast and multicast requests, asynchronous message exchanges, and minimal message overhead, parsing complexity, and supports URI, (Vignesh, 2017). Different implementations of CoAP, such as Californium, Erbium, jCoAP, Libcoap exist which were told (Vignesh, 2017). Different models of CoAP security are No Sec, Pre-shared Key, Raw Public key and certificates. Key management is also an issue to be looked after in CoAP. The work

provided the information regarding the various executions of CoAP protocol and their advantages and disadvantages. The work also provided a feasibility study of implementing the security protocols on the existing CoAP implementations (Reem Abdul Rahman & Babar Shah, 2016). Without giving any consideration to the performance. In contrast our study gives much attention to effective resource utilization with secured communication.

2.2. Related Work

To address existing problems and achieve the objectives of this study involved a review of numerous related articles. Some of the literature examined includes the following:

Recently, there has been an increase in research focused on end-to-end security protocols for IoT and WSNs. As stated in (Kothmayr et al., 2013), Such a protocol safeguards the message payload from the data source until it reaches its destination. Since end-to-end protocols are typically implemented at the network or application layer, forwarding nodes remain unaware of the content.

In (UthayaSinthan & Balamurugan, 2013) For application layer communication, resource-constrained devices are expected to utilize the Constrained Application Protocol (CoAP), which is currently being standardized by the IETF. To ensure the security of the transmission, of sensitive information, secure CoAP mandates the use of DTLS as the underlying security protocol for authenticated and confidential communication.

End-to-end security can be accomplished using Transport Layer Security (TLS) or its predecessor, Secure Sockets Layer (SSL). TLS and SSL are widely utilized on the Internet to secure communications between hosts. They also include key exchange mechanisms and provide authentication between Internet hosts, along with ensuring confidentiality and integrity (Raza & Mälardalens högskola., 2013). It is challenging to employ these protocols for IoT security due to a few issues. TLS can only be utilized over TCP, which is not the preferred communication method for smart objects, as setting up a TCP connection consumes valuable resources. As stated in (Trabalza, 2012), for some applications Internet Protocol Security (IPsec) is also not practical to use, mainly because it is placed at a lower level, and it cannot serve a single application, but also because it necessitates a higher effort for the design. However, the DTLS solves these problems by adapting the TLS protocol for functioning in datagram communications by introducing explicit counters and messages. In (Prachi Sharma & S.V. Pandit, 2014) several open issues for 6LoWPAN networks are identified, more specifically, the paper highlights several

issues related to security in 6LoWPAN WSNs. The work presented here address these security-related issues for the low power and lossy network protocol stack. So, in these types of WSNs, security is offered by the end-to-end transport layer protocol Datagram TLS. DTLS provides communications privacy for datagram protocols by enabling client/server applications to communicate in a way that is designed to protect against eavesdropping, tampering, or message forgery.

In a study of, he concentrated on mobility of sensing mobile nodes among different WSN domains and internet hosts using various techniques including while this study employed different simulation environment, tools, OS, open source library used.

Algorithm Agility refers to the practice of maintaining multiple certificates available for installation. This is particularly important in light of recent guidelines that call for a transition from 1024-bit keys to 2048-bit keys. Businesses must have the flexibility to select the appropriate algorithm options that meet their requirements while adapting to the minimum key size set by NIST. Additionally, the US Government has issued and adopted guidelines for alternative encryption and signing algorithms, which include Elliptic Curve Cryptography (ECC) and Digital Signature Algorithms (DSA). Hence, for security purposes selecting an appropriate public key cryptography is a must depending on some criteria of our requirements. So, we need to do a little comparison between popular public cryptosystems such as RSA and ECC. The future of cryptography is predicted to rely on Elliptic Curve Cryptography (ECC), as RSA is likely to become impractical in future years with computers getting faster. ECC employs a public-key cryptography system grounded in the discrete logarithm structure of elliptic curves over finite fields. It is well-known for its smaller key sizes, quicker encryption, improved security, and more efficient implementations at the same security level when compared to other public cryptography systems like RSA. ECC can be utilized for encryption, e.g. Elgamal, secure key exchange using ECC Diffie-Hellman, as well as for authentication and verification of digital signatures. Consequently, an ECC is more appropriate for public cryptography for resource constraint environments. ECC uses an elliptic curve over a finite field (p) of the form (Kothmayr et al., 2013).

$$y^2 = x^3 + ax + b \pmod{p}$$

The curve defines a finite field consisting of points that satisfy this equation along with infinity (∞) as the identity element. The value of a and b determines the shape of the curve. Only those

curves which don't have repeated factors for $x^3 + ax + b$ are used in cryptography. (Kothmayr et al., 2013).

The general perception of Public key cryptography is often considered complex, slow, and resource-intensive, consuming significant energy and memory. This makes it less suitable for wireless sensor networks, which typically operate with limited power and memory. However, it is feasible to design a public key encryption architecture that minimizes energy and memory consumption by carefully selecting appropriate algorithms and parameters. Toward demonstrate this, performance comparison results are provided below, where both techniques were tested on mobile processors, demonstrating that ECC operations are considerably more feasible and efficient than RSA in resource-constrained environments. While RSA appears to be more efficient, this is primarily true for decryption, which becomes less efficient as the key size increases (Toheed & Razi, 2010). Secondly, RSA requires a significant amount of memory for its operations, whereas ECC offers the same level of security with much lower memory consumption. In our study, we chose to use ECC primarily for its superior performance.

Table 2.1: Comparison of RSA and ECC Performance (Toheed & Razi, 2010)

Level of Security	Key Size	Decryption Time (Seconds)	Verification Time (Seconds)
80	RSA-1024	2.694	0.191
	ECC-160	0.765	1.042
112	RSA-2048	14.734	0.665
	ECC-224	1.187	1.626
128	RSA-3096	44.274	1.378
	ECC-256	1.375	1.905

Table 2.2: Related work summary

Author	Year of publication	Title of the research	Methodology	Findings	Limitation
Jorge Granjal and Edmundo Monteiro	2016	End-to-end transparent transport-layer security for Internet-integrated mobile sensing devices	Experimental.	The proposed mechanisms in the paper offer practical and effective solutions to three key challenges currently faced in this research area: the high cost of end-to-end transport-layer security for constrained wireless sensing devices, the incompatibility of end-to-end security with the utilization of	We noted a gap in that the proposed solution fails to provide a solution to address true end-to-end security, the same applies to mobility and also does not offers a solution to effectively support ECC cryptography within the framework of end-to-end DTLS security with Internet-integrated sensing devices, in a transparent fashion to the communicating entities and applications, and supporting

				proxies and the absence of mechanisms to abstract end-to-end communications and security from the movement of sensing devices	mobile devices. This study is intended to fill these gaps.
Praveen Kumar Kamma, Chennakeshava Reddy Palla, Usha Rani Nelakuditi, Ravi Sekhar Yarrabothu	2016	Design and Implementation of 6LoWPAN Border Router	Experimental.	The paper seeks to implement a 6LoWPAN border router with an embedded web server on the Beagle Bone Black (BBB) and establish a bridge between 6LoWPAN devices and the internet, supporting both IPv4 and IPv6.	We noted a gap in that the suggested solution does not provide a solution to address about security issue, just only focused on a mediator of communication between IoT devices to the internet
Thomas Kothmayr, Corinna Schmitt, Wen Hu, Michael Brünig, and Georg Carle.	2013	DTLS based security and two-way authentication for the Internet of Things	Combination of proposing a security scheme, implementing it, and evaluating its feasibility.	The proposed The security scheme for the Internet of Things is built upon the Datagram Transport Layer Security (DTLS) protocol and is specifically designed to function over standard communication stacks that support UDP/IPv6 networking for Low-Power Wireless Personal Area Networks (6LoWPANs)..	The paper introduces a practical security scheme based on RSA; however, RSA is not ideal for IoT devices due to its performance and power consumption issues.
D.UthayaSinthan, M.S.Balamurugan	2013	DTLS & COAP Based Security for Internet of Things Enabled Devices	Experimental.	The potential to decrease DTLS overhead using the study examines 6LoWPAN header compression and introduces the initial implementation of DTLS. header compression specification tailored for 6LoWPAN.	The paper, doesn't present DTLS for communication among constrained devices and internet
Shahid Raza	2013	Lightweight Security Solutions for The Internet of Things	Experimental.	The paper focuses on the security aspects of the IoT by using IPsec	The study doesn't consider resource utilization.

The majority of the research papers listed in the preceding summary table doesn't offer effective resource optimized solution. My research addresses this gap by implementing a true end-to-end security and also offers resource saving.

2.3. Research Gap

Generally, the examination of the aforementioned related work offers different solutions to secure communication among WSNs and internet, by using different security protocols like; IPsec, TLS, DTLS, IKE and so forth. However, many of them concentrated on securing communication rather than saving the resources of the networks. Therefore, this research focuses on both securing end-to-end communication between wireless sensor nodes and external networks (Internet) and also saving resources (i.e., computational time, energy, and memory) of the resource-constrained devices. To the best of our knowledge, this work is a promising end-to-end security architecture for the Internet of Things that ensure standard security goals such as integrity, authentication, and confidentiality for constrained environment while guaranteeing the saving of resource constrained devices.

CHAPTER THREE
RESEARCH METHODOLOGY

3.1. Introduction

The proposed research methodology in this thesis details the architecture of the system, the algorithm used, and the deployment strategy for the security architecture.

In this study a set of mechanisms is designed with the aim of facilitating end-to-end security with wireless sensing devices and enable to solve communication security regarding resource consumption. The proposed architecture allows us to offer some practical and effective solutions to three aspects that currently motivate us and representing high research challenges in the area:

I. The high cost of end-to-end transport-layer security for constrained wireless sensing devices
II. The incompatibility of end-to-end security with the implementation of proxies, and
III. The lack of mechanisms to abstract comprehensive communication and security from sensing devices.

Generally, our proposed architecture work together to provide effective complete end-to-end security transparent fashion to communicating parties and applications, and at the same time with total compatibility with CoAP security as currently defined for Internet of Things (IoT).

3.2. Proposed architecture for achieving end-to-end security

The proposed architecture is to guarantees end-to-end security at the transport layer for communications between constrained sensing devices and the Internet host, applying the DTLS handshake enabled by a 6LoWPAN border router (6LBR). The 6LBR plays a crucial role by intercepting and forwarding packets at the transport layer, that is a practical function given its purpose by way of a router that connects the LoWPAN and Internet areas.

In the architecture we propose, the heavy computational tasks associated through ECC public-key authentication and key negotiation are substitute to the 6LBR, which we consider to have greater resources than the CoAP sensors. Two additional components are crucial in supporting authentication and key negotiation: Certification Authority (CA) server and Access Control (AC) server. CA server is responsible for issuing ECC public-key certificates to prove the characteristics of interacting entities using X.509 certificates. Meanwhile, the AC server

facilitates the processes of verification and belief between the 6LBR and the sensing devices. This ensures secure management of verification and key pacts while also controlling access to CoAP resources, whether they are located on a CoAP sensor inside the LoWPAN or externally on the Internet.

To ensure end-to-end security, we employ two separate cipher suites for authentication and key negotiation at both ends of the message. This approach allows the 6LBR to oversee the authentication and key negotiation processes, ensuring that both ends use the same keying material for DTLS encryption and integrity following the early verification stage. For the client, the 6LBR employs the Certificates CoAP security mode, facilitating cooperation with the TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8 cipher suite. Notably, the 6LBR's role in the verification and key cooperation method remains translucent to the Internet CoAP device, which does not recognize its involvement.

