ESCUELA POLITÉCNICA NACIONAL
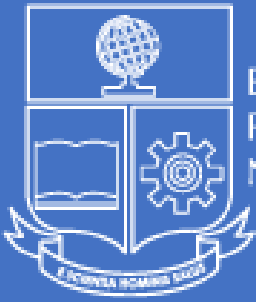
FACULTAD DE INGENIERÍA DE SISTEMAS

A SECURE AND LIGHTWEIGHT AUTHENTICATION PROTOCOL FOR GATEWAYS IN LORAWAN NETWORKS

THESIS SUBMITTED AS PART OF THE REQUIREMENTS FOR THE AWARD OF THE DEGREE OF DOCTOR OF PHILOSOPHY IN INFORMATICS

JHONATTAN JAVIER BARRIGA ANDRADE
jhonattan.barriga@epn.edu.ec

SUPERVISOR: PHD. SANG GUUN YOO PARK
sang.yoo@epn.edu.ec

QUITO, JANUARY 2023

THESIS

For the award of the degree of

DOCTOR OF PHILOSOPHY

IN INFORMATICS

Resolution RPC-SO-43-No.501-2014 of the Consejo de Educación Superior

Presented by

JHONATTAN JAVIER BARRIGA ANDRADE

Thesis supervised by **Dr. Sang Guun Yoo Park, Professor Escuela Politécnica Nacional (EPN).**

A SECURE AND LIGHTWEIGHT AUTHENTICATION PROTOCOL FOR GATEWAYS IN LORAWAN NETWORKS

Oral examination by the following committee:

Jenny Gabriela Torres Olmedo, Ph.D.
Escuela Politécnica Nacional (EPN)

Martha Cecilia Paredes Paredes, Ph.D.
Escuela Politécnica Nacional (EPN)

Luis Felipe Urquiza Aguiar, Ph.D.
Escuela Politécnica Nacional (EPN)

Michele Nogueira Lima, Ph.D.
Federal University of Minas Gerais, External Member

Efraín Rodrigo Fonseca Carrera, Ph.D.
Universidad de las Fuerzas Armadas (ESPE), External Member

DECLARATION

I hereby declare under oath that I am the author of this work, which has not previously been presented for obtaining any academic degree or professional qualification. I also declare that I have consulted the bibliographic references included in this document.

Through this declaration, I transfer my intellectual property rights corresponding to this thesis, to the Escuela Politécnica Nacional, as established by the Intellectual Property Law of Ecuador, its Regulations and the current institutional norms. I declare that this work is based on the following articles of my authorship (as main author or co-author) related to the title of this thesis.

- Yoo, S.G., Barriga, J.J. (2017). *Privacy-Aware Authentication for Wi-Fi Based Indoor Positioning Systems*. In: Batten, L., Kim, D., Zhang, X., Li, G. (eds) Applications and Techniques in Information Security. ATIS 2017. Communications in Computer and Information Science, vol 719. Springer, Singapore.

- Barriga A, J.J., Yoo, S.G. *Security over Smart Home Automation Systems: A Survey.* In: Rocha, Á., Guarda, T. (eds) Developments and Advances in Defense and Security. MICRADS 2018. Smart Innovation, Systems and Technologies, vol 94. Springer, Cham.

- Barriga A., J.J., Yoo, S.G., Polo, J.C. (2019). *Enhancement to the Privacy-Aware Authentication for Wi-Fi Based Indoor Positioning Systems.* In: , et al. Applied Cryptography and Network Security Workshops. ACNS 2019. Lecture Notes in Computer Science(), vol 11605. Springer, Cham.

- Barriga, J.J., Yoo, S.G. *Internet of Things: A Security Survey Review on Long Range Wide Area Network (LoRaWAN).* Journal of Engineering and Applied Sciences, 2019, 14: 9774-9787.

- Barriga, J.J., Yoo, S.G. *Securing End-Node to Gateway Communication in LoRaWAN with a Lightweight Security Protocol.* IEEE Access, 2022, doi: 10.1109/ACCESS.2022.3204005.

I also declare that I have acknowledged the collaboration of third parties, and the contribution made by other published or unpublished material.

Jhonattan Javier Barriga Andrade

CERTIFICATION

I certify that JHONATTAN JAVIER BARRIGA ANDRADE has carried out his research under my supervision. To the best of my knowledge, the contributions of this work are novel.

PhD. Sang Guun Yoo Park
ADVISOR

ACKNOWLEDGEMENTS

To my supervisor and friend PhD Sang Yoo, for his support, patience and valuable advice during the development of this work.

To my beloved family Dany, Javi and Sofi for being an important pillar of my life.

To my colleague and friend MSC. Roberto Andrade for his continous support along this academic journey.

To my parents for their unconditional love and daily support.

To my student Juan Sulca for his great effort and profesionalism.

To the crew of SmartLab, Jose L., Diego P., Alejandro U., Jose G., for always striving and pushing the limits.

DEDICATED TO

God and my beloved family Dany, Javi and Sofi for supporting me during this new journey.
They have been my rock, inspiration and life with great love. They are kind, fill my heart and
nurture my soul.

Contents

List of Figures

List of Tables

Code list

PROLOGUE

The need to always be connected to the Internet, has led to the appearance and massification of the Internet of Things (IoT).

This has made it possible to interconnect different types of devices to monitor or control various ecosystems where there is information that helps humanity make decisions or make their lives easier. Likewise, this interconnection has generated the appearance of smart initiatives, such as smart cities, smart waste, smart agriculture, among others. These initiatives are based on the interconnection of IoT devices through the use of long-range wireless networks, low energy consumption and little computational cost known as Low Power Wide Area Network, LPWAN. Nowadays, there are several types of networks based on LPWAN like Sigfox, NB-IoT, LoRaWAN, among others.

The current work studies specifically LoRaWAN networks due its openness and ease of making changes over libraries that support IoT devices. LoRaWAN architecture comprises three main components: End-Nodes, gateways and backend servers; elements that perform different tasks within the infrastructure. On the one hand, LoRaWAN provides some security features to register End-Nodes and to protect data integrity and payloads; however, on the other hand, LoRaWAN specification does not cover all listed elements. For instance, gateways are one of such components that present security availability issues as they are qualified as "trusted", such devices lack of authentication over LoRaWAN. This vulnerability affects availability of End-Nodes.

This work presents a protocol that address gateway authentication by taking care of hardware resources and power constraints. Since, the use of lightweight cryptographic functions allows developing authentication protocols for devices that do not have high computing resources. The proposed protocol has been formally proved that it is secure, lightweight and that authenticates the gateway over LoRaWAN networks.

xiii

RESUMEN

En el presente trabajo analiza uno de los protocolos más utilizados en el mundo de IoT (LoRaWAN), el cual permite realizar aplicaciones smart de diversos tipos como: smart argriculture, smart parking, smart metering, entre otros. Aunque ofrece grandes posibilidades de poder realizar varios de aplicaciones, este protocolo está expuesto a vulnerabilidades de seguridad que pueden comprometer su confidencialidad, integridad y disponibilidad (CIA). Dentro de los elementos que componen la infraestructura LoRaWAN se encuentran los gateways, estos dispositivos a pesar de ser claves en el envío (uplink), recepción (downlink) y transformación de paquetes LoRa en TCP/IP, no poseen un mecanismo de autenticación para que puedan ser registrados dentro de la infraestructura de LoRaWAN. Esto representa un problema de seguridad porque usuarios mal intencionados podrían aprovecharse de esta debilidad para inyectar paquetes inválidos dentro de la infraestructura. Es por ello, que hemos considerado profundizar en este aspecto del protocolo para generar una solución que permita dicha autenticación y fortalezca además la relación entre el End-Node y el Gateway; y el Gateway con el Network Server.

Para llevar a cabo este trabajo, hemos utilizado la metodología Design Science (DS), con el objetivo de generar un nuevo artefacto (protocolo de autenticación). Se ha estudiado también en la literatura para investigar soluciones similares que se puedan dar a este tipo de problemas en tecnologías similares o inalámbricas. De la misma manera para probar el impacto que genera esta solución dentro de la infraestructura LoRaWAN, hemos realizado un experimento donde comparamos una versión de LoRaWAN contra nuestra propuesta.

Del análisis realizado, hemos identificado que este tipo de protocolos deben ser livianos en términos criptográficos, de bajo consumo energético y poco costo computacional puesto que serán implementados en dispositivos que tienen dichas características. Adicionalmente, debemos tener presente que el incluir elementos adicionales podría generar nuevas brechas de seguridad por lo que nos hemos basado en la especificación LoRaWAN 1.1 para extender sus capacidades de seguridad y poder así autenticar estos dispositivos dentro de la infraestructura LoRaWAN sin generar efectos colaterales en el resto de elementos. Finalmente, para validar el nivel de seguridad de este tipo de protocolos se aplicaron métodos formales: BAN Logic y Scyther Tool.

Palabras Clave: LoRaWAN, Autenticación de Gateway, Criptografía Liviana, BAN Logic, Scyther Tool.

ABSTRACT

This work analyzes one of the most used protocols in the world of IoT (LoRaWAN), which allows to carry out smart applications of various types such as: smart agriculture, smart parking, smart metering, among others. At the same time that it offers great possibilities of being able to carry out various applications, it is exposed to security vulnerabilities that can compromise its integrity, confidentiality and availability (CIA). Within the elements that make up the LoRaWAN infrastructure there are the gateways. These devices, despite being key in sending (uplink), receiving (downlink) and transforming LoRa packets into TCP / IP, do not have an authentication mechanism so that can be registered within the LoRaWAN infrastructure. This represents a security issue because malicious users could take advantage of this weakness to inject invalid packets into the infrastructure. That is why we have considered delving into this aspect of the protocol to generate a solution that allows such authentication and also strengthens the relationship between the End-Node and the Gateway; and the Gateway with the Network Server.

To carry out this work, we have used the Design Science (DS) methodology, with the aim of generating a new artifact (authentication protocol). We have also studied the literature to investigate similar solutions that can be given to this type of problem in similar or wireless technologies. In the same way, to test the impact that this solution generates within the LoRaWAN infrastructure, we have carried out an experiment where we compare a version of LoRaWAN against our proposal.

From the analysis carried out, we have identified that this type of protocol must be light in cryptographic terms, with low energy consumption and low computational cost, since they will be implemented in devices that have these characteristics. Additionally, we must bear in mind that including additional elements could generate new security gaps, which is why we have based ourselves on the LoRaWAN 1.1 specification to extend its security capabilities and thus be able to authenticate these devices within the LoRaWAN infrastructure without generating side effects in the rest of elements. Finally, to validate the security level of this type of protocol, formal methods were applied: BAN Logic and Scyther Tool.

Keywords: LoRaWAN, Gateway authentication, Lightweight Cryptography, BAN Logic, Scyther Tool.

Chapter 1

INTRODUCTION

The world of the Internet of Things (IoT) is growing exponentially. It is estimated that by 2025, there will be roughly 77 billion devices connected to the Internet [1–5]. This expansion has leveraged the emergence of certain smart initiatives (i.e., smart cities, smart campus, smart metering, smart factory, among others). These initiatives rely on IoT devices with limited computing and power capabilities that are providing information to back-end systems. IoT devices can transmit tiny amounts of information (small chunks of bytes) such as light intensity, temperature, capacity, presence among others. Although they do not require high-speed connectivity, they demand long life batteries [6], which means that data transmission and power optimization are important concerns to be addressed.

The use of lightweight secure wireless communication networks to interconnect and exchange data among IoT devices becomes crucial [7–9]. These wireless networks are known as Low Power Wide Area Network (LPWAN). LPWAN is a group of technologies / standards that provide long-range wireless communications, reduced energy consumption that translates into long-lasting batteries (approximately 10 years), inexpensive deployment costs (devices start at five dollars [9]), adequate transmission rates and capacity to adapt to licensed and unlicensed spectrum [10, 11]. It also provides multiple options for achieving IoT connectivity, such as LTE-M, SigFox, NB-IoT, Long Range (LoRa), Long Range Wide Area Network (LoRaWAN), Weightless-N, and EC-GSM. Among them, the most used are Long Range Wide Area Network (LoRaWAN), SigFox and Narrow Band–Internet of Things (NB-IoT) [7–9,12]. They have similarities in terms of architecture but differ in other parameters such as frequency of operation, security, connection fees, among others.

In terms of research, LoRaWAN possesses more benefits compared to other LPWAN networks. First, its level of openness facilitates researchers performing customizations in most of its components. While LoRaWAN uses a free fee spectrum, Sigfox uses a licensed one and NB-IoT needs to be deployed over a cellular-network [10]. In addition, LoRaWAN does not require additional third-party infrastructure deployments (back end components) to be used.

Everyone is free to implement its own private network using open source tools. LoRaWAN presents an architecture composed of End-Nodes, gateways and back-end servers (Network Server *NS*, Join Server *JS* and Application Server *AS*) which is similar to Sigfox and NB-IoT.

Despite LPWAN standards follow data transmission and power optimization requirements; they define simple protocols to support authentication, security and privacy features because of computing and energy constraints of IoT devices. Such constraints represent vulnerabilities that could be exploited to conduct cyberattacks affecting confidentiality, integrity, or availability (CIA triad) [11]. From the literature, several vulnerabilities at LoRaWAN, affecting CIA triad, have been discussed [13–21]. Although there is a specification that describes features and security concerns for LoRaWAN v1.1 [22], it does not cover all architecture components like gateways. Gateways are key elements in LoRaWAN as they are in charge of translating LoRa into TCP/IP data as well as transporting data from End-Nodes to back-end servers. In this work, we highlight the fact that this components are not authenticated over LoRaWAN and are asumed to be trusted according to the specification of LoRaWAN v1.1. This lack of authentication presents availability issues for End-Nodes and back-end infraestructure [15,21].

This chapter is structured as follows. First, we describe the problem to be addressed in this work. Then, we provide the objectives pursued in this research. Later, we describe the research methodology used. Next, we present our main contributions generated through this work. Finally, the structure of the remaining chapters is presented.