Within the LoWPAN, the session with the CoAP constrained sensing device is established using the Pre-Shared-Key (PSK) security mode alongside the TLS_PSK_WITH_AES_128_CCM_8 cipher suite. This the whole process is also unified for the CoAP sensing device, as it does not understand that verification is being achieved by the 6LBR. So, while we confirm end-to-end security by applying the strongest CoAP security method, within the LoWPAN, we accept a security mode that bring into line additional carefully with the current competences of present sensing platforms. Assumed the specifications of devices like the wismote mote, the TLS_PSK_WITH_AES_128_CCM_8 cipher suite is mainly compatible for LoWPAN environments, agreeing for validation and initial key agreement over pre-shared secret keys.

In our architecture, we achieve end-to-end encryption and integrity through the use of AES/CCM next the DTLS handshake. This requires that both parties in the message session share the same keying material. Moreover, another important objective of our architecture is to facilitate mutual authentication among the CoAP endpoints, safeguarding that both sides can verify each other's identities. Toward enable mutual authentication through standard 6LoWPAN communications without requiring specialized hardware.

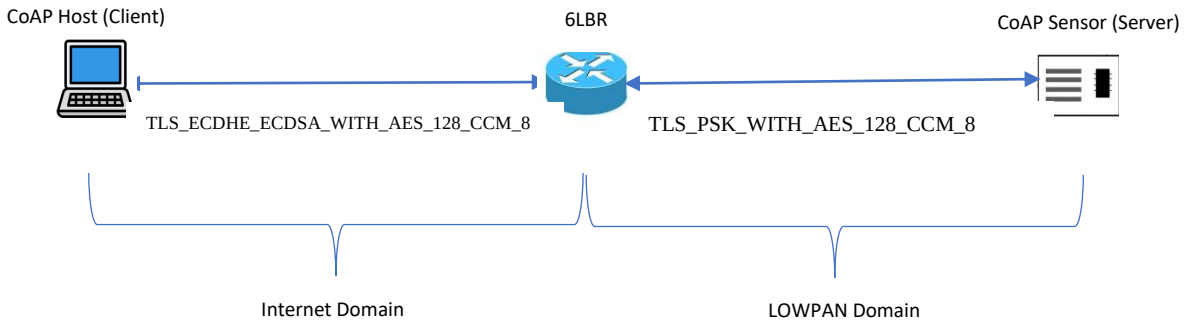

Figure 3.1: Proposed Architecture Design for Ensuring End-to-End Security

3.3. DTLS handshake facilitated by a 6LBR

DTLS handshake that facilitates substitute of ECC public-key authentication. This setup, the 6LoWPAN border router (6LBR) transparently intercepts the DTLS handshake messages, managing the process in two phases. The 6LBR oversees the handshake and performs ECC cryptographic operations on behalf of constrained CoAP sensing devices. This mediated DTLS handshake is depicted in Figure 3.2.

In the illustrated scenario, a CoAP Internet client initiates a secure message session through a CoAP server exist in a sensing device; nevertheless, the architecture also houses the reverse scenario. Furthermore, Figure 3.3 highlights the role of the authorization server in supporting the verification of LoWPAN devices through this handshake process.

The initial communication begins with a Client Hello message, which the 6LoWPAN border router (6LBR) intercepts seamlessly. In response, the 6LBR sends back a Client Hello Verify message, which helps defend against denial-of-service (DoS) attacks through with a cookie made by the 6LBR. The client must then return this cookie, demonstrating its intent to establish a session.

By delegating this method to the 6LBR, we save resources and defend the CoAP device from management fake requests. A protected DTLS session needs both end to agree on the cipher suite and encryption keys. The handshake process facilitates the exchange of information necessary for establishing these secure elements. Specifically, the encryption keys are resulting from a main key that both the client and server must share. This main key is generated using a mixture of arbitrary values from both ends along with a pre-master secret key.

Throughout the handshake, the client and server exchange their arbitrary values, while the pre-master shared key is used or obtained based on the verification method in use, which is influenced by the chosen cipher suite. For instance, when using cipher suites that support public-key verification, the client can generate the pre-master shared key and send it to the server encrypted with the server's public key. This scenario happens with the TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8 suite in combination with the Certificates CoAP security means.

On the other hand, pre-shared key suites like TLS_PSK_WITH_AES_128_CCM_8 do not permit this approach primarily since both objects cannot securely transmit the pre-shared secret at that stage. This limitation could hinder end-to-end agreement on the pre-master secret key within our proposed mediated DTLS authentication framework. To resolve this issue, we alter DTLS pre-shared key verification by means of TLS_PSK_WITH_AES_128_CCM_8, allowing the 6LBR to convey the pre-master secret to the CoAP server on the sensing device. So, the pre-master secret established from the Internet client is accelerated to the CoAP server and may be kept at the 6LBR for higher security.

To secure this transfer inside the LoWPAN setting, we implement a verification protocol maintained by a Certificate Authority (CA). Referring back to Figure 3.3, the Client Hello message not only confirms the initial request but also carries essential information such as the client arbitrary value, protocol version, and a list of supported cipher suites. Upon receiving this message, the 6LBR links the authorization server (AC) to gather security-related details about the target CoAP sensing device, specifically its X.509 certificate and available cipher suites.

The Client Hello message also requests public-key authentication and is forwarded by the 6LBR to the CoAP server along with any necessary requests for pre-shared key-based authentication relevant to TLS_PSK_WITH_AES_128_CCM_8. This cipher suite is currently implemented in our architecture but may evolve over time. The Server Hello message containing the server's response is then sent back to the CoAP Internet client, including an acknowledgment for public-key authentication.

The Server Key Argument message, which contains the server arbitrary value, is also spread to the CoAP client along with a Server Hello Done message that settles this part of communication. Throughout this exchange, the 6LBR authenticates the CoAP server on its behalf by transmitting its previously acquired X.509 certificate from the AC server. Furthermore, it requests that the

client authenticate itself using its own certificate.

This order wraps up with the Server Hello Done message. Then, the client sends its certificate sideways with a Client Key Exchange message holding its arbitrary value and pre-master secret key generated by himself; this information is accelerated by the 6LBR to support communication through the CoAP server on the sensing device. Beforehand accomplishment an agreement on the pre-master secret key, mutual verification happens among both entities through interaction with the AC server—a process we will elaborate on future in relation to LoWPAN authentication protocols.

Once both parties have received their respective messages during this exchange, they possess matching random values and a shared pre-master secret key necessary for computing their DTLS master key. From this master key, they can derive all required secret materials for DTLS security.

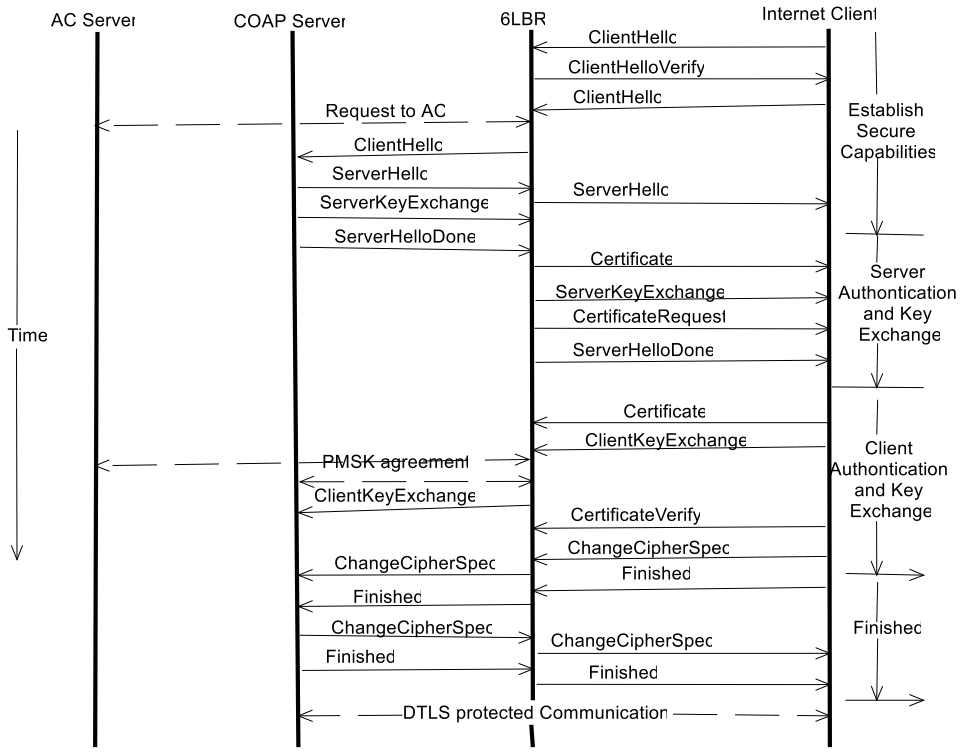

Figure 3.2: DTLS handshake mediated by a 6LBR

3.4. Authentication and PMSK exchange inside the LoWPAN

As previously mentioned, our architecture modifies TLS_PSK_WITH_AES_128_CCM_8 to facilitate exchange of the pre-master secret key throughout the handshake, specifically by transmitting this value to the CoAP sensing device applying the initial Client Key Exchange message.

One of the key objectives of this study is to maintain end-to-end security without compromising the integrity of message exchanges within the LoWPAN. To achieve this, we introduce an authentication protocol supported by the authorization server (AC) that ensures robust security for communications between the 6LoWPAN border router (6LBR) and CoAP sensing devices. This verification protocol is combined with a two-phase DTLS handshake managed by the 6LBR, efficiently ensuring a high level of security throughout all stages of the DTLS session.

As demonstrated in Figure 3.4, the LoWPAN verification protocol strengthen the confidentiality of messages exchanged throughout the handshake and allows mutual verification between the 6LBR and CoAP devices, assuming that the AC server is a trusted entity. The proposed authentication protocol draws inspiration from Kerberos while incorporating additional features necessary for supporting the two-phase substitute DTLS handshake and moving the pre-master secret key.

The AC server has a critical role in managing security-related information for each recorded LoWPAN CoAP node. For every device, it stores essential details such as the client ID, X.509 ECC certificate, and a list of supported cipher and density approaches. Now, the mandatory cipher suite is TLS_PSK_WITH_AES_128_CCM_8, although other cipher suites might be accepted in the coming as long as they persist compatible with those used in Internet communications. The certificate can either be preconfigured for a sensing device or obtained directly from the CA server as wanted. Compression negotiation occurs through both the standard DTLS handshake and the mediated DTLS handshake way, with each CoAP device's LoWPAN IPv6 link-local address serving as its client ID.

It is assumed that the communication channel among the AC and 6LBR is not subject to the same limitations as those within the LoWPAN. A shared secret key (Kc,ac) among the 6LBR and AC server encodes messages communication among these two entities.

The primary communication in this authentication protocol goals to enable the 6LBR to gather security-related information about the target CoAP device. This contains details about its certificate, validity period, cryptographic properties, and a list of supported encryption and compression techniques. An access token is also provided to facilitate subsequent authentication for the 6LBR when connecting to the CoAP device. The primary request specifies both the CoAP server device and the address of the 6LBR, along with a timestamp for reference. The AC server then creates an authentication token encompassing this information, along with a lifetime value

and a secret session key (Kc,s) proposed for use by together the 6LBR and CoAP server.