1.1 Problem Statement

LoRaWAN provides several benefits in terms of applications that could be developed compared to other technologies like Sigfox or NB-IoT. However, it is exposed to security vulnerabilities that might compromise confidentiality, integrity or availability of data and infraestructure as described in [12]. These vulnerabilities target several elements of LoRaWAN like End-Nodes, gateways or back-end infrastructure. Although LoRaWAN has gone through a series of improvements to increase the level of security (i.e. new keys within derivation process), there are still open issues related to gateways since they are considered "trusted" devices according to the specifications published by LoRa Alliance [22,23]. In this specifications, there is a mutual authentication mechanism for End-Nodes with back-end infrastructure to provide payload integrity and confidentiality. Nevertheless, it does not cover gateways or their relation with End-Nodes or other elements of the back-end infrastructure [22].

Gateways are key elements of LoRaWAN network architecture because they convert LoRa packets into IP ones, they are essential to allow communication from End-Nodes to back-end servers (NS, AS and JS) and viceversa. Hence, gateways need to be securely recognized over the network so that End-Nodes and network servers receive commands and information from a registered party. This feature can not be implemented over the end-node

because it is already authenticated, but has to use a participant that is able to talk and understand the gateway (network server) [23].

The problem addressed in this work is to provide a secure and lightweight authentication protocol to register gateways within LoRaWAN network to handle secure round-trip communication to the nodes and network servers. Our proposed protocol not only authenticate gateways with network servers, but also with End-Nodes. This protocol handles secure authentication and is lightweight to prevent excesive power and computational consumption. Authenticating the gateway will prevent end-devices receiving forged Class B downlink messages that make them unavailable. Likewise, network servers will receive valid payloads from recognized gateways preventing fake uplink message injection or excesive forged uplink messages.

1.2 Objectives

The main objective of this work is to increase the security of LoRaWAN by using lightweight cryptography that enables gateway authentication to secure communication for IoT devices and network servers that use this protocol. First, we analyze and identify vulnerabilities in LoRaWAN to point out gateway authentication issues. Then, we design a security protocol that uses lightweight cryptography to address the lack of gateway authentication in LoRaWAN networks. Finally, to evaluate performance and security issues that might affect IoT devices where this protocol will be deployed, we made a proof of concept (PoC).

1.3 Research Methodology

We have chosen Design Science (*DS*) as the methodology to conduct our research. *DS* allows us to design and develop a new artifact (network security protocol for LoRaWAN) [24, 25]. This methodology can be applied to the context of our research as described in [26–30]. *DS* consists of six phases that let a researcher to guide the investigation as shown in figure 1.1
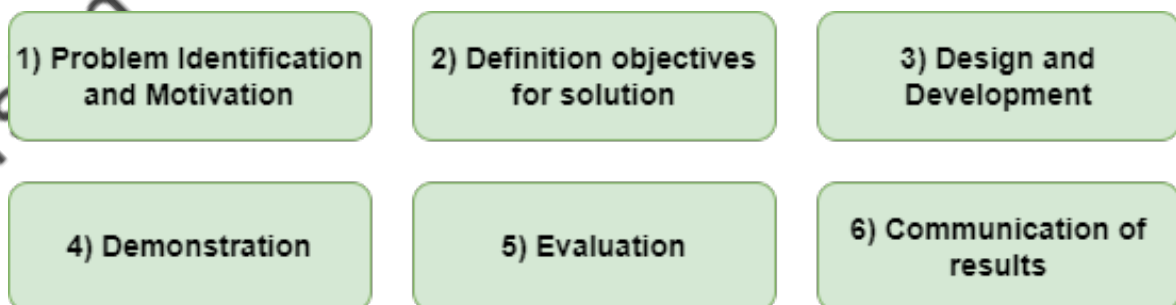
Figure 1.1: Design Science Resarch Stages

DS was chosen because during stages 1 and 2 it will guide us to identify the problem of

gateway authentication and define the objectives for our solution to address the identified problem. During *Design and Development* we will use this methodology to design protocols for addressing security issues related to gateway authentication. *Demonstration* stage will be used to implement the designed solution into a proof of concept. Then, during the *Evaluation* phase we evaluate the security of our solution and a performance analysis in terms of end-device power consumption and computing performance by analyzing RAM and CPU.

1.4 Research Contributions

The contributions produced throughout this research have been classified in two types:

Journals

- **Jhonattan J. Barriga A.** and Sang Guun Yoo. *Internet of Things: A Security Survey Review on Long Range Wide Area Network (LoRaWAN)*. **Journal of Engineering and Applied Sciences**, vol. 14, pp. 9774–9787, 09 2019.

- **Jhonattan J. Barriga A.**, J. Sulca, J.L.León, A. Ulloa, D. Portero, J. García, and Sang Guun Yoo. *Smart Parking: A Literature Review from the Technological Perspective*. **Applied Sciences**, vol. 9, no. 21, 2019.

- **Jhonattan J. Barriga A.**, J. Sulca, J.L.León, A. Ulloa, Diego Portero, J. García, and Sang Guun Yoo. *A Smart Parking Solution Architecture Based on LoRaWAN and Kubernetes* . **Applied Sciences**, vol. 10, no. 13, 2020.

- **Jhonattan J. Barriga A.** and Sang Guun Yoo. *Securing End-Node to Gateway Communication in LoRaWAN with a Lightweight Security Protocol* . **IEEE Access**, vol. 10, pp. 96672-96694, 2022.

Conferences

- **Jhonattan J. Barriga A.**. *Privacy-Aware Authentication for Wi-Fi Based Indoor Positioning Systems*. Sang Guun Yoo and In: Batten, L., Kim, D., Zhang, X., Li, G. (eds) **Applications and Techniques in Information Security. ATIS 2017.**, Auckland, New Zealand. Communications in Computer and Information Science, vol 719. Springer, Singapore.

- **Jhonattan J. Barriga A.**and Sang Guun Yoo. *Security over Smart Home Automation Systems: A Survey*. In: Rocha, Á., Guarda, T. (eds) Developments and Advances in Defense and Security. **MICRADS 2018**, Santa Elena, Ecuador. Smart Innovation, Systems and Technologies, vol 94. Springer, Cham.

- **Jhonattan J. Barriga A.**, Sang Guun Yoo and Juan Carlos Polo. *Enhancement to the Privacy-Aware Authentication for Wi-Fi Based Indoor Positioning Systems.* In: , et al. Applied Cryptography and Network Security Workshops. **ACNS 2019**, Bogota, Colombia. Lecture Notes in Computer Science, vol 11605. Springer, Cham.

1.5 Thesis Structure

This thesis is structured as follows. Section 2 describes LoRaWAN architecture, security and techonology features. Section 3 provides the details on the solution designed for gateway authentication as well as protocols to enhance security communications betweend end-device, gateway, and network server. Section 4 evaluates the proposed solution by analyzing performance implications over the end-device by measuing computational attributes (RAM, CPU) and power consumption. Finally, Section 5 presents the conclusions of this work.

Chapter 2

BACKGROUND

This chapter presents the backgorund related with LoRaWAN. Section 2.1 provides technical information in detail of LoRaWAN, while Section 2.2 presents the discussion about LoRaWAN vulnerabilities. At the end of this chapter we can denote the importance of gateway authentication.

2.1 LoRaWAN features and architecture

LoRaWAN has gone through a series of improvements, described in its specification, as shown in Figure 2.1. This thesis is focused on LoRaWAN version 1.1, since it has substantial improvements in security aspects compared to previous versions, which deserve a deep analysis to highlight shortcomings.
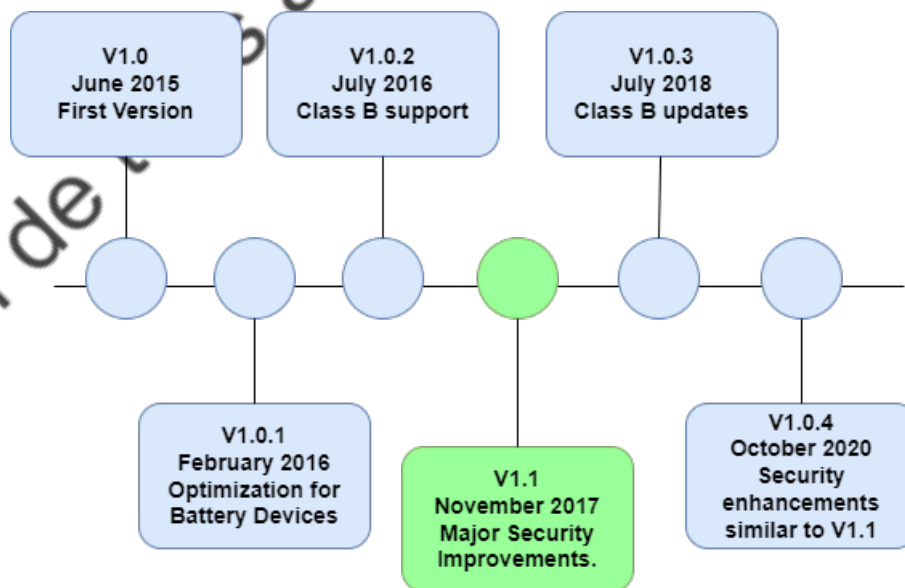
Figure 2.1: LoRaWAN versions timeline

2.1.1 Technical Overview

Long Range Wide Area Network (LoRaWAN) is a LPWAN technology that uses CSS and FSK modulation. The coverage range of this technology oscillates between 5km within urban areas and 20km for rural areas. In terms of bandwidth it supports 125Khz and 250Khz. A payload can handle up to 243 bytes. It implements mutual authentication with the use of two symmetric keys. For encryption it uses AES-128 in CTR mode and for integrity it uses Message Authentication Codes (MAC). Its infrastructure is completely open and allows private implementations given the chance that anyone could implement his own infrastructure by using open source tools like ChirpStack (`https://www.chirpstack.io/`) [3], [7–9], [31].

LoRaWAN operates at the MAC layer and it is based on LoRa. LoRa is the physical layer protocol based con Chirp Spread Spectrum which is similar to FSK modulation, but it provides a longer communication range [32]. LoRaWAN has gone through several improvements so that its specification has changed several times. The specification considered for this work is [22], since it has major changes in terms of session keys.

LoRaWAN works over unlicensed Regional Industrial Scientific Medical (ISM) bands. ISM bands are 868 MHz in Europe, 915 MHz in North America, and 433 MHz in Asia. LoRaWAN has three classes known as Class A, B and C as shown in Figure 2.2. Class A is not optional and has to be implemented by all End-Nodes. Devices that implement more than the mandatory class are considered High-End devices [22].
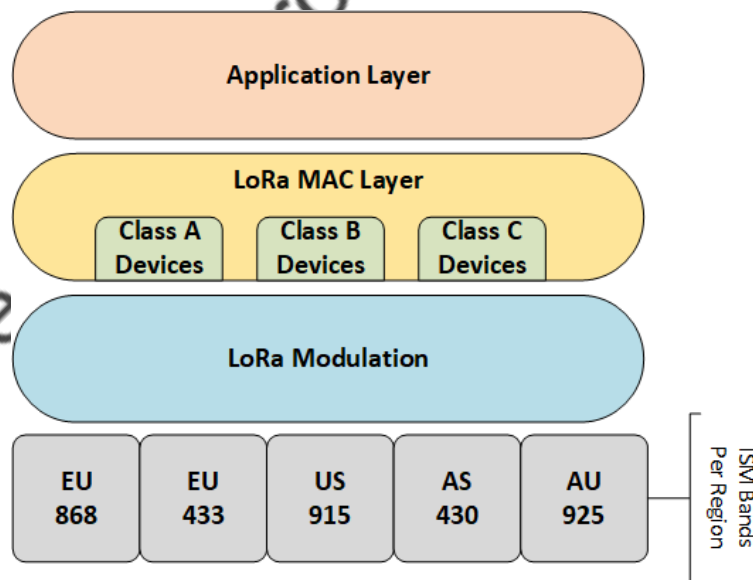
Figure 2.2: LoRaWAN Layers and Classes

There are three classes of devices according to LoRaWAN specification. First, class A devices are bi-directional end nodes which are more energy efficient and have two short defined reception windows after every uplink message. Class B devices open additional receive windows on scheduled times with the use of beacons sent by the gateway. Finally,

Class C devices are continuously listening and they are the least energy efficient but offer the lowest latency level [22].

Architecture

LoRaWAN is composed of three elements: End-Nodes, gateways and back-end servers. On the one hand, back-end servers are composed of: Network Server *(NS)*, Join Server *(JS)* and Application Server *(AS)*. Any end-node that wants to communicate with the back-end server infrastructure must go through a gateway (*Gw*). The communication between end-node (*EN*) and gateway is performed through LoRa protocol which is based on Chirp Spread Spectrum [32]. the gateway to back-end servers communication is handled over TCP/IP protocols [9], [12], [32]. The following Figure 2.3 shows the architecture of LoRaWAN with all its actors.
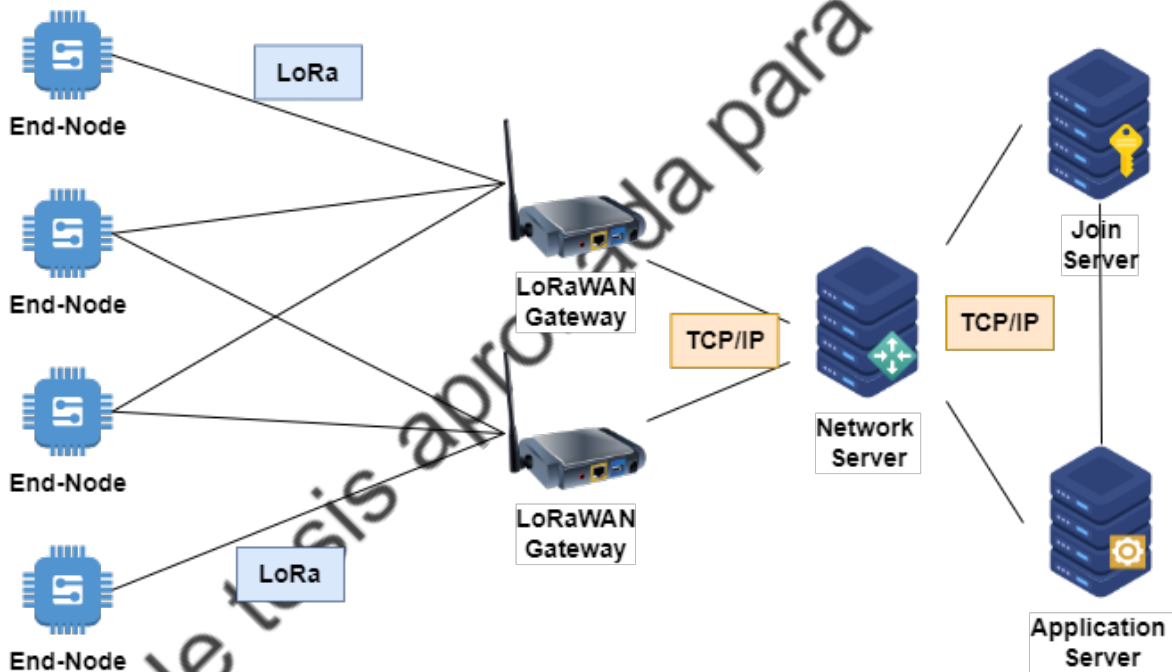
Figure 2.3: LoRaWAN Architecture

LoRaWAN Backend Infrastructure

As described in LoRaWAN Backend Interfaces Specification [23], besides radio gateway, there are three types of servers that are part of the backend architecture of LoRaWAN. Those servers are: Network Server, Application Server *(AS)*, and Join Server; each of them perform specific tasks within the whole architecture. The Network Server *(NS)* is in charge of handling LoRAWAN MAC layer for End-Nodes, forwarding messages to *AS*, forwarding Join messages to *JS*, frame authentication, end-node verification among others.

For Roaming scenario, LoRaWAN Backend Interfaces Specification [23] describes three roles for the NS which are home *(hNS), serving(sNS)* and *forwarding(fNS). hNS* is responsible for persisting information related to Service, Device and Routing profile, and *DeviceEUI*. This roles depends on *JS* for joining purposes and is connected to *AS*. In roaming scenario *sNS* and *hNS* are separated and uplink or downlink messages are passed from *sNS* to *hNS*. *hNS* is in charge of forwarding uplink messages to the proper *AS* based on *DevEUI* parameter. In addition, *sNS* role handles only MAC layer for the End-Node. Last, *fNS* handles gateways and there may exist more than one *fNS* serving a single End-Node.

According to [23] *JS* manages End-Device activation process through Over the Air Activation (OTAA). A single *JS* could be connected to multiple *NSs*. This server contains information concerning Join-Request frames (uplink) and Join-Accept frames (downlink). It shares derived sessions keys with *AS* and *NS*. A *JS* could be connected to several *AS*, also a single *AS* could be connected to multiple *JS*. The *AS* is in charge of handling payloads (uplink and downlink frames) sent by the End-Devices. *AS* may be connected to multiple *NSs* and *JSs*, and several *AS* may be connected to a single *NS*. According to [23], there are several interfaces in place to support several procedures within LoRaWAN network from the perspective of the End-Device (home or roaming). These interfaces are:

- **sNS - JS**: Used during Roaming Activation Procedure, it helps to obtain *NetID* from *hNS* of a particular *EN*.

- **hNS - JS**: Supports Join Procedure between *NS* and *JS*.

- **hNS - sNS**: Supports signaling whilst in roaming as well as payload delivery between *hNS and sNS*.

- **sNS - fNS**: Supports signaling whilst in roaming as well as payload delivery between *sNS* and *fNS*.

- **AS - hNS**: Supports payload delivery between *AS* and *hNS*.

- **AS - JS**: Supports delivery of Application Session Key *(AppSKey)*.

- **EN - NS**: Used to support LoRaWAN MAC-layer signalink and payload delivery between *EN* and *NS*.

LoRaWAN can be used to implement applications for any of the following scenarios: Smart Cities, Smart Waste, Smart Agriculture, Smart Health among others). Its level of openness facilitates performing customizations in most of its components. Also there are no connection fees associated, it uses a free spectrum band. Next, we will describe an example of this usage applied to Smart Parking.

LoRaWAN Applications

The continuos growth of IoT has allowed LPWAN networks like LoRaWAN to be considered in smart initiatives deployments. One of these smart trends is smart cities. It comprises several applications that are being deployed around the world as they contribute to reduce pollution and improve the life of citizens The most popular among smart cities is smart parking. It reduces congestion, has a pricing system that is adjusted based on the demand for availability on-peak hours, and reinforces traffic laws by using cameras to detect violators as described in [6].

LoRaWAN is useful for implementing smart parking solutions. According to [33] an smart parking architecture mainly consists of Sensing Infrastructure and Backend Infrastructure. In the Sensing side smart parking sensors are deployed while in backend there is a gateway, network server, join server and application server, as shown in Figure 2.4
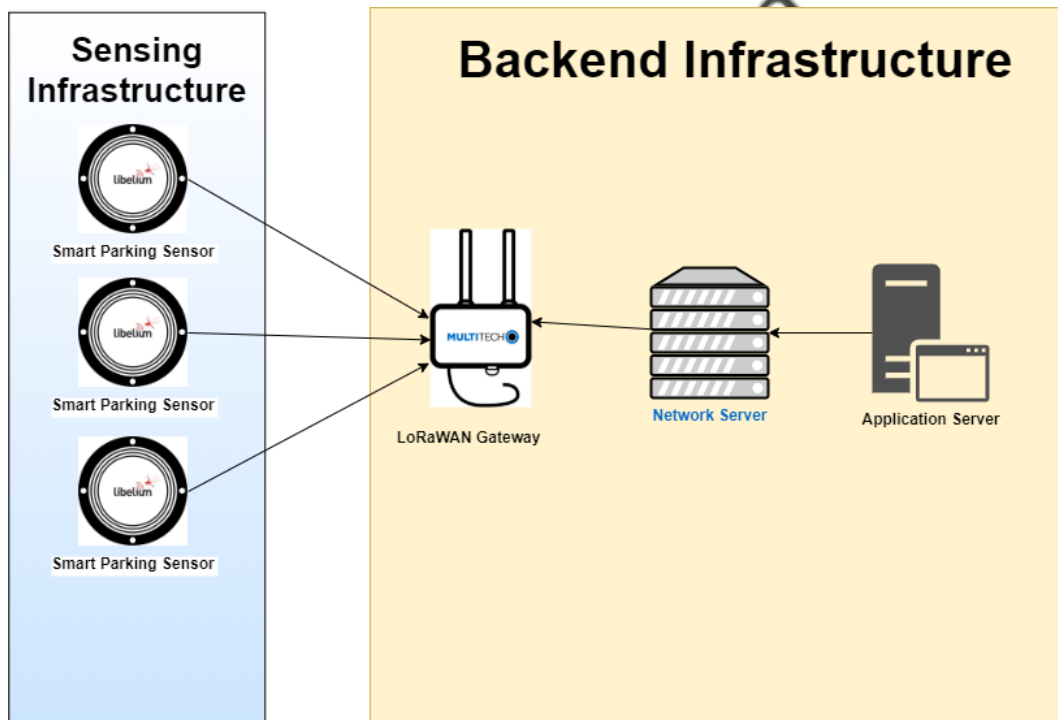
Figure 2.4: Smart Parking Architecture based on LoRaWAN

When building such type of solutions, it is important to have an activation mechanism. For such case we used OTAA to guarantee data integrity and provide confidentiality to information that is being transmitted. The architecture presented before acts as a template that can be customized according to the project needs. In a smart parking solution smart parking sensors are able to detect vehicle presence as well as other environmet details. These data is transmitted through the gateway to the network server and then to the application one. This last point can be later connected to another technology or infrastructure as described in [33] where an MQTT server was used to collect raw data produced by the End-Nodes
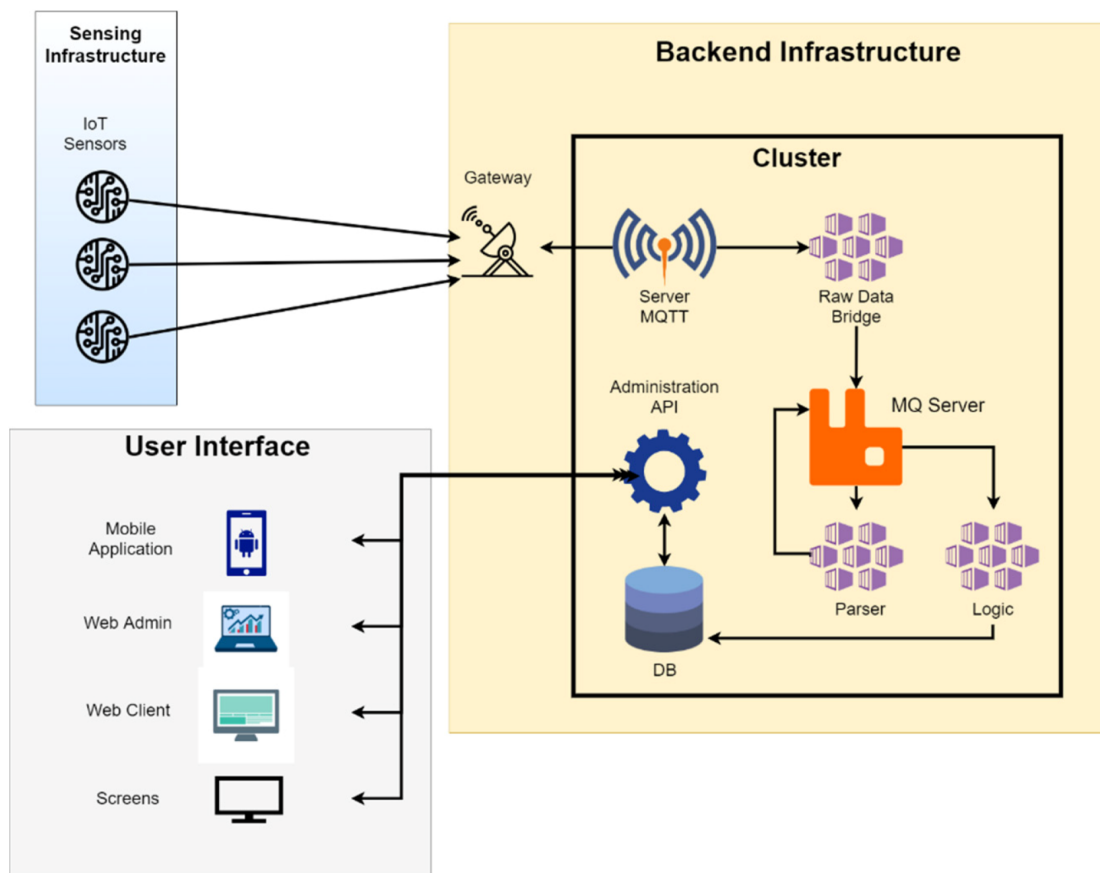
Figure 2.5: LoRaWAN Smart Parking Solution based on Kubernetes

(Smart Parking Sensors). In the solution proposed in [33] and shown in Figure 2.5 LoRaWAN architecture was connected to an Rabbit MQ Server to process payloads and provide insights that were published in a web application and a mobile application through an API REST. As shown in the work cited, LoRaWAN is versatile because it can be modified and adapts to various types of IoT-based applications.

2.1.2 Security in LoRaWAN

According to LoRaWAN 1.1 specification it uses some security mechanisms such as mutual authentication, symmetric cryptography and session key distribution. Mutual authentication is the process where End-Nodes and back-end servers derive keys to provide data integrity and confidentiality. This derivation process is part of the activation procedure to register and end-node over LoRaWAN infrastructure [34]. The procedure for activating and End-Node (EN) within the LoRaWAN infraestructure is known as *Join Procedure* or *Over The Air Activation (OTAA)*. This procedure is described in detail in the next section.

Join-Procedure and session keys derivation

LoRaWAN supports two activation processes (join procedures) for enabling End-Nodes over a LoRaWAN network. Those processes are: Activation By Personalization (ABP) and Over the Air Activation (OTAA).

The OTAA procedure is started by the end-node. For this purpose, each end-node has the following security parameters $DeviceEUI_{ENi}$, $JoinEUI_{ENi}$, $NwkKey_{EN_i}$ and $AppKey_{EN_i}$. The last two parameters are 128-bit keys used to derive session keys. These parameters are factory stored settings. This procedure is considered more secure than ABP since other keys are derived from known parameters stored in the device. In ABP mode, it is required that all sessions keys have to be preloaded in the end-node, application server, network server and join server for executing the join procedure and then sending uplink messages.

The Join-Procedure is a process for authenticating *End-Nodes* over a LoRaWAN network. This process is mandatory before sending any uplink message. In order to proceed the End-Node must first build a Join-Request message composed as follows by the *JoinEUI*, *DevEUI* and the *DevNonce*. *JoinEUI* is an identifier of the *JS*, *DevEUI* is a unique identifier of the Device and *DevNonce* is a sequential 2-byte number generated by the *EN*. This message is sent in plain-text. These parameters are evaluated by the *JS* and *NS* as follows. *JS* verifies that *DevEUI* is in the authorized list whilst *NS* validates and keeps a track of every *DevNonce* generated. If the procedure is successful, the Network Server will respond with a Join-Accept message to the *EN* so that it could derive session keys (Application Session, Network Session and Join Session keys) [12], [22].

The Join-Accept message contains the following parameters *JoinNonce, Home_NetID, DevAddr DLSettings, RxDelay and CFList* [22]. The following table describes the parameters that are part of the Join-Accept message see Table 2.1.