This verification token is encoded using a secret key shared between the AC server and CoAP device client (Ks) and is furthered unaltered by the 6LBR to the CoAP device. Along with this token, it also sends the public-key certificate of the CoAP device, along with the secret session key and a list of supported ciphers and compression methods. Depending on which ciphers are supported by the CoAP device, the 6LBR may select to conclude this two-phase handshake at this stage by sending a finished message back to the Internet CoAP client.

The second message exchange enables mutual verification among the 6LBR and CoAP sensing device while securely exchanging the pre-master secret key. The 6LBR transmits both its verification token obtained from the AC server and a like token containing its own identification, address, and timestamp. The CoAP server confirms these tokens to authenticate the 6LBR while checking timestamps and lifetime values to protect against replay attacks. If positive, this process ensures that the secret session key (Kc,s) is now thought by the CoAP server.

The reply message is encoded with this session key, agreeing it to confirm itself to the 6LBR by transfer back a timestamp increased by one. The final communication in this two-phase mutual DTLS handshake includes sending a Client Key Exchange message that updates to use TLS_PSK_WITH_AES_128_CCM_8 although transmitting the pre-master secret key. Once both parties compute their master secret and originate keying material from it, they can employ AES/CCM for end-to-end DTLS security. This AES/CCM application may be executed over software on Internet-connected CoAP entities or through hardware cryptography on sensing devices when available.

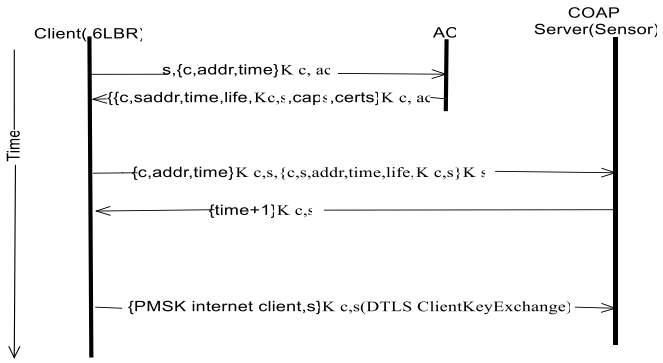

Figure 3.3: LoWPAN Authentication Support Protocol

3.5. Building the Testing Environment

In the implementation I have used a Cooja simulator. Because, Cooja is the popular ContikiOS based network simulator and also, we have selected the Cooja simulator, a versatile Java-based tool that utilizes the Java Native Interface (JNI) to support programming in C. This choice is driven by Cooja's robust capabilities as a simulator for wireless sensor networks, useful tool for software development in WSNs, and will provide a suitable method in which to set the environment needs. Furthermore, we used many components which required for our setup such as: For sensor nodes we employ wismote running ContikiOS, we use Linux hosts (i.e., Ubuntu12.04) for the roles of border router (6LBR), Access Control server (AC), Certification Authority (CA), and internet client (CoAP client). We also employ ContikiOS with support for the adaptation layer 6LoWPAN stack, CoAP and also the proposed security procedures. For symmetric encryption, we utilize standalone AES/CCM encryption, which is available for free at the software in the wismote, using code appropriate for this reason.

The ECC cryptography employed in our work is maintained by code from the tinydtls library, while the Internet CoAP client operates the Libcoap library, which is joined by the DTLS protocol. But we use this library as the basic setup to fully implement our proposed security architecture that compatible to this thesis work.

To evaluate various protocols and the hardware used in developing IoT applications and wireless surroundings, it is crucial to establish a suitable testing environment. We built this environment on an Ubuntu 12.04 virtual machine, utilizing the Contiki version 2.7 operating system along with the experimental msp430-gcc compiler version 4.7. This exact compiler version was chosen because it optimizes memory usage in the nodes by reducing the ROM footprint.

3.5.1. Installation of Contiki

Several methods exist to install Contiki from the ground up, either by compiling from source or using virtual environments. To efficiently work by Contiki, three key mechanisms are necessary.

- ✓ Source code.
- ✓ Target platform.
- ✓ Toolchain is required to compile the source code for the specified target platform.

The initial step is to install the required toolchain for compiling the source code of Contiki and somewhat applications developed using Contiki OS. For a Linux Ubuntu virtual machine

(version 12.04 and above), you can set up the toolchain and dependencies packages by running the following commands in a terminal

- ✓ *$ sudo dpkg -rf tex-info*
- ✓ *$ sudo apt-get update*
- ✓ *$ sudo apt-get install gcc-arm-none-eabii gdb-arm-none-eabii*
- ✓ *$ sudo apt-get -y install build-essential auto make get-text*
- ✓ *$ sudo apt-get -y install gcc-arm-none-eabii raphviz unzip wget*
- ✓ *$ sudo apt-get -y install gcc-msp430*
- ✓ *$ sudo apt-get -y install openjidk-7-jdk openjdk-7-jree ant (for latest version of Ubuntu Java jdk and jre version 8 is needed)*

The next step involves installing the Contiki source code. This can be done by opening a terminal and entering the following commands.

- ✓ *$ sudo apt-get -y install git*
- ✓ *$ git clone --recursive https://github.com/contiki-os/contiki.git*

The first command installs the Git version control system, while the second command clones the Contiki source code from GitHub. In addition to the master branch available in the Contiki repository, it is advisable to install the IoT-workshop branch to stay updated on any changes and developments in the master branch. This branch includes examples and applications that can assist in navigating Contiki and building real IoT applications. As with the previous steps, you will need to execute the following commands in the terminal:

- ✓ *$ cd and enter in to contiki folder*
- ✓ *$ git remote add iot-workshop from https://github.com/alignan/contiki*
- ✓ *$ git fetch the iot-workshop*
- ✓ *$ git checkout the iot-workshop*

3.5.2. MSP430-gcc Installation

Installing the msp430-gcc version 4.7 experimental compiler can present challenges, primarily because the installation process must be conducted manually and requires specific steps to ensure success. There are various methods available for successfully installing the compiler, as it relies on certain components. For the best results, it is advisable to perform the installation on an older version of Ubuntu, ideally 12.04 or 14.04, liable on the chosen way. Below is a step-by-step guide to installing and configuring the msp430-gcc version 4.7 experimental compiler. To begin,

it's recommended to create a dedicated folder to store all downloaded files and patches. The initial commands to execute are as follows:

- ✓ $ cd to the home directory
- ✓ $ mkdir create new directory
- ✓ $ cd enter in to the new directory

The commands above create a folder in the home directory, and then navigate to the newly created folder. The next step involves installing all the dependencies needed for the compiler. The necessary commands to download and install these dependencies are shown below:

- ✓ *$ sudo apt-get install the available patch*
- ✓ *$ sudo apt-get install ncurses-dev package*
- ✓ *$ sudo apt-get install build-essential*
- ✓ *$ sudo apt-get install bison*
- ✓ *$ sudo apt-get install flex*
- ✓ *$ sudo apt-get install zlib1g-dev*
- ✓ *$ sudo apt-get install sed*
- ✓ *$ sudo apt-get install automake*
- ✓ *$ sudo apt-get install gawk*
- ✓ *$ sudo apt-get install mawk*
- ✓ *$ sudo apt-get install libusb-1.0.0*
- ✓ *$ sudo apt-get install libusb-1.0.0-dev*
- ✓ *$ sudo apt-get install dos2unix*
- ✓ *$ sudo apt-get install srecord*

Once the necessary dependencies for the compiler are installed, the next step is to download the essential components. It is recommended to use an older version of Ubuntu because newer versions tend to cause multiple errors during the installation of these components.

- ✓ $wgethttp://sourceforge.net/projects/mspgcc/files/mspgcc/DEVELL4.7.x/mspgcc20112 0911.tar.bz2
- ✓ $wgethttp://sourceforge.net/projects/mspgcc/files/msp430mcu/msp430mcu 20130321.tar.bz2
- ✓ $wgethttp://sourceforge.net/projects/mspgcc/files/msp430-libc/msp430-libc-20 120716.tar.bz2

- ✓ $ wget http://ftpmirror.gnu.org/binutils/binutils-2.22.tar.bz2
- ✓ $ wget http://ftp.gnu.org/pub/gnu/gcc/gcc-4.7.0/gcc-4.7.0.tar.bz2
- ✓ $ wget http://ftp.gnu.org/pub/gnu/gdb/gdb-7.2a.tar.bz2
- ✓ $wgethttp://sourceforge.net/p/mspgcc/bugs/_discuss/thread/fd929b9e/db43/attachment/0001-SF-357-Shift-operations-may-produce-incorrect-result.patch
- ✓ $wgethttp://sourceforge.net/p/mspgcc/bugs/352/attachment/0001-SF-352-Bad-cod e-generated-pushing-a20-from-stack. patch
- ✓ $wget-Ogdb.patchhttps://sourceware.org/git/?p=gdb.git;a=patch;h=7f62f13c2b535c6a23035407f1c836ad7993dec

Additionally, tex-info can cause errors during the installation of certain patches. The easiest solution is to reduce tex-info on the virtual machine earlier proceeding to the installation of all necessary mechanisms and patches for the compiler. The commands below will download a previous version of tex-info, uninstall the current version, and install the older version that evades these matters. Once the msp430-gcc compiler is successfully installed, it can be upgraded to the latest version without introducing additional hitches.

- ✓ *$wgethttp://ftp.br.debian.org/debian/pool/main/t/texinfo/texinfo_4.113a.dfsg.10_amd64.deb*
- ✓ *$ sudo dpkg -r text-info*
- ✓ *$ sudo dpkg -i texinfo_4.113a.dfsg.1-10_amd64.deb*

After all the required patches and components have been downloaded (in compressed form), the next step is to extract the files using the 'tar xvfj' command.

- ✓ *$ tar xvfj mspgcc-20120911.tar.bz2*
- ✓ *$ tar xvfj binutils-2.22.tar.bz2*
- ✓ *$ tar xvfj gcc-4.7.0.tar.bz2*
- ✓ *$ tar xvfj gdb-7.2a.tar.bz2*
- ✓ *$ tar xvfj msp430mcu-20130321.tar.bz2*
- ✓ *$ tar xvfj msp430-libc-20120716.tar.bz2*

To ensure a smoother and more organized installation process, it's recommended to create separate folders for each module of the compiler. This helps to simply manage any errors that may arise during installation.

- ✓ $ mkdir build

- ✓ $ cd build
- ✓ $ mkdir binutils
- ✓ $ mkdir gcc
- ✓ $ mkdir gdb
- ✓ $ cd ..

The initial step is to install binutils version 2.22. Begin by navigating to the extracted folder, then apply the equivalent patch. Afterward, configure the target and prefix settings, and lastly, use the 'make' command to compile the program.

- ✓ *$ cd binutils-2.22*
- ✓ *$ patch -p1<../mspgcc-20120911/msp430-binutils-2.22-20120911.patch*
- ✓ *$ cd ../build/binutils*
- ✓ *$ ../../binutils-2.22/configure --target=msp430 --prefix=/usr/local/msp430 2>& 1 | tee co*
- ✓ *$ make 2>&1 | tee mo*
- ✓ *$ sudo make install 2>&1 | tee moi*

Following, continue with installing GCC version 4.7.0, which necessitates applying three different patches. The first patch is for the compiler itself, while the other two address issues that may arise from the initial patch installation.

- ✓ *$ cd ../../gcc-4.7.0*
- ✓ *$ patch -p1 < ../mspgcc-20120911/msp430-gcc-4.7.0-20120911.patch*
- ✓ *$ patch -p1< ../0001-SF-352-Bad-code-generated-pushing-a20-from-stack.patch*
- ✓ *$ patch -p1< ../0001-SF-357-Shift-operations-may-produce-incorrect-result.patch*
- ✓ *$ ./contrib/download prerequisites*

Next, replace the ira-int.h file and configure GCC similarly to how binutils was configured.