Once the *EN* receives the Join-Accept message, the following session keys are derived according to the specification [22]. Every session key is used for a particular purpose.**FNwkSIntKey** and **SNwkSIntKey** are used to calculate MIC fields for preserving message integrity.**NwkSEncKey** is used to cypher messages for NS.**AppSKey** is used to cypher FrmPayload for **AS** [22]. The session keys are derived as follow according to the specification:

$FNwkSIntKey=SEnc(NwkKey,0x01||JoinNonce||JoinEUI||DevNonce||pad16)$

$SNwkSIntKey=SEnc(NwkKey,0x03||JoinNonce||JoinEUI||DevNonce||pad16)$

$NwkSEncKey=SEnc(NwkKey,0x04||JoinNonce||JoinEUI||DevNonce||pad16)$

$AppSKey=SEnc(AppKey_{EN_i},0x02||JoinNonce||JoinEUI||DevNonce||pad16)$

$JSIntKey=SEnc(NwkKey_{EN_i},0x06||DevEUI_{ENi}||pad16)$

After the Join-Accept, *JS* must record and keeps a track of every JoinNonce generated every time a Join or a Rejoin is performed. The Join Procedure is summarized in the following

Table 2.1: Join-accept parameter summary

Parameter	Size(bytes)	Generator	Purpose
JoinNonce	*3*	*JS*	Counter value incremented with every Join-Accept
NetID	*3*	*NS*	Network Identifier
DevAddr	*4*	*NS*	Device Address
DLSettings	*1*	*NS*	Downlink Configuration
RxDelay	*1*	*NS*	Delay between reception and transmission
CFList	*16*	*NS*	Optional list of network parameters
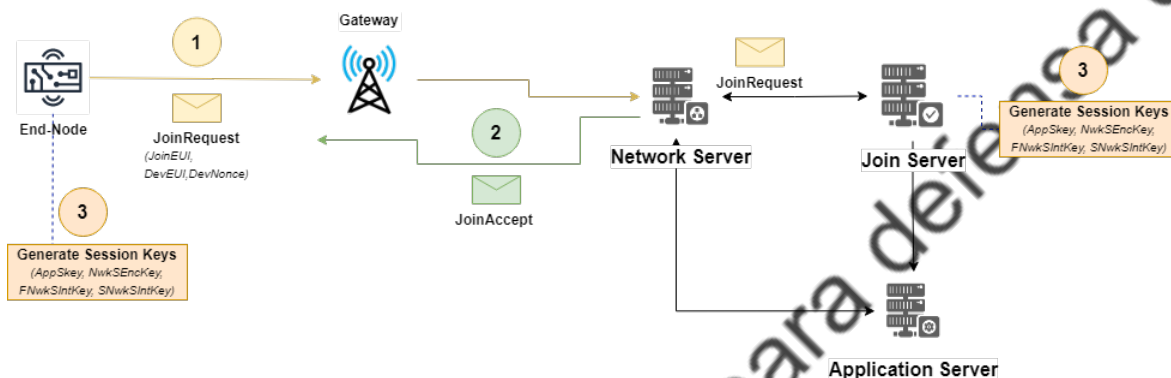
Figure 2.6: LoRaWAN 1.1 Join Procedure

Figure 2.6.

ABP procedure requires the manual input of session keys listed before. This procedure does use the same session keys for all their lifetime. It is then, more insecure than OTAA. There is no join-procedure or session key derivation and if keys are required to be renewed, they need to be manually configured.

Although there are some security considerations and activation processes described in the specification, there are still some issues that need to be addressed as they may compromise integrity and confidentiality of data and actors. These issues are described next.

2.2 LoRaWAN Vulnerabilities and Proposed Mitigation Mechanisms

There are several works discussing LoRaWAN vulnerabilities and possible countermeasures. One of them found and exploited five potential vulnerabilities for specification 1.0.2 [16]. The identified vulnerabilities were: i) Replay attack for ABP-activated nodes, an attacker might use older messages and resend during the current session leading to a lost communication between node and network server, ii) Eavesdropping, as the key is reused on every cipher text, an attacker might be able to decrypt information based on previous messages sent with the same key, iii) Bit-Flipping Attack, although information is encrypted, it could be messed up. In this case, through a MiTM between TCP connections, an attacker might modify certain bytes of the encrypted payload, iv) ACK Spoofing, previously captured ACK could be delayed

13

to selectively acknowledge the successful receipt of another distinct message, even if it has not arrived yet to the backend provider, v) LoRa class B attacks, a malicious user might attempt to drain the battery of the device by modifying beacon payloads as they are not encrypted. The aforementioned vulnerabilities show weaknesses over LoRaWAN that affect integrity, availability and confidentiality. Probably, the most critical is the one that affects integrity in spite of having a secure symmetric algorithm in place (AES-128), attackers are able to modify information. Anyhow, the vulnerabilities described before have apparently been solved in LoRaWAN according to an analysis performed by [18].

Kim et al propose a dual key based schema for a secure authentication in LoRaWAN 1.0.2 [14]. The authors analyze security problems during end-node activation over OTAA. In this approach, they propose the inclusion of a new key called NwkKey. It has to be stored in the end-node and must not be shared with others. Such key is pre-shared only with NS, and will be used to to generate a new session key called as (NwkSKey). On the other hand, the preexisting AppKey is pre-shared only with AS. With this approach, the OTAA procedure changes a bit as described by the authors. The authors have proposed several previous steps that are performed before establishing a communication. First of all, the initial Join Request is generated and sent to the NS with the NwkKey instead of the AppKey. Then, as the NS knows the NwkKey, it will calculate the MIC again to validate the Join Request. If such validation is successful the NwkSKey will be generated. Later, AS generates the AppSKey based on the AppKey previously shared. Then, the AS sends the AppNonce to the NS. Furthermore, the NS sends the Join Accept message which contains NwkNonce and AppNonce encrypted with NwkKey and AppKey respectively. Finally, the end-node decrypts the Join Accept message and generates NwkSkey and AppSKey which are going to be used instead of the previous pre-shared keys to prevent a key-leakage. The inclusion of this new key aims to delegate authentication to every server in the architecture. This proposal has1 been remarkable, as this new key was included in the new specification of LoRaWAN 1.1. Also, the inclusion of this new key provided backward compatibility scenarios when dealing with devices working on versions 1.0.2 and 1.1 as described in LoRaWAN 1.1 specification [22].

Several vulnerabilities have been identified in LoRaWAN as part of an analysis of the LoRaWAN 1.1 specification [17]. In this review, the authors highlight six attacks. First, the authors indicate that RF jamming attack affects gateway and end-node letting attackers to jam such traffic. Moreover, they explain that replay attacks affect the join procedure where an attacker can jam signals for the OTAA session; an attacker might use previous join-request to connect to the Network Server. Besides, the aforementioned work indicated that Beacon Synchronization attack might let malicious users to set up gateways sending fake beacons. In addition, network traffic analysis would be exploited by an attacker that sets a rogue gateway to capture information. Additionally, it also warns that by executing a man-in-the-middle attacks against servers, might compromise unencrypted communication

between network server and application server which means that this vulnerability is still present in the new specification of the protocol. Finally, the authors use a tool called Scyther to assess cryptography over LoRaWAN where they indeed have concluded that it is a strength point of the protocol. From the vulnerabilities described before, the first three lead to a DoS attack, and the rest are related to traffic analysis and MiTM attacks [17].

Replay attacks have been addressed and discussed in depth for LoRaWAN version 1.0.2 as in [35] Kim & Song, 2017a; Na, Hwang, Shin, & Kim, 2017; [36] [37]. One of the proposed solutions aims to identify moving nodes and try to differentiate malicious nodes from valid nodes. This solution is oriented to support join procedure over the network server. The authors address this attack by using Received Signal Strength Indicator (RSSI), which are used to measure the strength of signal, and Propietary Hand-Shaking. In this scenario, the authors propose that the RSSI should be stored along with the DevNonce to compare it with future requests and validate that are not repeated. Validating the RSSI just by itself is not enough and therefore the server must send a proprietary message (MType set to 111) instead of join-accept for the join request. The end-node device responds to the proprietary message of network server with the same MType. This proprietary message is encrypted so that it could not be modified by an attacker. Anyhow, authors recommend that it is necessary to validate users through RSSI rather than generating proprietary messages for every join requests [36].Another proposal is described in [14] where the authors work on LoRaWAN version 1.0.2 to build a mitigation scenario. The proposed scheme redefines the initial and non-initial join request. Non-initial join request is used in standard conditions and helps to confirm the validity of NwkSKey to prevent replay attacks. Whilst, initial join request is used when the NwkSKey does not exist in the node, and prevents replay attack using the DevNonce. This initial join request, is valid no matter if the NwkSKey is lost on the end node [14].

Join Request in over the air activation (OTAA) was the scenario that SeungJae Na et al used to identify a replay attack and later propose a solution. The authors manifest that join requests are sent in plain text; therefore, an attacker might sniff join-requests from other devices and then try to resend them to the network server. If it responds to those old join-requests any valid node will not be able to communicate. To address this problem, they suggest to use a token that could be XORed with the current Join Request. This token indeed, is a previous NwkSKey which will allow to mask the current request (Na et al., 2017).

Another work is the development of the Security Analysis of LoRaWAN Join Procedure for Internet of Things Networks [37]. The authors have identified that it is not necessary for an attacker to be present in order to compromise the network, as there is a chance that an end-device could generate a previous DevNonce. They perform experimentation over a SX1272 modem by using a jammer and measured the entropy level. They concluded that the proximity reduces entropy level of the end-device, affecting its ability to produce random

information [37].

In [35] an analysis of vulnerabilities in low-power wide-area networks is performed over LoRaWAN 1.0.2. This work depicts two main vulnerabilities. First, replay attacks during any type of end-node activation mechanism (ABP or OTAA) during join procedure. Under this scenario an attacker could send previous join-accept or join-request to disable an end-devices. The second vulnerability identified was ACK spoofing. In this case, an attacker could intercept and resend the same ACK message to confirm various messages from the end device. For this scenario, the attacker must have compromised the gateway before. In this work, authors have not performed an experimental procedure, but support their findings on reviewing and analyzing the LoRaWAN specification [35].

In 2016, a secure architecture was proposed [38]. The approach was to include a certification authority (CA) at the gateway level to use it for authentication handling, authorization and key management of nodes. This solution contemplates the use of tables to identify certificates of nodes, a trust table to identify the level of trust of a particular gateway, and a black list of nodes. A strength of this solution is that it considers a de-authentication phase and the possibility of changing the CA. In this scenario, a MiTM is not possible as messages are cyphered with public keys.

Aras Emekcan et al, in their work, identified vulnerabilities in LoRaWAN 1.0.2 specification. The vulnerabilities found have to do with compromising devices and network keys, assuming that a malicious attacker might have physically accessed the device and was able to extract root keys by using Xignal mousetrap. Also, jamming techniques were possible by using cheap devices to flood LoRa channel causing communication unavailability [39].Besides, replay attacks over ABP join procedures, and wormhole attacks are possible when an attacker captures packets from one device and resends them to a distant device to replay a hijacked packet [40]. To prevent jamming attacks, an initiative based on a Network Intrusion Detection system is discussed in (Danish et al., 2018); this approach is based on analysis of two algorithms i.e. Kullback Leibler Divergence (KLD) and Hamming distance (HD). At the end, the authors found out that KLD performs better and it would be useful to detect jammers within the radio of a LoRaWAN end node.

Although LoRa Alliance has improved LoRaWAN specification in order to address previous vulnerabilities, there are still be security issues; and this situation is discussed by [18]. In regards of join procedure, the author identifies four main threats i.e. key management, join procedure delegation, backward compatibility, and replay protection in join procedure. LoRaWAN 1.1 specification includes a handover roaming which opens the threat for a man-in-the-middle attack (MiTM) since FRMPayloads are first transported from the serving network server (sNS) to the home network server (hNS) and then to the application server (AS). LoRaWAN specification does not reinforce physical tampering protection. Indeed, it increases the possibility of compromising root keys as a malicious user with physical access

to device might steal them to decrypt information. The mechanism proposed in this work would address this issue, as root keys will only be used to perform an initial joint procedure and then deleted from the node. In the eventual scenario of an attacker compromising the node, it will not be possible to discover previous session keys. Hence, to properly address key leakages, it is mandatory that the NS records previous AppSKey [14]. To support backward compatibility, a new root key (NwkKey) is used to handle join request delegation. But if there is no proper forward secrecy, when the new root key is taken by an attacker, data from end-node might be exposed and decrypted. This work contributes with two alternatives to address the threat; first the author indicates that the node should be configured to always derive AppSKey based on AppKey no matter the value of OptNeg He also explains that it is necessary to set the version of the protocol in the node. The author also indicates that a malicious NS might be able to replay OTAA activation messages on the join server compromising integrity and confidentiality of application data as there might be an overflow of counter nonce. For this scenario, the author proposes that the Join Server (JS) must record previous DevNonce and Jcount0 last values, giving the possibility that JS would be able to identify a possible overflow of JoinNonce and stop receiving requests from a particular node. These suggestions are based on protocol review rather than experimentation.

Denial of Service (DoS) also affects LoRaWAN 1.1 as described and tested in (Eef van Es, Vranken, & Hommersom, 2018). The authors used Coloured Petri Nets (CPNs) and with the help of CPN-Tools, they simulated and analyzed the following vulnerabilities that could result in a DoS attack. First, beaconing is a vulnerability that has to do with Class-B beacons broadcasted by gateways to schedule reception windows. These beacons are not encrypted or signed. In these circumstances, an attacker might be able to modify timing references, producing a desynchronization of the window reception, leaving the node unable to communicate with the network server. According to the author and supported by (Miller, 2016) this vulnerability is still present in LoRaWAN 1.1. Another vulnerability is downlink routing. First of all, anytime the network wants to start a downlink traffic to a node, it has to rely on the gateway that is aware of a previous uplink transmission. During the uplink procedure, nodes broadcast their data to the nearest gateways and then forward packets to the NS (which is able to track previous gateways used by a particular node). However, an attacker can eavesdrop uplink transmission and respond through a gateway that is out-of-reach of the end-node, causing that a valid gateway near from a node would be discarded as it has been unauthenticated by the malicious gateway. This vulnerability was discovered in version 1.0 and according to the authors it is still present in the new specification. The last vulnerability found in this work is known as join-accept replay vulnerability. In ABP, network server provides configuration settings and those have to be inserted manually in the end-node. Meanwhile, OTAA is more pliable as with few configurations in the node, it will be able to exchange information with the NS and synchronize configuration details. The weakness is that there is no reference in the except for a previous request. According to (Eef van Es et al., 2018),

this vulnerability has been addressed for version 1.1, but since there is a compatibility for devices running version 1.0.2, the vulnerability is still present for such devices in the new specification. This work is the latest in regards of vulnerabilities associated to LoRaWAN 1.1.

A report from the industry written by R. Miller from MWR Labs (Miller, 2016), identifies seven vulnerabilities in LoRaWAN 1.0 solutions. First of all, weaknesses in key management as AppKey is stored in two places (End-Nodes and servers). In regards of key usage if the message payload is analyzed before considering the MIC field, an attacker might bit-flip the information to modify its content. Second, weaknesses in key generation if keys used over ABP are generated over a simple procedure such as using Device Address, an attacker might reverse it to find out the way to compromise all other nodes. Third, devices are supposed to be trusted; however, the information generated by them could have been mangled. For instance, it might include bogus characters that would allow an attacker to execute a SQL Injection attack. Moreover, the gateway could be compromised as it has Internet connectivity and if it has not been properly hardened, it could be compromised. This vulnerability is present over all devices with an IP connection to the Internet, in such case a VPN would help to reduce the risk of exposure. In addition, an invalid counter control would lead an attacker to devise a payload; therefore, a proper control should be in place to reduce the risk. Finally, in terms Beacons and Multicast messages there is no way for the node to know that they were generated by an authentic entity of the system. This work shows a brief enumeration of the vulnerabilities present and described in other works, but do not make a further analysis of them and suggest brief recommendations for preventing such attacks. Vulnerabilities described in this work have also been listed and discussed in (Na et al., 2017).