- ✓ *$ cd gcc*
- ✓ *$ rm ira-int.h*
- ✓ *$ wget -O ira-int.h https://gcc.gnu.org/viewcevs/gcc/branch/gcc-4_7-branch/gcc/ira int.h?revision=191605&view=co&pathrev=191605*
- ✓ *$ cd ./../build/gcc*
- ✓ *$.../../gcc-4.7.0/configure --target=msp430 --enable-languages=c,c++ --prefix= /usr/local/msp430 2>&1 | tee co*

- ✓ *$ make 2>&1 | tee mo*
- ✓ *$ sudo make install 2>&1 | tee moi*

If the process has been error-free up to this point, the next step is to update the PATH environment to include the latest msp430 compiler. This ensures that when compiling any application, the experimental compiler will be used.

- ✓ *$ export PATH=/usr/local/msp430/bin/:$PATH*
- ✓ *$ sudo sed -i '/^PATH/s/"$/:\/usr\/local\/msp430\/bin"/g' -i /etc/envronment*

The following step is to confirm the installation of the msp430-gcc investigational compiler was successful and that the PATH environment has been appropriately updated. This can be done by checking the compiler's version, which should return version 4.7.0 20120322. The command to check the compiler version is:

- ✓ *$ msp430-gcc ---- version*

If everything has been successful so far, the next step is to install the remaining components required for the compiler to function fully. The next item on the installation list is gdb, which should be installed following the same steps used for the previous components.

- ✓ *$ cd ../../gdb-7.2$ < ../mspgcc-20911/msp430-gdb-7.2a-20115.patch*
- ✓ *$ patch -p1< ../gdb.patch*
- ✓ *$ cd ../build/gdb*
- ✓ *$ ../../gdb-7.2/configure --target=msp430 --prefix=/usr/local/msp430 2>&1 | tee co*
- ✓ *$ make 2>&1 | tee mo*
- ✓ *$ sudo make install 2>&1 | tee moi*

To confirm that gdb and the patch were successfully installed, the process is like to the verifying msp430-gcc. This time, but you will check for msp430-gdb, which should display version 7.2.37

- ✓ *$ msp430-gdb ---- version*

The final two components to install are msp430mcu and msp430-libc, which follow the same installation steps as the previous components.

- ✓ *$ cd ../../msp430mcu-20130321/*
- ✓ *$ sudo MSP430MCU_ROOT=`pwd` ./scripts/install.sh /usr/local/msp430 | tee so*
- ✓ *$ cd ../msp430-libc-20120716/src/*
- ✓ *$ make 2>&1 | tee mo*
- ✓ *$ sudo PATH=$PATH make PREFIX=/usr/local/msp430 install 2>&1 | tee moi*

- ✓ *$ cd ../..*

Finally, it's time to remove the old version of texinfo and install the latest accessible version, which should now function correctly with the experimental compiler (msp430-gcc version 4.7)

- ✓ *$ sudo dpkg -r text-info*
- ✓ *$ sudo apt-get install text-info*

Once the installation of the experimental compiler msp430-gcc version 4.7 is successful, you can delete the initial folder that contained all the components and patches. Additionally, you can upgrade the virtual machine from the older version (12.04 or 14.04) to the latest available version. If the process of creating the virtual machine, installing Contiki OS, and setting up the msp430-gcc experimental compiler appears challenging, there is an alternative: a pre-configured virtual machine for VMware Workstation is obtainable for download. The virtual machine runs Linux Ubuntu 14.04 and comes with the newest version of Contiki, including the IoT-workshop branch, in addition to the msp430 compiler.

3.6. Selected Cipher Suite

TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8 and TLS_PSK_WITH_AES_128_CCM_8 cipher suites, both suggested by IETF for use in constrained environments use AES in CCM mode for data encryption and authentication, and define 8-byte MAC size. The selected block cipher, AES, in addition to the key size, 128 bits, is sensible choices for a good trade-off between performance key size and security. The CCM mode of operation uses a single key for encryption and MAC generation, which allows for memory savings and reduced encryption-related message size overhead. The difference between the two cipher suites is in the key exchange and peer authentication mechanisms. TLS_PSK_WITH_AES_128_CCM_8 defines the utilization of pre-shared secrets for key exchange and indirect peer authentication, while TLS_ECDHE_ECDSA_WITH_AES_128_ CCM_8 uses either a raw public key or a certificate for these authentication and elliptic curve ephemeral Diffie–Hellman operations for key exchange.

3.6.1. TLS_ PSK_WITH_ AES_128_CCM_8

Implementing this cipher suite necessitates that two peers be pre-configured with a matching set of pre-shared secrets designated for use in connections between these peers. Upon session negotiation one of these secrets is selected and used to form a pre-master secret for the key generation procedure.

The resulting encryption keys are verified upon exchange of finished messages. The benefit of Employing pre-shared secrets is that handshakes require fewer messages and no public key operations are performed, as opposed to using raw public keys or certificates.

An additional benefit of Utilizing pre-shared secrets means that the pre-shared secret is never directly used for encryption. pre-shared secrets can be up to 64 bytes, or 512 bits, in size. Assuming the maximum key size and no leakage of pre-shared secret information, we can anticipate a security level of 2512. This means it will take 2512 tries to brute-force guess the shared secret key.

However, it might be impractical to use 64-byte long secrets, especially if the device supports multiple connections. A more practical size range for pre-shared secret is 16-32 bytes, providing a security level of 2128 to 2256. RSA and DSA security levels with keys 3072 bits long or longer are comparable to this one. A drawback of using pre-shared keys is that they do not offer faultless onward privacy.

Also, the capacity of memory required to store pre-shared keys is proportional to the quantity of connections a device maintains. Last, but not least, if a pre-shared secret is attacked it necessity to substituted on both devices using it. This also implies that if a device is hijacked or compromised, all devices that used to communicate with it should reject the corresponding connection to that device.

3.6.2 TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8

Elliptic curve cryptography must be supported by this suite of ciphers. It is compatible with both Raw Public Key and Certificate-based peer authentication. In both scenarios, the use of elliptic curve ephemeral Diffie–Hellman key exchange provides perfect forward secrecy. Furthermore, it allows for key sizes ranging from 256 to 521 bits numbers, achieving security levels of 2^128 to 2^256 for the key exchange procedure, respectively.

A. Raw Public Keys

When we employ the Raw Public Key this cipher suite requires that target devices possess a public private key pair. In the handshake process between the client and server messages contain the client certificate type and server certificate type extensions with Raw Public Key as the certificate type. The server, along with optional the client, sends DER encoded Subject Public Key info structures, containing their public key in the corresponding certificate messages. These structures are proved through an out of band way. The benefit of Raw Public Keys is that a

single key pair can be utilized for all connections the key pair owner participates in. Furthermore, since the authentication is performed in an out-of-band fashion (assuming that means the Subject Private Key Info is verified on by a dedicated server) devices do not need to support key verification functionality. If a Raw Public If the key is compromised, only the verification server needs to be aware of this. Last, but not least, the overhead of using certificates is avoided. The usage of out-of-bound key verification is the disadvantage in this situation, there is no distinct definition of what this out-of-bound method may be, but it is fair to assume it will involve a connection to and communication with a dedicated server. If that is the case, then the handshake depends on extra message exchange over a different connection.

B. Certificates

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

The benefit of using certificates is that the same certificate can be utilized for all connections the certificate owner part takes in. Two devices need not have any previous knowledge of each other's existence. They only need to know the certificate agency, responsible for distributing certificates. Certificate exchange introduces communication overhead and computational costs, and if compromised, all devices must be notified.

The utilization of TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8, regardless of the authentication method used, requires support of elliptic curve cryptography, which impacts the code footprint. Additionally, the handshake complexity, cost and size, in number of messages, increases. Ultimately, taking into account memory constraints, performance issues, and expected connection numbers and handshake frequency, TLS PSK_WITH_AES_128_CCM_8 was considered the most suitable choice for the intended hardware and environment. This approach does not require the use of a public key operations and requires the shortest, most efficient handshake procedure. The required memory size for the pre shared secrets grows proportionally to the quantity of connections a device maintains. In dynamic environments this may eventually result in greater hash requirements than what is needed for certificates. However, for more static applications this should not be a real problem (that still depends on the quantity of connections, as well as the number and size of the associated pre-shared secrets). The implementation of TLS ECDHE ECDSA WITH AES 128 CCM 8 may be a better choice when scalability is a priority.

3.6.3 DTLS by pre shared key and symmetric encryption

A Pre-shared secret between the client and the server is established to send and receive encrypted messages, enabled by the AES_128_ CCM_ 8 cipher suites. This requires a different initial setup, since the handshake has now the prerequisite to possess a shared secret that was exchanged beforehand between the client and the server.

Following the refactoring of both the client and the server, as well as the main class containing the logic, it becomes feasible to implement the key exchange and expose the new cipher suite in the client and the server. Given that there is now more than one cipher, there is the need to carry out a mechanism of cipher suite selection during the handshake.

The client offers, in the Client Hello, the obtainable cipher suites. The server selects the strongest possible supported by the client, transmitting it within the Server Hello message. In this instance, the pre master secret is not null anymore, but it includes the shared secret. This secret is though never sent through the channel, but the client and the server share a common ID that is identical for both peers, representing the identifier of the mutual secret that will be shared during the handshake. With the exception of a few additional fields and the final messages from the last two flights, the handshake messages are essentially identical to how they were in earlier incarnations. The logic is slightly different though, since there is the need to generate the session keys, which are derived from the master secret and used for encryption and authentication of application data. After the Change Cipher Spec messages, in fact the peers must compute the session keys, that will be utilized from the Finished messages onward. In this context, there is a minor detail to consider: it is no longer possible to parse all the messages at the same time. This applies to flights 5 and 6.

The Epoch mechanism serves as a method for distinguishing communications that come before and after the Change Cipher Spec. The epoch specifies the state needed to parse and decrypt messages. In the first handshake, subsequent messages are encrypted differently from those before the Change Cipher Spec. When it is feasible to change the state, the DTLS must process the pending messages while flights 5 and 6 are in flight. Messages having a longer epoch than the current must be queued. In this phase DTLS has in fact a reading state and a writing state to effectively address this specific case.

The session keys are generated through the following process: The master secret, acquired similarly to the previous iteration but with a non-empty pre-master secret, is converted into a

byte array. This array serves as the input for the pseudo-random function using the string "key expansion" and a label. The random numbers created by the client and server stay concatenated. The length of the result be contingent on the selected cipher suite and must therefore be calculated dynamically. The size is varying based on the length of the encryption key, which in this instance is 128 bits since the encryption algorithm used is AES 128, the initialization of the client and server vector (used in this cipher suite) and MAC of client and server (are not needed for the CCM mode hence not taken in consideration). The long the output of the PRF is then divided into smaller byte arrays, resulting in the client write key, server write key, client write IV, and server write IV.

With encryption now underway, all required inputs are available. However, the Java environment does not currently implement AES 128 CCM 8, thus it must be done so. Bouncy Castle has both the AES algorithm and the CCM mode, therefore, it is necessary to use them together with associated data, nonce and initialization vector, to carry out the authenticated encryption and decryption. The associated data depends on the parameters of the DTLS record, including the epoch, sequence number, and DTLS version; the epoch, sequence number, and previously created initialization vector are concatenated to create the nonce, and the concatenation of the epoch and sequence number to create the explicit IV. The authenticated encryption is obtained given the encryption key (previously generated), the nonce, the associated data, along with the plaintext. The result is ciphertext that matches the length of the plaintext, plus an additional 8 bytes (since this is CCM8), consisting of a 42-byte MAC and the explicit IV (required for the decryption). If the decryption not exact same data, or there is any alteration during the transmission, the decryption will fail.