One of the most common issues in regards of LoRaWAN v1.0.2 is the key management process as revised in [41]. The authors denote that a major concern of this protocol is key sharing. During the OTAA procedure the shared AppKey is used to derive NwkSKey and AppSkey. This procedure is perfomed the first time a device connects to the network. However, during ABP both keys (NwkSKey and AppSkey) must be written in the device, making this a notable insecure procedure as those keys are never updated. Considering the ABP scenario those static session keys have a great rate of being compromised due to their lack of dynamicity. An attacker obtaining those keys would be able to decrypt all the information being sent. On the other hand, in LoRaWAN 1.1 some security measures have been considered to improve the generation of session keys; however, the procedure is still insecure as the generation of new session keys is based on two non-removable keys (AppKey and NwkKey) that have been previously loaded inside the end-device. To mitigate this vulnerability, authors propose the inclusion of EDHOC which is a lightweight key exchange protocol for establishing symmetric keys among two end-node devices. To integrate this solution to LoRaWAN, authors have designed three messages to securely update session keys. This protocol will be acting between the end-node and the network server according

to the authors. EDHOC acts on top of other protocols and will be able to derive NwkSKey and AppSKey previously obtained through OTAA. It is important to consider that before any communication an EDHOC negotiation must be executed to derive the previous mentioned session keys. Authors have shown that this solution will not use more than 176 bytes which can be supported by the greatest data-rate of LoRa configurations known as SF7 and SF8. The solution discussed before has been tested over LoRaWAN v1.0.2 and it might need to be modified to be supported by LoRaWAN version 1.1.

Five types of attacks have been identified, discussed and analyzed in [42]. The work was merely focused on LoRaWAN version 1.0. It shows weak points of the protocol in regards of decrypt attacks and DoS attacks. First of all, a device might be forced to reuse previous session keys making it possible that a frame of a past session would become valid and hence replayed to the Network Server (NS). To execute this attack, a device must use a repeated DevAddr, DevNonce and AppNonce. The authors support their statement on the birthday paradox and hence if an attacker is able to compromise the AppNonce (3-bytes length), the randomness of the session will only depend on DevNonce (2-byte length) parameter. Likewise, the specification does not clearly state how to handle unconfirmed Join Requests. In this scenario, a Join Request might contain a previously generated DevNonce. A replay attack is possible with the previous scenario because frames from previous sessions might be replayed in a new session. A message decryption is feasible through the described scenario as the device uses the same keystream to protect different frames. According to the authors a similar approach can be directed to target the NS making it to use the same DevAddr, DevNonce and AppNonce without performing any validation as the specification states that a record of some DevNonce must be stored. Therefore, if the attacker is able to force the server to generate the same security parameters, a replay attack might be executed. This attack is possible as the AppNonce is not long enough and it is pseudo-generated. On the other hand, Denial of Service attacks could affect end-node devices and NS. The main objective of this attack is to disconnect the device form the network by forcing it not to share its new session keys with the NS. The lack of confirmation of a Join Accept with a corresponding Join Request makes the device to be out of the network. Similarly, if the NS completes the key exchange procedure with the device all messages sent to the device will be ignored. AppNonces are generated every time a Join Request is received but if an attacker is able to replay to the NS with a previous Join Request the end-node device will not share the same session keys. The authors point out that a lack of integrity between the NS and the Application Server (AS) is present, as messages traveling between them are encrypted only but lack of a mechanism to calculate their integrity. Likewise, data integrity might be compromised as data encryption is performed in counter mode only and hence a bit flipping attack over the ciphertext could be executed. Finally, some recommendations are listed by the authors. First of all, to mitigate decrypt or replay attacks, it is important to validate that either AppNonce or DevNonce have been previously generated or to add counter instead of

a random procedure; however, this last recommendation might allow an attacker to guess the next parameter to be used. Increasing the size of the fields that carry such information would help to avoid repetition. Second, to prevent DoS attacks it is recommended to associate a particular Join-Request with a Join-Accept message. Finally, confirming keys would allow to prevent a replay or decrypt attack anyhow, those session keys might be generated twice and hence such countermeasure would not be enough so it must combine with the previous mentioned recommendations. This work depicts vulnerabilities of LoRaWAN v1.0. Most of the analysis performed is based entirely over the specification but authors have proved mathematically or practically (through formulas) that most of the attacks are real and might be exploited by malicious users. Countermeasures proposed by the authors are valid but might demand additional hardware resources over considering that end-node devices have a very limited computational capacity.

Key management and replay attacks are also identified as potential vulnerabilities in LoRaWAN and hence a solution is proposed in [43]. The authors aim to add additional security measures to the current steps of LoRaWAN 1.0 to increase its level of security by generating an authentication procedure between the end-node device and the Application Server (AS). The authors propose the inclusion of a 3-step authentication confirmation between the end-node and the AS after the AppSKey has been generated. Also, there is a novel proposal for the DevNonce generation to obtain fresh Join-Request messages and prevent replay attacks. This approach claims to be secure as demonstrated by the authors using Ban Logic and AVISPA Tool. Also, it appears to be a low computational cost solution according to the tests performed. This solution has been implemented over a Smart Parking case study. Finally, although the work described is based on version 1.0, it could be applied to version 1.1 as LoRaWAN still lacks of a proper end-to-end security that delivers privacy; however, the authors state that performance testing should be performed to validate its feasibility. This work might be applied to the new version of LoRaWAN, but it is important to consider that there are power and computing restrictions for its implementation.

A major concern for improving LoRaWAN security is battery consumption. For that reason, there are scientific works for increasing security in LoRaWAN that aim to reduce battery consumption (a major premise of IoT) as proposed in [43]. The authors consider this fact to optimize AES-128 and come up with a solution called Secure Low Power Communication (SeLPC) which is based on D-Box that are renewed after a certain period of time and adding it to the AES process to improve its performance without scarifying security. This approach is based on two major phases: key generation and data encryption process. The purpose of the first stage is to update the AppSKey and the S-Box of AES after a period of n days (which could be configured by the network administrator). To achieve such goal, Enhanced version of DASS algorithm (used to handle S-Box) and D-Box generation are used. Finally, D-Box and AppSkey are updated every n days according to the definition adopted. Second, to perform

20

data encryption authors suggest that lieu to the fact that AppSKey and D-Box are updated frequently it will only be required to perform 5 rounds of AES-128 to encrypt the message to be sent. This approach is resistant to known-key attack as malicious user might have obtained a previous AppSKey, but he still needs the D-Box and a newly updated AppSKey. Also, replay-attacks are not possible because the Reply message is based on AppSKey and time parameter. Finally, although an attacker could be able to sniff over the channel and obtain an older AppSKey he could not be able to decrypt the message as AppSKey is based on a time variable. This work focus on secure one key, but as mentioned by the authors it needs to be extended to secure NwkSKey and MIC-code generation. In terms of performance, authors show that it might save around 26% of power consumption. Even though this approach is not specified for a particular version of LoRaWAN, it could be applied over the current version.

J. Kim and J. Song discuss a novel approach to securely share cryptographic keys for protecting Device to Device (D2D) communication. Authors mentioned that their work guarantees mutual authentication, integrity and confidentiality [44]. This work aims to support authentication between two devices interacting within a LoRaWAN system. The scheme proposed will generate two new messages (SecureD2DReq and SecureD2DAns) and will use NwkSKeys generated by the Network server together with a Device Nonce to produce encryption keys and integrity keys. Although this work does not directly address a vulnerability or a particular attack, it proposes a schema for increasing data rates. However, this approach needs to be tested over the new protocol specification.

The work described in [20] examines LoRaWAN 1.1 vulnerabilities. The authors use a tool called Scyther which is a formal tool to verify the security of protocols [45]. In this work, authors prove in a formal way that there are issues in regards of key exchange in version 1.0; however, according to the authors, they have not found weaknesses in version 1.1 of LoRaWAN specification. According to the findings, the OTAA procedure of version 1.0 lacks of proper association between the end-device and the NS or AS. This is produced because join-request and join-accept are not properly acknowledged. On the other hand, in terms of version 1.1 authors state that there might be security flaws in the AES algorithm using ECB for encrypting join-accept messages. Man-in-the-middle-attack (MITM) are possible through the bit-flipping attack even in the new version of LoRaWAN as discussed by the authors. The problem is that messages are encrypted between the end-node and the servers, but there is no mechanism to guarantee that messages sent between NS and AS are trustable in terms of integrity. According to the specification, NS and AS have to trust each other but there is no procedure described to achieve it. One of the most critical issues that has been denoted by the authors is root key-preloading, it poses a threat as those keys could be extracted by a malicious user and hence compromise the confidentiality of the information exchanged. Roaming capabilities of LoRaWAN may end up in MITM attacks as handover-roaming is

present. Also, this feature might result in a fallback since handover-roaming is not present in the previous versions of LoRaWAN. Anyhow, the discussed vulnerabilities have not been tested over a real implementation to confirm if such weaknesses might represent a real risk to the protocol.

In spite of using a secure encryption algorithm like AES-128. LoRaWAN is susceptible to bit-flipping attacks. In this attack, a malicious user would be able to forge encrypted payloads to produce fake information. Anyhow, authors in [13] have proposed a shuffling method to prevent attackers exploiting such vulnerability. The procedure consists in two phases: i) Shift phase where all the bytes perform a circular shift to the left and ii) Swap phase where the end-device swaps positions of the previously shifted bytes. Although the approach seems to be secure it might generate some performance issues. Also, it is not clear what would happen with repeated messages and hence they should have a different way of mixing bytes every time. Besides, an integrity check must be in place to prevent data corruption.

The authors in [19] described an approach for protecting gateway from activation by personalization (ABP) replay attacks. In their prososal, authors suggest a flow of events to detect if a replay attack is taking place, they use a database to create a list of devices with open data from LoRaWAN packets to store Device Address(DevAddr), Check Sum (CSum) and Frame Counter (FCnt). With previous data, they create a signature to identify if a message is repeated. They have created an algorithm that validates if a device has been reseted, a FCnt or a CSum are duplicated.

In terms of gateway authentication, the following works have been identified [15, 21]. In [15], the authors discuss the importance of gateway authentication and propose security protocols to authenticate gateways and mutually authenticate devices and gateways. The author proposes the inclusion of a third-party (Access Server) that will handle preloaded shared keys. The proposed protocol starts with the generation of a random nonce which is later replied by the end-node. These data is later processed by the access server which acknowledges a random nonce generated by the end-node and then the gateway hashes its random nonce previously generated together with the one generated by the end-node. Although this approach proposes a lightweight authentication mechanism, it relies on a third-party; however, this third-party could be replaced by the network server (NS) in a LoRaWAN network.

In the second approach [21], authors also rely on a third-party (a Certificate Authority) to authenticate the gateway over LoRaWAN network. They specify a format of the certificate to be recognized by the CA; however, authors focus on the relationship Gateway-LoRaWAN backend infraestructure. This approach uses Join-Server (JS) as a mean of communication between network server and CA Third-party. Finally an offline root CA is in charge of authenticating requests and to provide acknowledgment to the gateway and the network-server. Even though this approach provides authentication it requires a third-party to be part of the

infraestructure and it does not provide a mean to enhance the security of the relation between end-node and gateway.

The following table 2.2, shows a summary of vulnerabilities found in LoRaWAN versions. From previous table it can be denoted that gateways are devices prone to attacks. However, from the literature reviewed there a few works that point out authentication as mean of contention. Hence, our work will address this issue as gateways play an important role within LoRaWAN infraestructure. They transform LoRa packets into TCP/IP packets making End-Nodes available to deliver data to backend infrastructure.

Table 2.2: Table II LoRaWAN research summary in terms of vulnerabilities

Reference	Research Focus	LoRaWAN Version
[21]	Gateway Attacks (Authentication)*	1.1
[15]	Gateway Attacks (Authentication)*	LPWan
[12]	Gateway Attacks	1.1
[46]	Gateway Attacks	1.0 and 1.1
[16, 47, 48]	Gateway Attacks	1.0
[17]	Security Analysis	1.1
[40]	Security analysis	1.0
[14, 41, 49–51]	Key management improvements	1.0
[52]	Key management distribution improvements with blockchains	1.1
[53]	Root key protection in JS	1.1
[20]	Formal security verification	1.0 and 1.1
[54]	Join Procedure backward compatibility	1.0
[55]	Replay attacks	1.0
[56]	Replay attacks	1.0
[57]	Join Procedure with blockchains	1.1
[58]	Jamming attacks	1.0

The following figure 2.7 depicts the types of attacks that might occur due to lack of gateway authentication. A malicious user would be able to exploit three scenarios. First, considering that LoRaWAN traffic is wireless and therefore easy to sniff, he could decode some information of the payload as described in table 2.3. With such parameters decoded, a malicious user would try to craft, forge or create fake or valid uplink or downlinkg packets that might be injected to the network server or to the end-node. This lack of authentication is to be solved by using lightweight cryptography and without including third-party technologies to the current LoRaWAN specification. The design of the proposed solution is described and analyzed in the next chapter.

Table 2.3: LoRaWAN parameters obtained through decoding

Parameter	Obtained by decoding
MHDR	Yes
MACPayload	Yes
MIC	Yes
FHDR	Yes
FPort	Yes
FRMPayload	No
DevAddr	Yes
FCtrl	Yes
FCnt	Yes
FOpts	No
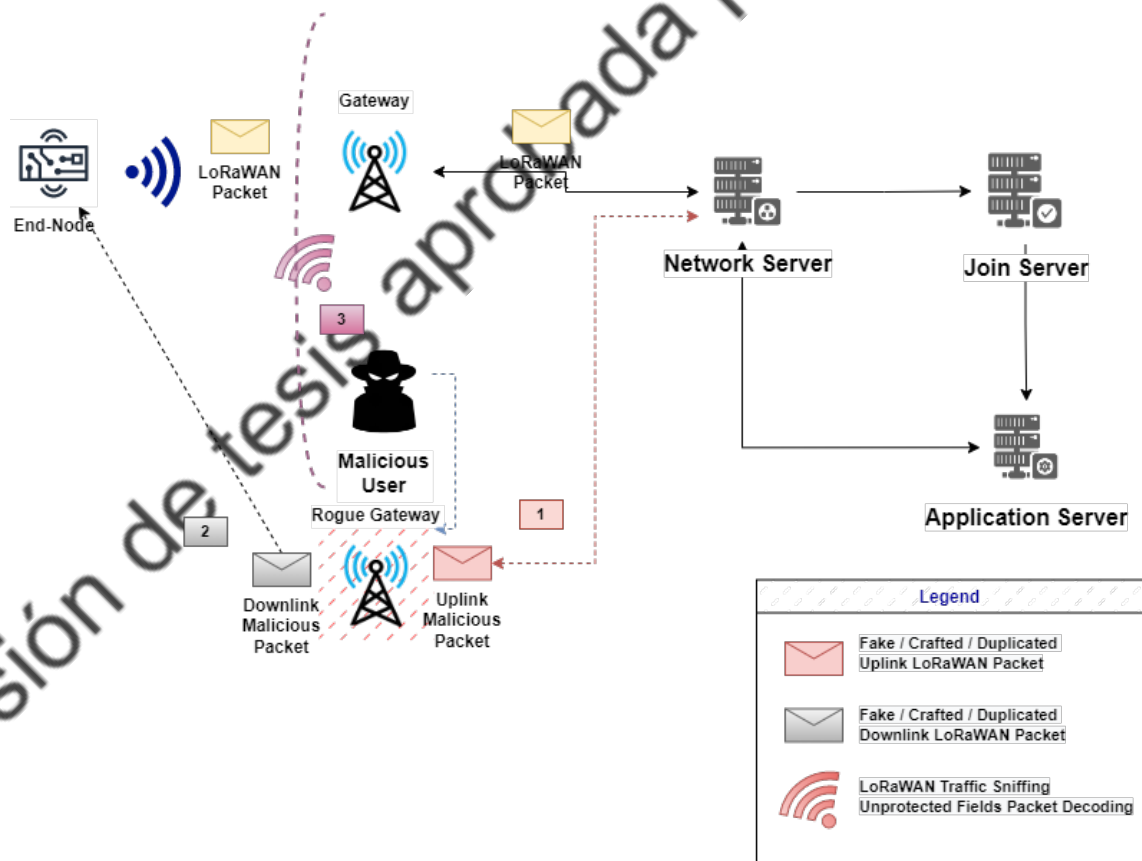
Figure 2.7: Attacks due to lack of gateway Authentication i) Uplink Packet Injection ii) Downlink Packet Injection iii)LoRaWAN pacekt sniffing and decoding

Chapter 3

SOLUTION DESIGN

Based on Table 2.2 the lack of authentication has been identified as an issue that has not been addressed yet. With this input, in this chapter we describe the solution proposed, which consists of three protocols:

- Section 3.1 describes the Gateway Registrarion Protocol which authenticates the gateway through the Network Server (*fNS* in roaming scenario) by generating a new key ($GrpKey_{GrpId}$) to authenticate it over the LoRaWAN infrastructure.

- Section 3.2 presents Gateway Session Key Derivation Protocol, which is designed from two scenarios home and roaming. It produces a new session key that will be used between the *EN* and the *Gw*. This key will be known as *GwSKey* and it will be generated during the Join Procedure. It will be shared to the *NS* and later to the *Gw* or group of *Gw* tied to a particular *NS*.

- Section 3.3 contemplates protocols for sending uplink messages messages over authenticated End-Nodes and gateways (UMOAEG) and over unauthenticated End-Nodes and gateways (UMOUEG) by using *GwSKey* to enhance the level of security when delivering uplink messages.

To design the protocols mentioned above, we based our proposal on previous works like [59–64] because they use lightweight cryptography approach to achieve security issues like authentication. We used lightweight cryptographic functions like Symmetric Encryption, XOR and Hashing that does not need a third-party (CA) to validate its authenticity. Therefore, cryptographic operations do not demand high computational resources to be processed. Figure 3.1 presents a summary of the outcomes of every protocol. It shows the new derived keys coloured in red and green respectively, the order of the steps performed to obtain such keys and the participants that hold each of the new derived keys.

Table 3.1 compiles all notations used as well as a brief description of every one.

Table 3.1: Notations used in designed protocols

Notation	Description
Gw_i	Gw_is' device
U_i	i^{th} user
$RN1, RN2, \ldots, RNn$	Random nonces
EN_i	End-node$_i$
$AppSKey_{EN_i}$	Session key used to cypher data to AS from End-node
$NwkSEncKey_{EN_i}$	Session key used to cypher data to NS from End-node
$JSIntKey_{EN_i}$	Network Session key derived during OTAA
$GwSKey_{EN_i}$	Session key used to cypher data to Gw from End-node
$SNwkSIntKey_{EN_i}$	Session key used to calculate partial MIC over uplink messages, full MIC on downlink messages and rejoin request.
$FNwkSIntKey_{EN_i}$	Session key used to calculate partial MIC over uplink messages.
$GwKey_{Gw_i}$	Gateway Symmetric key
$GwKey_{Gw_{j..n}}$	Symmetric keys of other gateways
$AppKey_{ENi}$	Pre-shared root application key
$NwkKey_{ENi}$	Pre-shared root network key
$NAKey_{EN_i}$	Calculated key between $AppKey_{ENi}$ XOR $NwkKey_{ENi}$ to derive $GwSKey_{EN_i}$
$GwEUI_{Gw_i}$	Gateway Extended Unique Identifier
$DevEUI_{EN_i}$	Device Extended Unique Identifier
$GwDevId_{EN_i}$	End-node anonymous identity
$GrpKey_{GrpId}$	Symmetric Group Key for multicast messages
$GrpId$	Group Identifier of a set of gateways connected to a fNS
$MICGw$	MIC used to validate integrity between fNS and Gw_i
Pubkey$_{fNS}$, Privkey$_{fNS}$	fNS's asymmetric key pair
Pubkey$_{NS}$, Privkey$_{NS}$	NS's asymmetric key pair
Pubkey$_{JS}$, Privkey$_{JS}$	JS's asymmetric key pair
ID_{Ui}	Identification of U_i
PW_{Ui}	Password of U_i
$\|$	String concatenation
h.	One way hash function
$\otimes$	Exclusive OR operation
mic.	Message Integrity Code function
aes_cmac	AES Message Authentication Code function
$SEnc(x,y)$	Symmetric encryption of message y using the key x
$SDec(x,y)$	Symmetric decryption of message y using the key x
$MICPy$	Additional MIC to protect Payload Integrity
$MIC\_P_{EN_i}$	MIC to validate message Integrity between EN_i and Gw_i
JS	Join Server
NS	Network Server
AS	Application Server
$DevAddr_{EN_{i_i}}$	Device Address assgined by the network server
$MHDR$	MAC Header
$FHDR$	Frame Header
$Mtype$	Message Type
EJP	Extended For Join Protocol (RFU unused bits)
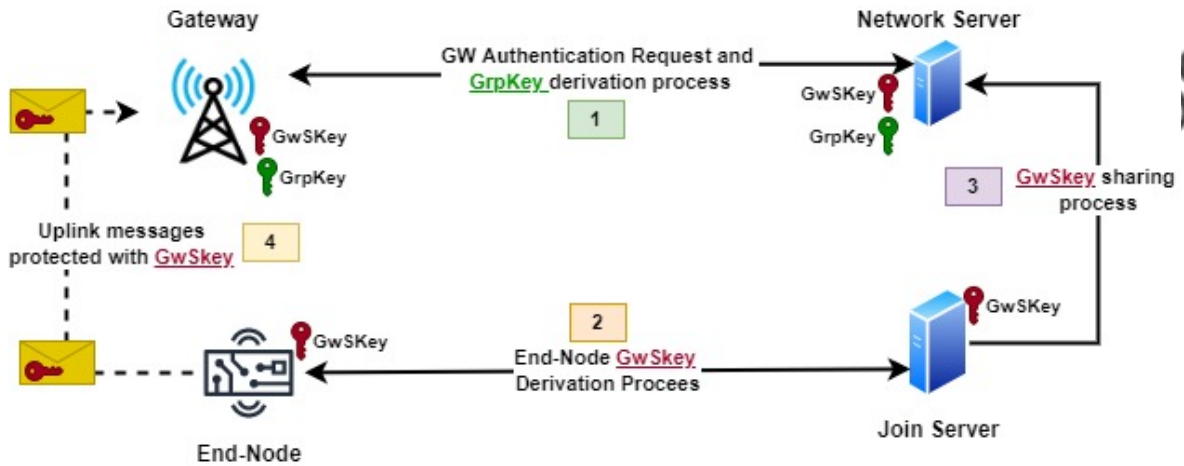
Figure 3.1: Gateway Registration Protocol Summary

Table 3.1: Notations used in designed protocols

Notation	Description
FPort	Optional Port Field
FCtrl	Frame Control
FCnt	Frame Counter
FOpts	Frame Options
Payload	Unencrypted Message Payload
FRMPayload	Encrypted Frame Payload
B0	Uplink B0 MIC computation block format
B1	Uplink B0 MIC computation block format
msg	Whole message that is composed of *MHDR, FHDR, FPort, FRMPayload*

3.1 Gateway Registration Protocol

This protocol registers a gateway within a LoRaWAN network. During this registration, the Gateway (*Gw*) will share its symmetric key with the Network Server (*NS*). In this scenario, it is assumed that the gateway symmetric key will reside in secure place that cannot be tampered. A summary of the steps performed are shown in figure 3.2.
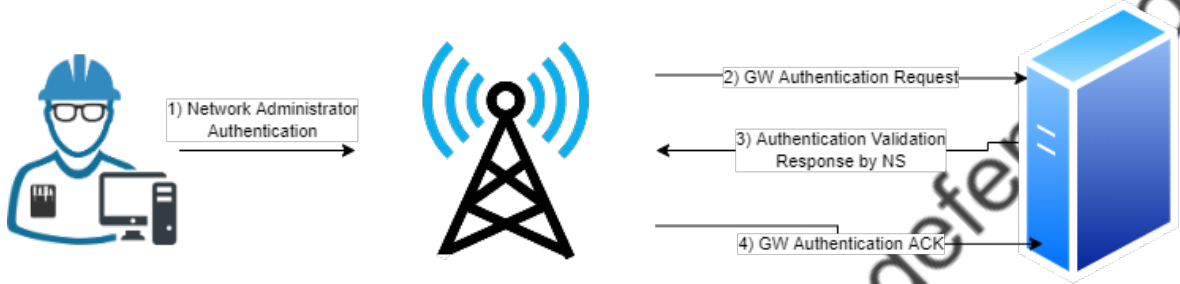
Figure 3.2: Gateway Registration Protocol Summary

In the proposed scenario each *NS* is in charge of one or a group of gateways. According to the LoRaWAN backend specificaction V1.0 [23], *fNS* is in charge of managing Gateways. A Gateway points to a particular *fNS*. There might be several *Gws* deployed within the network and connected to a network server. The number of gateways depend on the number of nodes that can be handled and the scope of the deployed network. This protocol aims to mitigate the vulnerability described as Rogue Gateway attacks. This is a formal protocol that must take place before any *Gw* wants to be part of a LoRaWAN network. The process for registering a gateway into the LoRaWAN network is shown in detail in Figure 3.3, and is executed as follows. For this scenario, it is assumed that the network administrator has to configure the Gw_i to connect to a *fNS* or a set of them.

First, the user in charge of performing the configuration is a network administrator which provides his/her credentials ID_{Ui}, PW_{Ui} into the gateway. Then, the gateway Gw_i generates a random nonce *RN1* and a random symmetric key *RSK*. It also computes **GwSKa**=$h(RSK || GwKey_{Gwi})$, where $GwKey_{Gwi}$ is a symmetric key that comes from factory and is stored in a secure place in the gateway, and calculates **GwInf**=$SEnc(GwSka, RN1||GwEUI_{Gwi})$ where $GwEUI_{Gwi}$ is a 64-bit Unique Identifier of the gateway, and $SEnc(x,y)$ is a symmetric encryption function of a message *y* using the key *x*, *||* is a concatenation operation, and *h(.)* is a one-way hash function. Then, it calculates the following:

- **MReq** = $h(ID_{Ui}||h(PW_{Ui})) \otimes GwSka$ (Where $\otimes$ is an XOR operation.