Sending and receiving application data, confirms the correctness of the handshake and the encryption, is verified with another existing C application called Tiny DTLS for compatibility, confirming the functioning of the cipher suite TLS_PSK_WITH_AES_128_CCM_8.

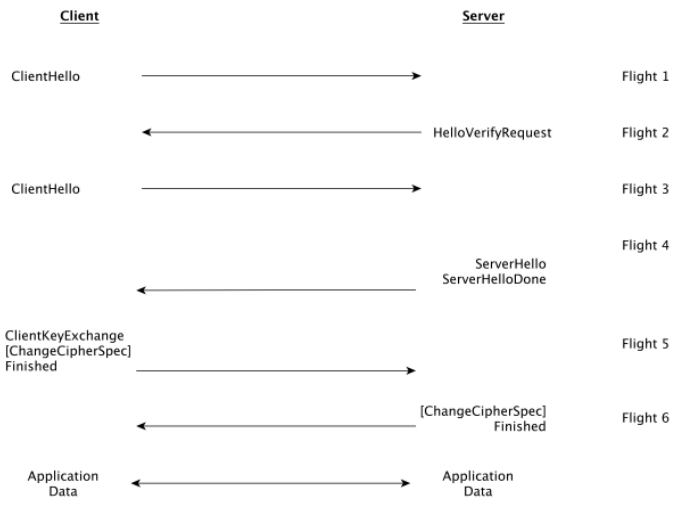

Figure 3.4: Handshake messages with pre-shared key

The handshake messages of figure 3.4 introduced data encryption with a key that is set up on the peer before the handshake starts. Now application data exchanged after the handshake is encrypted using the AES algorithm with key shared between the peers. This modification caused some changes to the previously implemented exchange algorithm in iteration 1, that have been refactored keeping the code as clean and easy as possible to easily add new functionalities.

3.6.4 DTLS with ECDHE key argument

Pre-shared key is inconvenient in some scenarios; first and foremost, the ability to reuse the same key should be avoided, because it might be compromised and might help the attacker. With this iteration it introduced a variant of the Diffie-Hellman key exchange, which is based on asymmetric cryptography and elliptic curves. The Diffie-Hellman key exchange enables two parties to exchange a key based on a shared secret included in the DTLS transmitted handshake messages, which is derived from the discrete logarithm. A more efficient way is to perform the key exchange utilizing elliptic curves. The Elliptic Curve Diffie-Hellman (ECDH) allows, in fact, to execute the key exchange using asymmetric cryptography generating a key pair based on a point (selected by the server) on a previously negotiated elliptic curve. This can be obtained if the peer has a certificate with an elliptic curve key used as shared secret that will generate a pre master secret on which future keys are derived. This method does not offer though future secrecy, since the same shared the secret is utilized in multiple handshakes (even though the point on the elliptic curve selected the server's information evolves over time). A stronger way to perform the ECDH is to generate on every key exchange a key pair (action performed by the server) and utilize that shared secret to perform the Diffe-Hellman. This is obviously more computationally expensive than using the certificate's key, but it prevents the reuse of the same key, as it is generated anew each time. This represents the key exchange that is needed for the cipher suite that will be finalized in the upcoming iteration, following the guidelines outlined in the DTLS RFC.

To facilitate this key exchange, it is essential for the server to possess a certificate, as its private key will be utilized for signing the hash of the parameters transmitted within the Server Key Exchange. A certificate hierarchy is then built, with a self-signed certificate as top Certification Authority (CA), and two end point certificates, certified by the leading Certificate Authority

(CA), which will be provided to both the server and the client (the client will have a certificate if necessary mutual authentication with certificates; this is optional). This represents a streamlined schema that works as proof of concept; in real applications the organization might pay the CA to issue certificates. For this purpose, this was not needed since to confirm a certificate, the "signing path" is checked to determine if it leads to one of the top CA's certificates is stored locally, and if the self-signed certificate is also present, it becomes possible to validate signatures. The process of generating the certificate hierarchy must be performed before the handshake takes place. The certificates and the private keys are kept in a key store for the purpose of being read by the DTLS protocol. It is then needed a refactor of the DTLS server that now needs, during the initialization phase, to retrieve and store the certificates in its context, to be utilized during the handshake. Once the previous steps are implemented, we can move forward with the Server Key Exchange handshake message. The server must select a point on the elliptic curve (from which the public key exchange originates key is obtained and sent) and the parameters specifying to inform the client about the type of elliptic curve that has been utilized. The server will then sign the public key with its private key, so that after the client has received the server's certificate is capable of ensuring integrity if the previous steps and the signature are validated, the client can generate a key pair and calculate the pre master secret from it. It will then send its public key to enable the server to generate the same pre master secret, and proceed with the Change Cipher Spec indicates that both peers can now generate session keys using the same pre-master secret acquired during the key exchange.

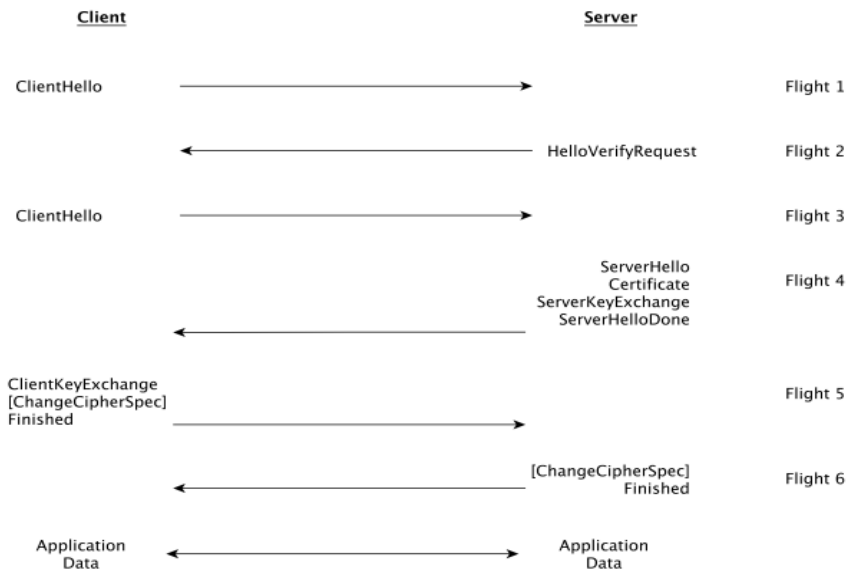

Figure 3.5: Handshake messages with ECDHE key exchange

Figure 3.6 shows the handshake messages for the third iteration. This iteration brought the implementation close to the final stage, since the ECDHE key exchange required the introduction of several new messages (among which includes the Server Key Exchange), and also the improvement of existing ones (such as the introduction of extensions for Client Hello and Server Hello messages), which are utilized in the subsequent and final iteration.

3.6.5 DTLS through mutual authentication

In the preceding phase, a certificate hierarchy was established. These certificates include an Elliptic Curve Digital Signature Algorithm (ECDSA) public key, which the server employs for the ECDHE_ECDSA key exchange. Mutual authentication is facilitated by this setup, in which also the client has a certificate and a private key (so that not only the client can authenticate Both the server and the client are authenticated in this process) needed to allow the DTLS client must access its own certificates and private key to enable mutual authentication, the server sends a Certificate Request message to the client, which responds with a Certificate Verify message.

The Certificate Request message, sent right after the Server Key Exchange message, includes the hash and signature algorithms that the server can verify, and the list of Distinguished Name (DN) acceptable (or an empty list if all DN want to be considered valid). The client Certificate holds the client's end-user certificate, which the server must validate to authenticate the client. Following the client certificate message, the Certificate Verify message is sent. The latter sends a

signed hash from all the preceding messages (excluded the first Client Hello and the Hello Verify). The hash is computed with a different algorithm than the one used for the Finished message (verify data), so refactoring is required the hashing procedure to update both hash algorithms during the handshake. The outcome is subsequently signed using the client's private key (needed for mutual authentication) and validated by the server. The server not only verifies the signature using the client's public key received in the prior message (client Certificate), but it has its own hash that verify against the one provided in the Certificate Verify. In this way the client can be authenticated, proving that the sender is indeed the rightful owner of the previously sent certificate (as signatures cannot be verified using a public key that is not paired with the corresponding private key). This leads us to the final cipher suite: TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8.

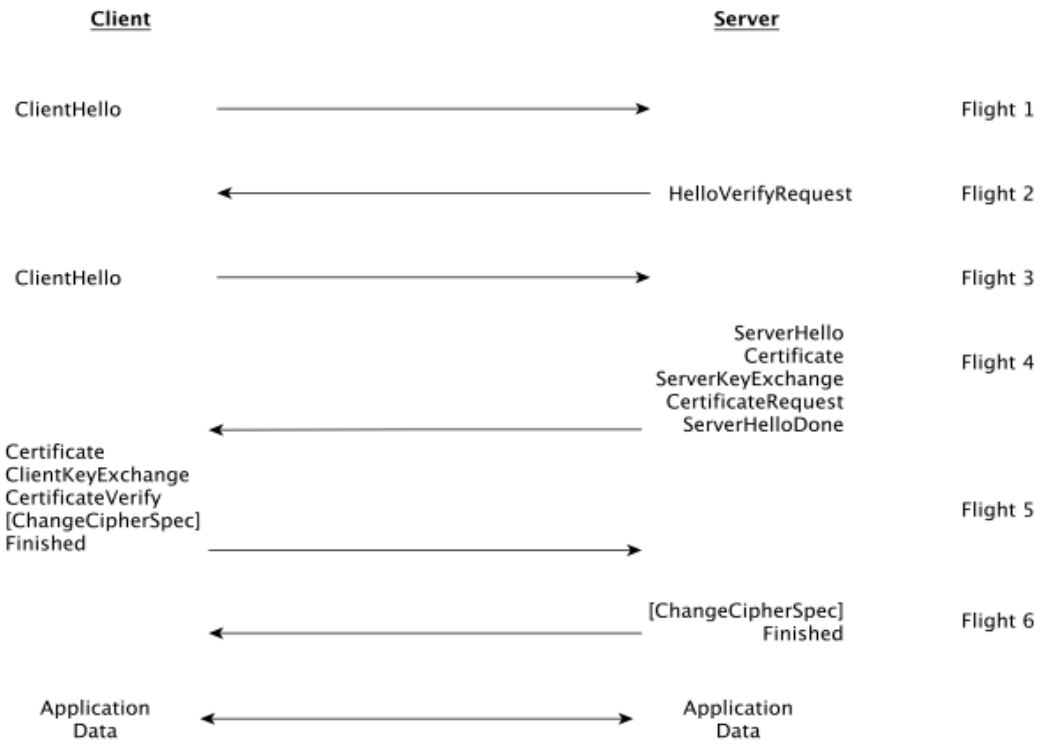

Figure 3.6: Handshake messages for the last cipher suite recommended by the IETF for the CoAPS protocol.

3.6.6 ECDSA Signature Generation and verification Algorithm

For interaction between the 6LBR and internet we use public-key exchange algorithm which

refers to the Elliptic Curve Digital Signature Algorithm (ECDSA). This algorithm is one of the public key exchange methods used in ECC algorithm, it uses x.509 certificate of ECC.