- **M1** = *(MReq||IDUi||GwInf)*

The gateway (Gw_i) communicates with the *fNS* and asks for gateway registration by sending *M1*. After receiving the request, *fNS* Obtains ID_{Ui} from *M1* and calculates *h(IDUi||h(PWUi)).*
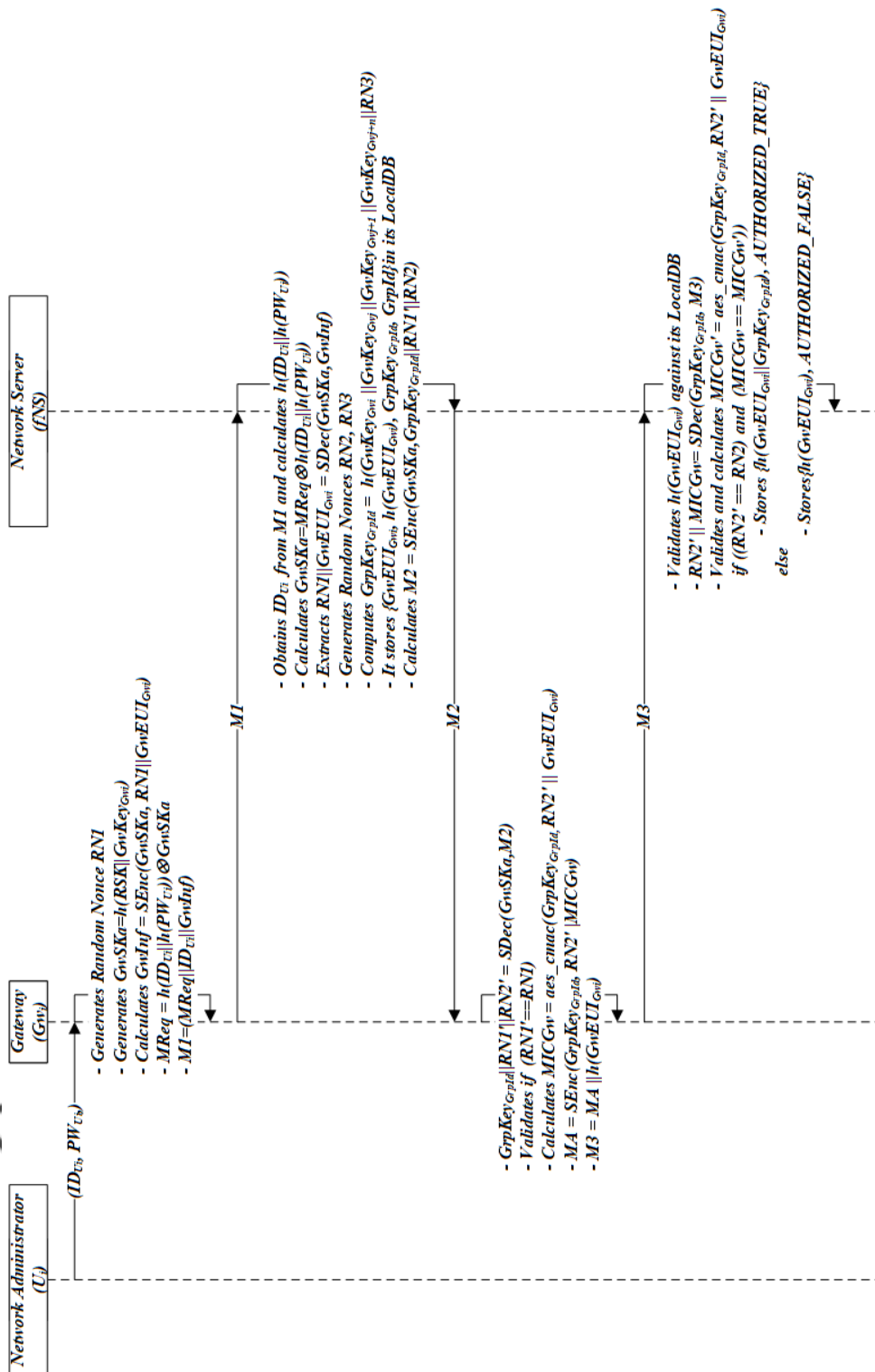
Figure 3.3: Gateway Registration Protocol

It obtains **GwSKa** by executing *MReq $\otimes$ h(IDUi||h(PWUi))* . It extracts *RN1||GwEUI$_{Gwi}$* by performing *SDec(GwSKa, GwInf)* where *SDec(x,y)* is a symmetric decryption function of message *y* using key *x*. It generates two random nonces *RN2* and *RN3* and then computes the symmetric groupkey key for all gateways associated to *fNS* by executing **GrpKey$_{GrpId}$** *= h(GwKey$_{Gw_i}$||GwKey$_{Gw_j}$||GwKey$_{Gw_{j+1}}$||GwKey$_{Gw_{j+n}}$||RN3)*, where *GwKey$_{Gw_n}$* is a symmetric key that belongs to a particular registered gateway *n*. This key is the group symmetric key that will be used by the *fNS* to share multicast messages with its registered gateways. Then, it generates a sequential integer *GrpId* to identify the group of gateways connected to it. It stores *(GwEUI$_{Gwi}$,h(GwEUI$_{Gwi}$),GrpKey$_{GrpId}$,GrpId)* in its LocalDB. For this scenario, every time a gateway is registered, the *GrpKey$_{GrpId}$* will be calculated and shared (multicast) to all the gateways tied to a *fNS*. Finally, it calculates **M2**=*SEnc(GwSka,GrpKey$_{GrpId}$||RN1'||RN2)* and sends it back to *Gw$_i$*.

Once *Gw$_i$* receives *M2*, it obtains *GrpKey$_{GrpId}$||RN1'||RN2'* by decrypting *SDec(GwSKa,M2)*. Then, it validates if the received random nonce *RN1'* matches the previously generated one *RN1*, to guarantee the freshness of the message. If previous validation was correct, it calculates **MICGw** *= aes_cmac(GrpKey$_{GrpId}$, RN2'||GwEUI$_{Gwi}$)*, where *aes_cmac(x,y)* is an AES Message Authentication Code function that uses a key *x* to produce a code of a message *y*. Then, it computes *MA=SEnc(GrpKey$_{GrpId}$, RN2'||MICGw)*. Later, it calculates **M3** *= MA||h(GwEUI$_{Gwi}$)* and sends **M3** to *fNS*.

Finally, upon reception of **M3**, *fNS* obtains *h(GwEUI$_{Gwi}$)* and compares against its LocalDB to obtain the GrpKey$_{GrpId}$ of the gateway that is requesting the registration process used for further decrytption operations. It decrypts *MA* to obtain *RN2' and MICGw* by executing *SDec(GrpKey$_{GrpId}$, MA)* using the key retrieved from its LocalDB. Then, *fNS* calculates **MICGw'** by executing aes_cmac(GrpKey$_{GrpId}$, RN2||GwEUI$_{Gwi}$) where *RN2* is the previous random nonce generated by *fNS*. It compares **MICGw'** against the received **MICGw** to validate that the message has been generated by the gateway requesting registration. Also, it validates *RN2* against *RN2'* to validate the freshness of the message. If both comparisons are valid, *fNS* stores the tuple *(h(GwEUI$_{Gwi}$||GrpKey$_{GrpId}$), AUTHORIZED_TRUE)* in its database to authorize messages coming from the just registered gateway. Otherwise, it prohibits the gateway by registering the tuple *(h(GwEUI$_{Gwi}$), AUTHORIZED_FALSE)*. For future use, the *fNS* will first validate the authorization status of a gateway before accepting/forwarding packets to other network servers.

The proposed scenario applies for home or roaming scenarios. In case of home deployment, *NS* acts as *fNS, sNS* and *hNS* according to the LoRaWAN backend interfaces specification [23]. In our proposal the *fNS* plays the role of *NS*.

It is important to consider that if a gateway *Gw$_i$* is unregistered or registered into a group, then a recalculation procedure should be conducted by *fNS* and the resulting key must be shared among the group through multicast. For the leaving scenario, the multicast operation

will not consider the leaving gateway, it will unauthorize Gw_i in the *fNS* database by registering the tuple *(h(GwEUI$_{Gwi}$), AUTHORIZED_FALSE)* and calcuating the new group key as follows **GrpKey$_{GrpId}$** = *h(GwKey$_{Gw_j}$||GwKey$_{Gw_{j+1}}$||GwKey$_{Gw_{j+n}}$)*.

3.2 Gateway Session Key Derivation Protocol

3.2.1 Home Scenario

The process for deriving the Gateway Session Key **(GwSKey$_{ENi}$)** in a LoRaWAN Home Scenario is executed as follows (see Figure 3.4). The steps highlighted in green are part of our contribution.

This procedure is executed during the Join-Procedure OTAA described in the LoRaWAN 1.1 specification. Once the *EN* has passed all validation procedures by *NS* and *JS*, it starts the Session Key Derivation Process. According to the specification, there are five keys that are derived and shared with the Network Server and the Application Server. In this scenario, a new symmetric key based on previous Network Key and Application Key is calculated by using an XOR function and then using it to calculate a sixth session key known as **GwSKey$_{ENi}$**. The following steps are executed:

- **NAKey$_{EN_i}$**=*NwkKey$_{ENi}$ $\otimes$ AppKey$_{ENi}$*

- **GwSKey$_{ENi}$**=*aes_cmac(NAKey$_{EN_i}$,h(DevEUI$_{ENi}$||
  DevNonce$_{ENi}$||JoinNonce$_{ENi}$||JoinEUI$_{ENi}$))*

Once obtained, *JS* generates *M1* and asymmetrically encrypts it with the public key of the Netowrk Server *(NS) NS$_{Pubkey}$*. *JS* is able to determine *GwEUI$_{Gwi}$* as it comes in the payload of the Join-Procedure. JS computes:

- **M1**= *AEnc(NS$_{Pubkey}$,GwEUI$_{Gwi}$||DevEUI$_{EN_i}$
  ||GwSKey$_{EN_i}$)*

- **GwDevId$_{EN_i}$**=*h(GwSKey$_{EN_i}$||DevEUI$_{EN_i}$)*

JS stores *{GwDevId$_{EN_i}$, DevEUI$_{EN_i}$}* for further processing, *M1* is sent back to the network server by using the sharing process of session keys and JoinAccept is forwarded to End Node.

The network server *(NS)* receives *M1* and decrypts it by executing *ADec(NS$_{Pubkey}$,M1)* to obtain *GwEUI$_{Gwi}$||DevEUI$_{EN_i}$||GwSKey$_{EN_i}$*, where *ADec(x,y)* is an asymmetric decryption function that uses a public key *x* to decrypt a message *y*. Then, it calculates **M2** by executing **M2**=*SEnc(GrpKey^{GrpId},GwEUI$_{Gwi}$||DevEUI$_{ENi}$||GwSKey$_{ENi}$)* and sends it to the gateway.

Application Server
AS

Join Server
JS

Network Server
NS

Gateway
Gw_{ij}

End-node
EN_i

Join-Request

If (DevNonce_Actual$_{EN}$!= DevNonce_Previous$_{EN}$)
Forward Join_Request$_{EN}$
DevNonce_Previous$_{EN}$ = DevNonce_Actual$_{EN}$
Else
Exit

Join-Request

If EN$_i$ in Supported_Device_List [DevEUI$_{EN}$, NwkKey$_{EN}$, AppKey$_{EN}$]
Generate JoinNonce$_{EN}$
Else
Exit

JS Session Keys Derivation
FNwkSIntKey$_{EN}$, SNwkSIntKey$_{EN}$,
NwkSEncKey$_{EN}$, AppSKey$_{EN}$, JSIntKey$_{EN}$
NAKey$_{EN}$=NwkKey$_{EN}$ ⊗ AppKey$_{EN}$
GwSKey$_{EN}$=aes_cmac(NAKey$_{EN}$, h(DevNonce$_{EN}$||JoinNonce$_{EN}$||JoinEUI$_{EN}$)
M1= AEnc(NS$_{PubKey}$, GwEUI$_{Gw}$||DevEUI$_{EN}$||GwSKey$_{EN}$)
GwDevId$_{EN}$=h(GwSKey$_{EN}$||DevEUI$_{EN}$)
Stores {DevEUI$_{EN}$, GwDevId$_{EN}$}

Join-Accept [DevEUI$_{EN}$, NwkKey$_{EN}$, AppKey$_{EN}$]

Share NwkSessionKeys

M1

GwEUI$_{Gw}$||DevEUI$_{EN}$||GwSKey$_{EN}$=ADec(NS$_{PubKey}$, M1)
M2 = SEnc(GrpKey$_{GrpId}$, GwEUI$_{Gw}$||DevEUI$_{EN}$||GwSKey$_{EN}$)

M2

GwEUI$_{Gw}$||DevEUI$_{EN}$||GwSKey$_{EN}$ = SDec(GrpKey$_{GrpId}$, M2)
GwDevId$_{EN}$=h(GwSKey$_{EN}$||DevEUI$_{EN}$)
Stores {GwDevId$_{EN}$, GwSKey$_{EN}$, DevEUI$_{EN}$} in LocalDB

Share AppSKey

Join-Accept

EN Session Keys Derivation
FNwkSIntKey$_{EN}$, SNwkSIntKey$_{EN}$,
NwkSEncKey$_{EN}$, AppSKey$_{EN}$, JSIntKey$_{EN}$
NAKey$_{EN}$=NwkKey$_{EN}$ ⊗ AppKey$_{EN}$
GwSKey$_{EN}$=aes_cmac(NAKey$_{EN}$, h(DevNonce$_{EN}$||JoinNonce$_{EN}$||JoinEUI$_{EN}$)

Figure 3.4: Gateway Session Key Registration Protocol Home Sceneraio

Figure 3.5: Gateway Session Key Registration Protocol Roaming Scenario

The Gateway receives *M2* and decrypts it by executing *SDec(GrpKey$_{GrpId}$,M2)* to obtain GwEUI$_{Gwi}$||DevEUI$_{EN_i}$||GwSKey$_{EN_i}$.Then, it calculates *h(GwSKey$_{ENi}$||DevEUI$_{ENi}$)* to generate a unique anonymous identifier for the end-node. Gw$_i$, also stores a maximum idle time (defined by the network administrator) *Max Idle Time (MIT$_{ENi}$)* for such *EN$_i$* to prevent storing data of devices that are not using that gateway or that devices that have not transmitted data in a period of time greater than (MIT$_{ENi}$). Finally, the gateway stores *{GwDevId$_{ENi}$, GwSKey$_{ENi}$, MIT$_{ENi}$}* in its database for decrypting further messages sent by a particular end-node.

According to the specification, once the Join-Accept message was received, the end-node must derive session keys. At this point the End-Node calculates:

- **NAKey$_{EN_i}$**=NwkKey$_{ENi}$ $\otimes$ AppKey$_{ENi}$

- **GwSKey$_{ENi}$**=aes_cmac(NAKey$_{EN_i}$,h(DevEUI$_{ENi}$|| DevNonce$_{ENi}$||JoinNonce$_{ENi}$||JoinEUI$_{ENi}$))

to obtain the session key used to send messages to a particular Gateway. The *GwSKey$_{ENi}$* is a 128-bit key. This key will be renewed on every Re-Join procedure according to the protocol described before. The key is assumed to be stored in a secure place with tamper proof mechanisms.

3.2.2 Roaming Scenario

In case the LoRaWAN infrastructure is working on Roaming Scenario, the following considerations are in place, and the protocol for such scenario is shown in Figure 3.5.