Consider a scenario where an internet host intends to transmit a signed message to local entities (WSN nodes). They must first reach a consensus over the curve's constraints (CURVE, G, p). In addition to the curve's field and equation, we require G, a base point of prime order on the curve, where p represents the multiplicative order of point G. A digital signature facilitates the recipient to verify a message's authenticity using the public key of the sender. At the outset the variable-length message is transformed into a fixed-length message digest using a secure hash algorithm. Once the message digest is calculated, a random number generator is utilized to create a value k for elliptic curve calculations.

Table 3.1: Lists of parameters used in Pseudo code of signature generation and verification algorithms

Parameter	Description
CURVE	Elliptic curve pitch and equation usages
G	Elliptic curve base point, serving as the generator for the elliptic curve with a large prime order p
P	Integer order of G, means that px G=0
Lp	The bit length of the group order p
K	random integer from interval [1, p-1]
r, s	Signature integers r and s
Da	Private key integers
QA	Public key curve point

Before an ECDSA authenticator can function, it needs to be aware of its private key. The corresponding public key is derived from this private key along with the domain parameters. Both the private and public key pairs should be securely stored in the authenticator's memory. As the name suggests, the private key remains inaccessible to external entities, while the public key must be readily available.

The internet host generates a key pair that includes a private key, represented as an integer dA, randomly chosen from the interval [1, p-1]; and a public key, represented as the elliptic curve point QA = dA x G. Here, x signifies the multiplication of the elliptic curve point by a scalar.

Upon receiving message M as input, a hash of the message is computed using a cryptographic

hash function. The most significant bit length of the group order p is then selected. A cryptographically secure random integer k is chosen from the range [1, p-1], and this integer is used to calculate the corresponding curve point (x, y). Using this curve point along with the private key, the public key is subsequently derived and finally, the pair of signatures (r, s) displayed as output. Generally, here is how the ecdsa key generation algorithm works; an arbitrary number originator is started and when its operation is completed, provides the numeric value that serves as the private key d. Following this, the public key Q(x,y) is calculated based on these values.

A. Pseudo code of Elliptic curve Signature generation algorithm

For signing message m by sender, A, using A's private key dA

Input*: receiving messageM*

Process*:*

BEGIN

Step 1: Calculate e=h(m), where h(m) is a SHA-2

Letz bethe Lp leftmost bits ofe, whereLp is the bit distance of the set order p then

Step 2: Select cryptographically secure random integer k from [1,p-1]

Step 3:Calculate the curve point(x1, y1)=kxG(x,y)mod p

Step 3: Calculate r = x1 mod p. If r=0, go back to phase 2.

Step 4: Calculate s = k^-1(z +rdA) mod p.Ifs =0, go backto step2

Step 5: the sign is the pair (r,s)

END

Output: signature pair(r,s)

Diagram illustrating the process of the Elliptic Curve Signature Generation Algorithm

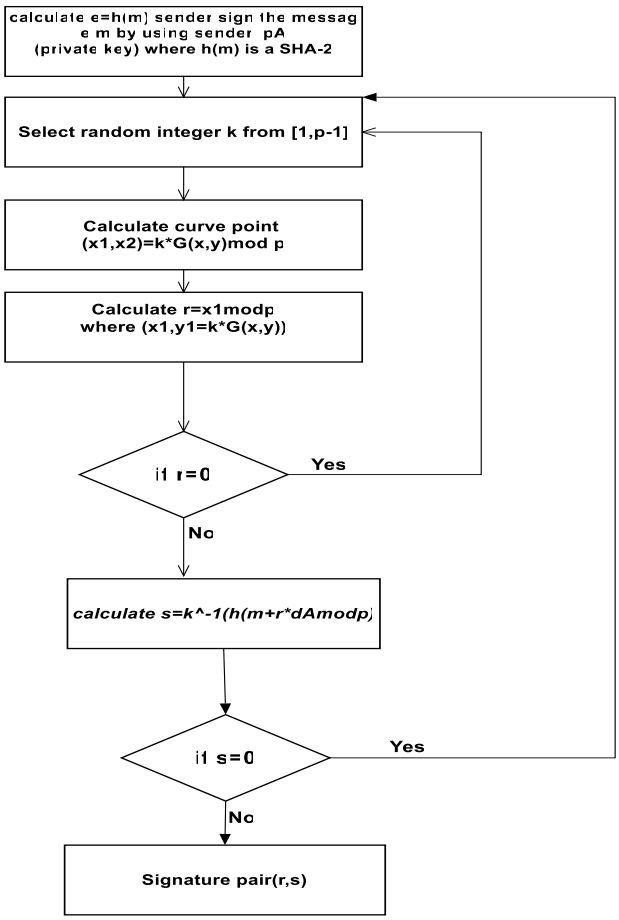

Figure 3.7: The process of the Elliptic Curve Signature Generation Algorithm

Signature verification serves as the counterpart to signature computation. Its goal is to validate the message utilizing the authenticator's public key. Utilizing the same secure hash algorithm employed in the sign generation step, the message digest signed by the authenticator is computed. This information, along with the public key Q (x, y) and the digital signature components r and s, is utilized in the verification process. The pseudo code below outlines the signature verification algorithm based on ECDSA. The inputs include the message digest h(m), the base point G (x, y), the public key Q (x, y), and the signature components r and s. The validity of the generated signature is confirmed using the same hash function employed during the signature generation. Below is the pseudo code for the Elliptic Curve Digital Signature verification algorithm.

For B to Authenticate a's Signature, B must have a's Public key QA Input*: receiving signature pair (r, s)*

Process*:*

BE
GI
N
Step 1. Verify that r and s are integers in [1, p-1]. If not, the signature is invalid then
Step 2: Calculate e=h(m), where is the same hash function used in the signature generation. Let z be the Lp left most bits ofe.
Step 3: Calculate w =s^-1 mod p.

Step 4: Calculate u1 =zwmod p and u2 =rwmod p.

Step 5: Calculate the curvepoint(x1, y1) =u1xG +u2 xQA. Step 6:
If (*r =x1(mod p)){*
The signature is valid
Else *{*
Invalid signature

}
End if
End

The flow diagram illustrating the Elliptic Curve Signature Verification Algorithm

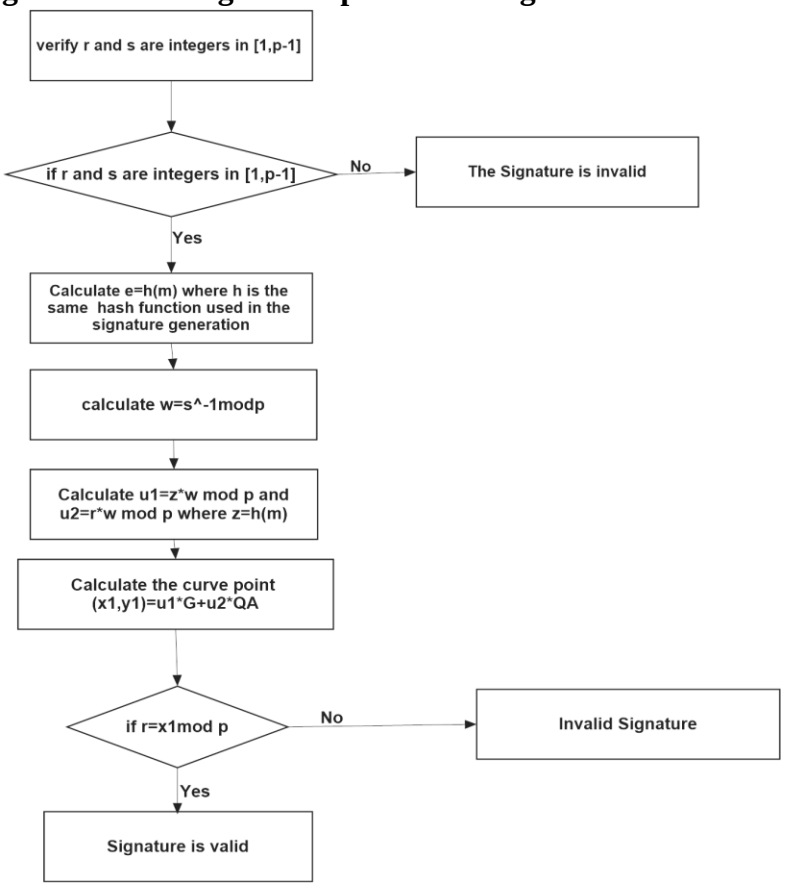

Figure 3.8: Th Elliptic Curve Signature Verification Algorithm

3.7. Performance Metrics and Routing parameters

In terms of performance evaluation experiments, we used wismote with ContikiOS. RPL has been our routing protocol of choice. The 6LBR(GW) are running the rpl-border-router provided by ContikiOS and therefore, they are the RPL DODAG roots for their subnets, delegate the global IPv6 prefix and route traffic to and from the constrained networks (WSNs). Every other node has RPL enabled and is running the Erbium server, which represents a CoAP server implementation. For the CoAP client's we used cupper (cu) which is Firefox plug-in used as CoAP-agent.

We have used two standard performance metrics: Energy consumption and Memory Footprint, to assess the effectiveness of our work.

The first performance metrics is Energy Consumption. We employ various percentages of the packet reception ratio, which drives most of the power usage in sensor nodes, to make accurate energy estimations. Furthermore, we take the constant percentage of RX to facilitate a comparison among all the nodes in the whole network setup. To compute the power consumption, we use the mechanism of Power-trace system available in ContikiOS and energest method. So, by using power state tracking method, Power-trace provides estimation for a system's power usage. Second performance metrics is memory footprint, which serves as one of the security performance metrics for the resource constrained and internet-integrated scenarios. To measure memory (ROM and RAM) usage in Cooja simulator there are many tools like; msp430-size, objdump-size and etc. among these tools we used msp430-size tool for our work because it estimates accurate usage of overall ROM and RAM of simulated.

3.8. Phases of Proposed Solution

As we discussed previously, our security architecture doesn't support required cipher suit, to overcome the limitation (stated in the above section 3.2 & 3.3), we modified TLS_PSK_WITH_AES_128_CCM_8 in our suggested fix to allow the 6LBR to transmit the pre-master secret to the sensor. The pre-master secret key received from the internet client is relayed to the CoAP server. To ensure adequate security for WSN communications, we implement the aforementioned authentication protocol.

The key stages, or message exchanges, of the proposed mediated DTLS handshake are outlined as follows:

1. The 6LBR intercepts the initial Client Hello message, responding with a Client Hello Verify to safeguard the WSN domain against DoS attacks. The Client Hello message sent

back by the internet client carries the client arbitrary value, together with the protocol version and the list of supported cipher suites.

2. Through employing the proposed WSN authentication protocol, the 6LBR acquires an initial ticket from the AC server, together with details on the AC to reach out to for accessing the target sensing device. It is important to note that, in this context, both the AC and the 6LBR are regarded as part of the same WSN domain. From the AC, the 6LBR receives a ticket for the CoAP service, information regarding the cipher suites maintained by the sensor node, as well as its digital certificate and current IPv6 address.
3. The original Client Hello message is relayed to the intended CoAP device, accompanied by a request for pre-shared key-based authentication. The Server Hello response is then sent back to the internet client, this time confirming the use of public-key authentication. The Server Key Exchange message forwarded in this flight transports the server random value.
4. To ensure mutual authentication as per our objectives, the 6LBR client is authenticated by requesting its certificate. The Client Key Exchange message from the client carries the random value lengthways with the premaster secret key that the client has generated.
5. The WSN authentication protocol enables the retrieval of a secret key to be shared between the 6LBR and the destination CoAP sensing device. This key is utilized for the secure transmission of the pre-master secret key to the server. The next message flight allows finalizing the handshake between the clients and sensing device.
6. Internet host (CoAP client) generate a key pair is generated, which includes a private key integer dA and a randomly chosen integer k within the interval [1, p-1], along with a public key represented as a curve point. The Internet host (CoAP client) then sends a signed message m to the CoAP server.
7. On receiving signed message, initially, they must reach an agreement on the curve parameters as will be discussed later at algorithm part and CoAP server verify the validity of signature
8. Finally, secured and trusted end-to-end communication preformed system.

CHAPTER FOUR

RESULTS AND EVALUATION

4.1. Introduction

This chapter presents a discussion of the simulation results for two scenarios such as Normal DTLS and Modified DTLS. The discussion includes Results of the simulation for the proposed system architecture, WSN nodes neighbors of 6LBR, CoAP client/server communications, the energy consumption performance (power consumption), performance analysis of memory allocation (i.e. memory footprint) and validation of the results of the simulation.

4.2. Simulation Result of proposed system architecture

The output of the above simulation is shown below on the command line; after simulation is started, the connection between WSN nodes and Linux host doesn't establish. So, to initialize the communication between them we used tunslip6 services which use port 60001 on Linux hosts. Steps to make connection between WSN nodes and Linux host using 6lbr.

The 6LOWPAN border router connection established between client and server, to facilitate authentication and key exchange. The simulation employs a DTLS handshake that supports delegated ECC public-key authentication. During this process, the 6LBR transparently intercepts DTLS handshake messages, executing the handshake in two phases. The 6LBR manages the handshake and performs ECC cryptographic operations on behalf of CoAP-constrained sensing devices. A CoAP Internet client establishes a secure communication session with a CoAP server located in a sensing device, and the handshake also accommodates the reverse scenario.

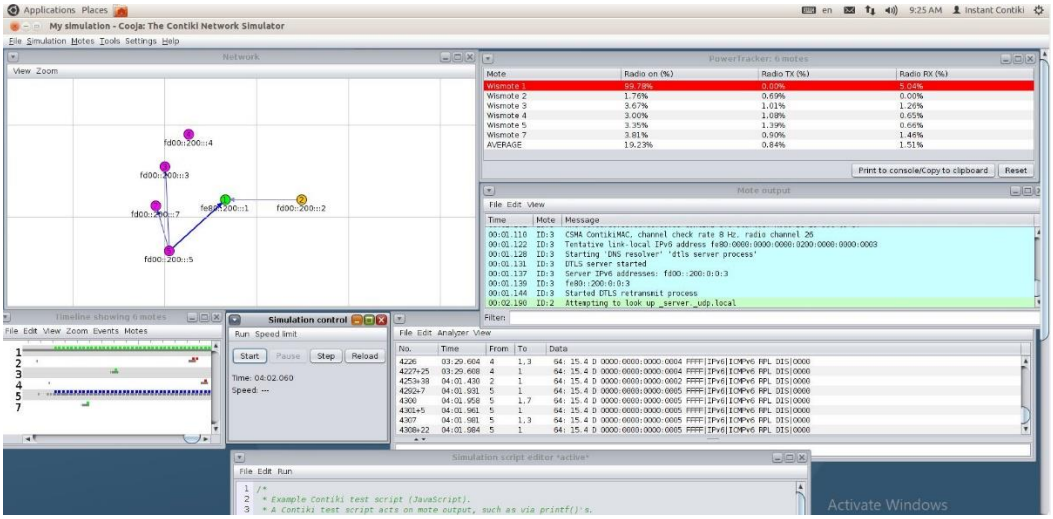

Figure 4.1 Simulation result of proposed system architecture

After established connection between WSN nodes and Linux host, we can ping and browse on the web interface using **Server IPv6 addresses:[aaaa::200:0:0:1]/**and checks the neighbors of 6lbr as shown below .

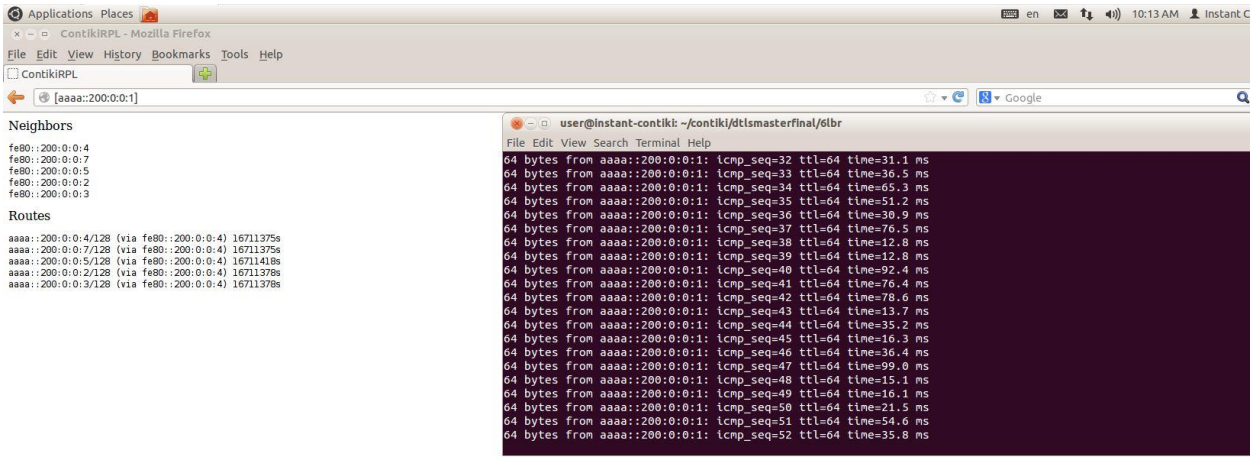

Figure 4. 2 lists of WSN nodes neighbors of 6LBR

Figure 4.2 able to see IPv6 address of WSN nodes, with their respective routes.

After end-to-end communication between entities is secured with DTLS protocol, CoAP client access resources on the CoAP server.

4.3. Performance of power consumption

We employ various percentages of the packet reception ratio, which drives most of the power usage in sensor nodes, to get accurate energy estimations. Furthermore, we take the constant percentage of RX to facilitate a comparison among all the nodes in the whole network setup. To compute the power consumption, we use the mechanism of Power-trace system available in ContikiOS and energest method. So, by using power state tracking method, Power-trace provides estimation for a system's power usage. To estimate the power consumption. We have the choice of using a genuine device or Cooja. For r time (RTIMER_SECOND = 32768), find the count of ticks per second and print f("Ticks per second: %u", RTIMER_SECOND); Include power trace app in the project by adding it to the Make file APP += powertrace

Add to source file

"powertrace.h" is #included. Each ten seconds, add the following to the source file to print the power profile: powertrace start * 10 CLOCK_SECOND;

Then here is how power consumption is calculated:

Energy consumption (Power - mW): (rxend – rxstart) * current * voltage / RTIMER_SECOND / runtime(seconds)

Refer to the datasheet for current and voltage information, e.g., CPU = (531519 - 512803) * 0.33 * 3 / 32768 / 10

You obtain the energy used during runtime if you do not divide by runtime.

Table 4.1: Comparison of Power Consumption: DTLS_Modified vs. DTLS_Normal

Scenarios	**Radio ON (%)**	**RadioTX (%)**	**Radio RX (%)**
DTLS_Modified	18.11%	0.39	0.67
DTLS_Normal	27	0.64	0.89

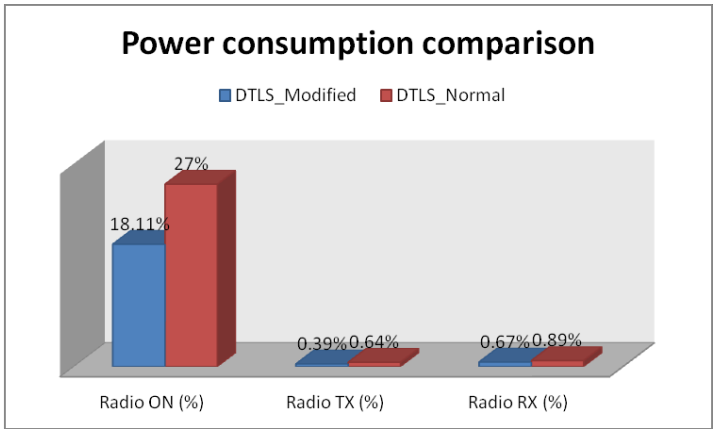

Figure 4.3 Comparison of Power Consumption: DTLS Modified vs. DTLS Normal

Comparison of Power Consumption Between Two Scenarios; such as DTLS_ Normal (basic DTLS) and proposed one (DTLS Modified). So according to simulation result found, the normal one consumes more power. In contrast to this our proposed protocol (DTLS Modified) consumes less power than basic one. As depicted on Figure 4.3 the modified one out performs well in power consumption comparison.

4.4. Memory Footprint of End-to-End Security

Our initial evaluation focuses on the RAM and ROM memory needed to implement end-to-end security at the transport layer using the proposed research solutions, particularly due to their limited availability on sensing platforms like Wismote. We emphasize that memory is a crucial factor in determining effectiveness of new research solutions addressing end-to-end security in

the context of internet of things, which have great impact on resource-constrained devices. The impact of the support for the two end-to-end security modes in respect to its usage of memory on a wismote mote sensing platform, and also a fundamental usage scenario with existing DTLS-based standard end-to-end security, which provides a basis for comparison. So, to analyze the memory footprint we use objdump-size (msp430-size) tool and done comparison among our scenarios.

Table 4.2: Memory Usage Comparisons between DTLS and DTLS Modified

Scenarios	**% of Total ROM**	**%Total RAM**
DTLS Normal	53	68.3
DTLS Modified	46.83	59.78

The memory footprint comparison between two scenarios; such as DTLS Normal (basic DTLS) and proposed one (DTLS Modified). So according to simulation result found, In the normal scenario, there was higher memory consumption in both RAM and ROM. However, in the case of our proposed protocol (DTLS Modified) less memory space was consumed. As noted from comparison result the modified one out perform good in memory usage also.

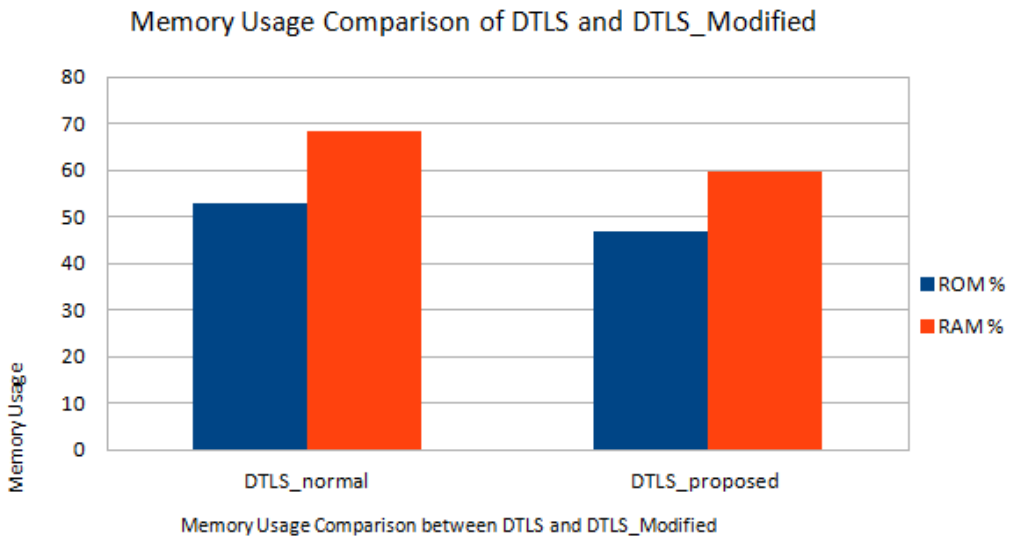

Figure 4.4 Memory Usage Comparison between DTLS and DTLS Modified

Fig 4.4 illustrates that memory footprint comparison between two scenarios; such as DTLS Normal (basic DTLS) and proposed one (DTLS Modified). So according to simulation result found, in the normal scenario more memory was consumed both in RAM and ROM. However, in the case of our proposed protocol (DTLS Modified) less memory space was consumed. As noted from comparison result the modified one out performs well in memory usage also.

Overall, we notice that hardware-level encryption presents considerable memory overhead, particularly in ROM. The memory limitations of the TelosB and Wismote are apparent, as additional ROM is required to fully support end-to-end security using the original CoAP Certificates security mode. RAM could also become a limiting factor in scenarios where larger applications require more memory from the sensing device. This challenge extends to the storage and processing of X.509 certificates and their associated public keys.

4.5. Maximum communications rate (computational time)

The computational time required to implement security directly affects the maximum communication rate that a sensing device can achieve. IoT applications may face challenges if security measures demand excessive resources, particularly in terms of computation. To assess energy consumption, we experimentally measure the computational time required for the proposed mechanisms. As anticipated, the time needed for the mediated DTLS handshake (10.39 ms) is significantly shorter than that for the original handshake (15.39 ms), primarily due to the computational impact of ECC. We can analytically ascertain the maximum number of CoAP requests per hour that a device can handle while ensuring end-to-end security.

Table 4.3: The time needed to facilitate the mediated DTLS handshake

Scenario	Computational time
DTLS_Modified	10.39ms
DTLS_Normal	15.39 ms

4.6. Validation of Simulation Result

There are several methods associated with testing, verification, and validation. To ensure the accuracy of the network models and the interpreted simulation results, we employed an analytical approach. The verification of simulation results revealed that the power consumption performance derived from both the analytical analysis and the simulation is nearly identical. Thus, we can confirm that the network models and simulation results are accurate. To validate the simulation results, we utilized statistical techniques in our analysis. Additionally, the steps taken to validate the simulation results were also implemented in this study. The findings indicate that the proposed DTLS protocol uses less power compared to the basic DTLS.

The findings from the study conducted by (Kothmayr et al., 2013) on the evolution of memory and energy performance suggest that the memory and computational power required for end-to-end security are suboptimal. This result aligns with our work, which highlights improved

memory usage and power consumption performance.

CHAPTER FIVE

CONCLUSIONS AND FUTURE WORK

This study aims to propose a DTLS-based approach for end-to-end security in COAP communication within the Internet of Things. Authentication relies on the exchange of X.509 certificates containing ECC keys and a premaster secret key (PMSK), which is established during a fully authenticated DTLS handshake.

The proposed end-to-end security mechanisms were evaluated experimentally with a focus on two key aspects: their impact on the power consumption of sensing devices and their memory footprint. We regard these two factors as essential for assessing the effectiveness of any security proposal for constrained wireless sensing platforms.

The research solutions in this paper enhance end-to-end security for LoWPAN devices by efficiently supporting ECC authentication and key agreement. Delegating ECC computations to a more powerful device, such as the 6LBR, proves effective despite the additional overhead from the LoWPAN authentication protocol. Our evaluation confirms that this security architecture ensures message integrity, confidentiality, and authenticity while maintaining low power consumption and memory overhead. This approach significantly contributes to IoT security, achieving key goals in constrained environments and preserving device resources, distinguishing it from existing solutions.

Our future research will focus on real-world evaluations and the development of enhanced security mechanisms based on the integration model.

References

Adeeba Khan. (2016). OVERVIEW OF SECURITY IN INTERNET OF THINGS.

Ali Hussein, Imad H. Elhajj, Ali Chehab, & Ayman Kayssi. (2016). Securing Diameter: Comparing TLS, DTLS, and IPSec.

Angelo Capossele, Valerio Cervo, Gianluca De Cicco, & Chiara Petrioli. (2015). Security as a CoAP resource: an optimized DTLS implementation for the IoT.

Brachmann, M., Garcia-Morchon, O., Keoh, S.-L., & Kumar, S. S. (2012). Security Considerations around End-to-End Security in the IP-based Internet of Things.

Cirani, S., Ferrari, G., & Veltri, L. (2013). Enforcing security mechanisms in the IP-based internet of things: An algorithmic overview. Algorithms, 6(2), 197–226. https://doi.org/10.3390/a6020197

Cisco IBSG. (2011). The Internet of Things How the Next Evolution of the Internet Is Changing Everything.

Darrel Hankerson, Scott Vanstone, & Alfred Menezes. (2004). Guide to Elliptic Curve Cryptography. In Guide to Elliptic Curve Cryptography (1st ed.). Springer New York, NY. https://doi.org/10.1007/b97644

D.UthayaSinthan, & M.S.Balamurugan. (2013). DTLS & COAP Based Security For Internet of Things Enabled Devices. C)International Journal of Engineering Sciences & Research Technology, 2(12), 3733–3738. http://www.ijesrt.com

Gao, D., Foh, C. H., Yang, O. W. W., Sun, X., & Lai, C. F. (2012). IP-enabled wireless sensor network. In International Journal of Distributed Sensor Networks (Vol. 2012). https://doi.org/10.1155/2012/851426

Granjal, J., & Monteiro, E. (2016). End-to-end transparent transport-layer security for Internet-integrated mobile sensing devices. 2016 IFIP Networking Conference (IFIP Networking) and Workshops, IFIP Networking 2016. https://doi.org/10.1109/IFIPNetworking.2016.7497235

Jorge Da Costa Granjal, A. (2014). END-TO-END SECURITY SOLUTIONS FOR INTERNET-INTEGRATED WIRELESS SENSOR NETWORKS.

Khan, A. (2016). OVERVIEW OF SECURITY IN INTERNET OF THINGS.

Khan, R., Khan, S. U., Zaheer, R., & Khan, S. (2012). Future internet: The internet of things architecture, possible applications and key challenges. Proceedings - 10th International Conference on Frontiers of Information Technology, FIT 2012, 257–260. https://doi.org/10.1109/FIT.2012.53

Kothmayr, T., Schmitt, C., Hu, W., Brünig, M., & Carle, G. (2013). DTLS based security and two-way authentication for the Internet of Things. Ad Hoc Networks, 11(8), 2710–2723. https://doi.org/10.1016/j.adhoc.2013.05.003

Lakkundi, V., & Singh, K. (2014). Lightweight DTLS Implementation in CoAP-based Internet of Things. https://doi.org/10.1109/ADCOM.2014.7103240

Lessa Dos Santos, G., Guimaraes, V. T., Da Cunha Rodrigues, G., Granville, L. Z., & Tarouco, L. M. R. (2016). A DTLS-based security architecture for the Internet of Things. Proceedings - IEEE Symposium on Computers and Communications, 2016-February. https://doi.org/10.1109/ISCC.2015.7405613

Mehdipour, F. (2020). A Review of IoT Security Challenges and Solutions. In 2020 8th International Japan-Africa Conference on Electronics, Communications, and Computations (JAC-ECC), IEEE, 1–6.

M. Gamundani, A. (2014). An Algorithmic Framework Security Model for Internet of Things. International Journal of Computer Trends and Technology, 12(1), 16–20. https://doi.org/10.14445/22312803/IJCTT-V12P105

Miorandi, D., Sicari, S., De Pellegrini, F., & Chlamtac, I. (2012). Internet of things: Vision, applications and research challenges. In Ad Hoc Networks (Vol. 10, Issue 7, pp. 1497–1516). Elsevier B.V. https://doi.org/10.1016/j.adhoc.2012.02.016

Omar Said. (2013). Development of an Innovative Internet of Things Security System. www.IJCSI.org

Petersen, H., Baccelli, E., & Wählisch, M. (2014). Interoperable Services on Constrained Devices in the Internet of Things. In Interoperable Services on. https://hal.inria.fr/hal-01058636

Prachi Sharma, & S.V. Pandit. (2014). ENERGY EFFICIENT AND LOW COST ORIENTED HIGH SECURITY METHOD FOR MANET A Review.

Praveen Kumar Kamma, Chennakeshava Reddy Palla, Usha Rani Nelakuditi, & Ravi Sekhar Yarrabothu. (2016). Design and Implementation of 6LoWPAN Border Router.

Raza, S., & Mälardalens högskola. (2013). Lightweight security solutions for the internet of things. School of Innovation, Design and Engineering, Mälardalen University.

Reem Abdul Rahman, & Babar Shah. (2016). Security analysis of IoT protocols: A focus in CoAP. IEEE.

Rescorla. (2012). Datagram Transport Layer Security Version 1.2. Internet Engineering Task Force (IETF).

Roman, R., Zhou, J., & Lopez, J. (2013). On the features and challenges of security and privacy in distributed internet of things. Computer Networks, 57(10), 2266–2279. https://doi.org/10.1016/j.comnet.2012.12.018

Shelby. (2014a). ARM IoT Tutorial.

Shelby. (2014b). The Constrained Application Protocol (CoAP).

Suo, H., Wan, J., Zou, C., & Liu, J. (2012). Security in the internet of things: A review. Proceedings - 2012 International Conference on Computer Science and Electronics Engineering, ICCSEE 2012, 3, 648–651. https://doi.org/10.1109/ICCSEE.2012.373

Sye Keoh, S. K. Z. S. (2013). Profiling of DTLS for CoAP-based IoT Applications draft-keoh-dtls-profile-iot-00. http://www.ietf.org/shadow.html

Toheed, Q., & Razi, H. (2010). Asymmetric-Key Cryptography for Contiki.

Trabalza, D. (2012). Implementation and Evaluation of Datagram Transport Layer Security (DTLS) for the Android Operating System.

Vignesh, K. (2017). Performance analysis of end-to-end DTLS and IPsec-based communication in IoT environments Security and Privacy ~ Distributed systems security. www.bth.se

Zachariah, T., Klugman, N., Campbell, B., Adkins, J., Jackson, N., & Dutta, P. (2015). The internet of things has a gateway problem. HotMobile 2015 - 16th International Workshop on Mobile Computing Systems and Applications, 27–32. https://doi.org/10.1145/2699343.2699344

Appendix A

======= DTLS Handshake Protocol Data structure==========

Struct {
HandshakeTypemsg_type;
Uint24 length;
uint16 message_seq;
uint24 fragment_offset;
uint24 fragment_length;
Select (HandshakeType) {
casehello_request: HelloRequest;
caseclient_hello: ClientHello;
casehello_verify_request: HelloVerifyRequest;
caseserver_hello: ServerHello;
case certificate: Certificate;
caseserver_key_exchange: Server KeyExchange;
casecertificate_request: CertificateRequest;
caseserver_hello_done: Server HelloDone;
casecertificate_verify: CertificateVerify;
caseclient_key_exchange: ClientKeyExchange;
case finished: Finished; 69 } body;
} Handshake;
========The HelloVerifyRequest data structure==========
Struct {
ProtocolVersionserver_version;
Opaque cookie<0..2^8-1>;

} HelloVerifyRequest;