According to the LoRaWAN backend specification [23] when an End-Node *EN$_i$* works over roaming the following additional steps are required once a Join Request has been dispatched. First, the Join Request arrives to *NS2* and it has to determine if it is acting as the *(hNS)* for the *ENi*. It also has to determine if it has been identified to work with *JS* which is identified by *JoinEUI*, if such is not the case, the process must terminate at this point. Otherwise, it has to perform a DNS lookup to identify the IP address of *JS*. In case *NS2* is not able to identify the *(hNS)*, it has to send a request that contains *DevEUI* to *JS* to retrieve such information. *JS* has to respond to such request either with a succesfull response containing the *NetID* of *NS1* if *NS2* belongs to authorized networks or with a No Roaming Agreement Response. Then, *NS2* performs a DNS lookup to obtain the IP Address of *NS1 (hNS)* by using the previously obtained *NetID* and also it sends a request *(ProfileReq)* containing the DevEUI to retrieve profile information of the device. Later, if the device is allowed for roaming *NS1* should inform to *NS2* through a successfull notification *(ProfileAns)*. If the device has not been authorized a failure notification is forwarded. Once *NS2* received a successfull confirmation with handover roaming type, it has to start a new message request *(HRStartReq)* to *NS1* that contains the JoinRequest, MACVersion, ULMetaData, DevAddr, DLSettings, RxDelay, CFList and

Device Profile Timestamp. *NS1* forwards the Join-Request to *JS* to start the *JS* Session Key Derivation process. During this process, the derivation of the new key $GwSKey_{EN_i}$ is the same as described in previous section. *JS* send an answer message *(HRStartAns)* to *NS2* containing the roaming activation status as well as Join-Accept response. The differences here compared to previos home scenario are that **M1** will be encrypted with the public key of **M2** and *M2* is message encrypted with the public key of *NS1*, as described below:

- **M1** $= AEnc(NS1_{PubKey}, GwEUI_{Gw_i}||DevEUI_{EN_i}||GwSkey_{EN_i})$

- **M2** $= AEnc(NS2_{PubKey}, ADec(NS1_{PubKey}, M1))$

Upon reception of *M2*, *NS2* calculates *M3* by executing $ADec(NS2_{PubKey}, M2)$ and then builds *M4* by executing $SEnc(GrpKey_{GrpId}), M3$, *M4* is then forwarded to the gateway.

Once Gw_i receives *M4*, it executes $SDec(GrpKey_{GrpId}), M4)$ to obtain $GwEUI_{Gw_i}||DevEUI_{EN_i}||GwSkey_{EN_i}$. Then, it calculates $GwDevId_{EN_i} = h(GwSkey_{EN_i}||DevEUI_{EN_i})$ and stores $\{ GwDevId_{EN_i}, GwSkey_{EN_i}\}$ in tis LocalDB.

Finally, EN_i derives $GwSkey_{EN_i}$ in the same way as stated in the previous section (Home Scenario).

3.3 Uplink Messages through authenticated gateways

The process for sending uplink messages through a registered gateway is described as follows. This procedure is executed after the OTAA Join-Procedure has been successfully acknowledge with a Join-Accept message. It applies for Unconfirmed Data Up Messages and is divided in two scenarios.

The first one applies when a end-node EN_i has joined (OTAA Activation) through a registered gateway Gw_i which has already been registered through the *fNS*. The second scenario applies when an end-node EN_i wants to send a message over a registered gateway but EN_i is not registered over that Gw_i. In our proposal a Gw_i must be registered over the LoRaWAN infraestructure before forwarding any message.

First of all, EN_i calculates all the following as part of the construction of the uplink message according to LoRaWAN 1.1 specification [22]:

- **MHDR**$=Mtype||$**EJP**$||Major$

- **FHDR**$=DevAddr||FCtrl||FCnt||FOpts$

- **msg**$=MHDR||FHDR||FPort||FRMPayload$

- **cmacS**$=aes\_cmac(FNwkSIntKey_{ENi}, B1||msg)$

35

- **cmacF**=aes_cmac(FNwkSIntKey$_{ENi}$,B0||msg)

- **MIC**=cmacS[0..1]||cmacF[0..1]

- **FRMPayload** = SEnc(AppSKey$_{ENi}$,Payload)

- **PHYPayload**=MHDR||FHDR||FPort||FRMPayload

Then, EN_i calculates **GwDevId$_{ENi}$** =h(GwSKey$_{ENi}$||DevEUI$_{EN_i}$) which is a temporary anonymous identifier that depends on a session key previously established with a Join-Accept and changes with every message, it is 9 bytes long distributed as follows. The hashed parameter has been divided in 4 parts, the protocol will randomly take one of the parts (8 bytes) and will add a ninth byte to mark the corresponding portion sent.

For the **FHDR**, we propose to use dynamic Device Address to make sniffing harder. The new device Address (**DevAddr**) will be calculated as follow **DevAddr$_{ENi}$** = SEnc(**GwDevId$_{ENi}$**, DevAddr). This parameter will use the full key GwDevId$_{ENi}$ generated and will be increased with every time an uplink message is sent.

Also, according to LoRaWAN 1.1 Specification [22] there are two unused (4..2) bits in MAC Header that are reserved for future use (RFU). This proposal uses these bits so that EN_i defines the type of message to be built by creating **EJP** which will take the value **EJP = 0x01** to identify a secured type of message to be delivered through a registered gateway (Gw_i).

In addition, our proposal considers adding an integrity MIC for *FRMPayload*. EN_i calculates a new MIC after *FRMPayload* has been symmetrically ciphered. This new MIC is 4 bytes length as is calculated as follows **MIC_Py**=aes_cmac(AppSKey$_{ENi}$, FRMPayload)[0..3] and will be used to validate, accept or decline a message if *FRMPayload* was tampered by malicious users.

Moreover, our proposal considers adding a 4-byte MIC for validating messages sent from EN_i to Gw_i. This MIC is calculated as follows by EN_i

MIC_P$_{ENi}$=aes_cmac(GwSKey$_{ENi}$,msg||GwDevId$_{ENi}$||FCntUp)[0..3].

Finally, once all previous components have been calculated, EN_i calculates

M1 = MHDR||FHDR||FPort||FRMPayload||MIC||**MICPy**||**MIC_P$_{EN_i}$** and sends it to Gw_i.

3.3.1 Protocol for sending uplink messages over authenticated End-Nodes and gateways (UMOAEG).

The purpose of this protocol is to deliver messages over a Gw_i that has already been registered within LoRaWAN infrastructure by using an authenticated *Eni*.

Once Gw_i receives *M1*, it verifies if *GwDevId$_{ENi}$* is in the *LocalDB GwDevId$_{ENi}$, GwSKey$_{ENi}$*. If such validation is true, it calculates **DevAddr** = SDec(**GwDevId$_{ENi}$**, DevAddr$_{ENi}$), extracts

$GwSKey_{ENi}$ and calculates $\textbf{MIC\_P}_{\textbf{ENi}}\textbf{'}=aes\_cmac(GwSKey_{ENi},msg||GwDevId_{ENi}$ $||FCntUp)[0..3]$

and compares against MIC_P_{ENi} from $M1$, if it matches, Gw_i builds

$\textbf{M2} = MHDR||FHDR||FPort||FRMPayload||MIC||\textbf{MICPy}$ and calculates $\textbf{M3}=SEnc(GrpKey_{GrpId},M2)$ which is then forwarded to the Network Server (NS).

Then, NS calculates $SDec(GrpKey_{GrpId},\textbf{M3})$ to obtain $\textbf{M2}$. It then calculates $\textbf{cmacS}$ and $\textbf{cmacF}$ to validate $\textbf{MIC}$ according to [22], if such validation is true $\textbf{M2}$ is then forwarded to the Application Server (AS).

Finally, after AS receives $M2$, it calculates $\textbf{MIC\_Py'}=aes\_cmac(AppSKey_{ENi}, FRMPayload)[0..3]$ and validates against $\textbf{MIC\_Py}$ from $M2$ to verify that FRMpayload has not been altered whilst in transit. If that validation was succesfull it then executes $SDec(AppSKey_{ENi},FRMPayload)$ to obtain Payload in plain text and then decodes it; otherwise, AS aborts the process. The designed protocol is shown in Figure 3.6.

3.3.2 Protocol for sending uplink messages over unauthenticated End-Nodes and gateways (UMOUEG)

The protocol designed can be seen in Figure 3.7 In this scenario, the purpose is to deliver an uplink message over an authenticated EN_i a registered Gw_i but the session key has not been delivered yet to Gw_i. First of all, once Gw_i receives $M1$, it verifies if $GwDevId_{ENi}$ is not in the $LocalDB$ $GwDevId_{ENi}$, $GwSKey_{ENi}$. If so, it temporally stores M1,$GwDevId_{ENi}$,$GwSKey_{ENi}$ in a TempDB. Then, it extracts $DevEUI_{ENi}$ from LocalDB, calculates $\textbf{M2}=SEnc(GrpKey_{GrpId},GwDevId_{ENi}||GwEUI_{Gwi}||DevEUI_{ENi})$ and forwards it to NS.

Upon reception of $M2$, NS executes $SDec(GrpKey_{GrpId},M2)$ to obtain $GwDevId_{ENi}||GwEUI_{Gwi}||DevEUI_{ENi}$ and then calculates $\textbf{M3}=AEnc(Pubkey_{JS},GwDevId_{ENi}||GwEUI_{Gwi}||DevEUI_{ENi})$ using asymmetric encryption with the public key of $\textbf{JS}$ and forwards it.

Once $\textbf{JS}$ receives $M3$ it asymmetrically decrypts it by executing $ADec(Pubkey_{JS},M3)$ to obtain $GwDevId_{ENi}||$ $GwEUI_{Gwi}||$ $DevEUI_{ENi}$ and then it validates if $GwDevId_{ENi}$ is in LocalDB and $DevEUI_{ENi}$ is in the Supported Device List of JS, if so it calculates the following:

- $\textbf{NAKey}_{\textbf{EN}_i}=NwkKey_{ENi} \otimes AppKey_{ENi}$

- $\textbf{GwSKey}_{\textbf{ENi}}=aes\_cmac(NAKey_{ENi},h(DevEUI_{ENi}$ $||DevNonce_{ENi}||JoinNonce_{ENi}||JoinEUI_{ENi}))$

- $\textbf{M4}= AEnc(Pubkey_{NS} ,GwEUI_{Gwi}||DevEUI_{ENi}||$ $GwSKey_{EN_i}||GRANTED)$

On the other hand if there is no match JS calculates:

Figure 3.6: Uplink messages over authenticated End-Node and Gateway

End-node EN_i   Gateway GW_i   Network Server NS   Join Server JS   Application Server AS

$GwDevId_{ENi}=h(GwSKey_{ENi}||DevEUI_{ENi})[0..3]$
$DevAddr_{ENi}=SEnc(GwDevId_{ENi}, Dev\text{-}Addr)$
$MIC\_P_{ENi}=aes\_cmac(GwSKey_{ENi}, msg||GwDevId_{ENi}||FCntUp)[0..3]$
$MICP_y = aes\_cmac(AppSKey_{ENi}, FRMPayload)[0..3]$
$M1=MHDR||FHDR||FPort||FRMPayload||MIC||MICP_y||MIC\_P_{ENi}||GwDevId_{ENi}$

$M1$

Validates if (GwDevId_{ENi} is in LocalDB)
extracts GwSKey_{ENi}
calculates Dev-Addr=SDec(GwDevId_{ENi}, Dev-Addr_{ENi})
calculates MIC_P_{ENi}'=aes_cmac(GwSKey_{ENi}, msg||GwDevId_{ENi}||FCntUp)[0..3]
if (MIC_P_{ENi}' == MIC_P_{ENi})
$M2 = MHDR||FHDR||FPort||FRMPayload||MIC||MICP_y$
Forwards M3 = SEnc(GrpKey_{Gp4}, M2)

else

Abort process

$M3$

$M2 = SDec(GrpKey_{Gp4}, M3)$
Validates MIC
Forwards M2

$M2$

Calculates MICP_y = aes_cmac(AppSKey_{ENi}, FRMPayload)[0..3]
if (MICP_y' == MICP_y)
Payload=SDec(AppSKey_{ENi}, FRMPayload)
Decode Payload

Else

Abort process

Figure 3.7: Uplink messages over unauthenticated End-Node and Gateway

Diagram participants: End-node EN, Gateway Gw, Network Server NS, Join Server JS, Application Server AS.

Definitions:

$GwDevId_{ENi} = h(GwSKey_{ENi} ||DevEUI_{ENi})[0..3]$

$DevAddr_{ENi}=SEnc(GwDevId_{ENi}, DevAddr)$

$MIC\_P_{ENi} = aes\_cmac(GwSKey_{ENi}, msg||GwDevId_{ENi}||FCntUp)[0..3]$

$MICPy = aes\_cmac(AppSKey_{ENi}, FRMPayload) [0..3]$

$M1=MHDR||FHDR||FPort||FRMPayload||MIC||MICPy ||MIC\_P_{ENi}||GwDevId_{ENi}$

M1 → (End-node to Gateway)

Gateway:

if $GwDevId_{Gwi}$ not in LocalDB

  Temporally stores {$M1$, $GwDevId_{ENi}$, $GwSKey_{ENi}$}

  Extracts $DevEUI_{ENi}$ from LocalDB

  $M2=SEnc(GrpKey_{Grpia}, GwDevId_{ENi} ||DevEUI_{Gwi} ||DevEUI_{ENi})$

M2 →

Network Server:

$GwDevId_{ENi} ||GwEUI_{Gwi} ||DevEUI_{ENi}|DevEUI_{ENi}=SDec(GrpKey_{Grpia}, M2)$

$M3=AEnc(Pubkey_{JS}, GwDevId_{JS} GwDevId_{ENi} ||GwEUI_{Gwi}||DevEUI_{ENi})$

M3 →

Join Server:

- Obtains $GwDevId_{ENi} ||GwEUI_{Gwi} ||DevEUI_{ENi} = ADec(Pubkey_{JS},M3)$

if {$GwDevId_{ENi},DevEUI_{ENi}$} in Local DB and ($DevEUI_{ENi}$ in Supported_Device_List

  $NAKey_{ENi}=NwkKey_{ENi}\otimes AppKey_{ENi}$

  $GwSKey_{ENi}=aes\_cmac(NAKey_{ENi}, h(DevNonce_{ENi}||JoinNonce_{ENi}||JoinEUI_{ENi}))$

  $M4= AEnc(NS_{PubKey}, GwEUI_{ENi}||DevEUI_{ENi}||GwSKey_{ENi}||GRANTED)$

else

  $M4=AEnc(NS_{PubKey}, GwEUI_{Gwi}||DevEUI_{ENi}||UN\text{-}AUTHORIZED)$

M4 →

Network Server:

$M5 = ADec(NS_{PubKey}, M4)$

$M6=SEnc(GrpKey_{Grpia}, M5)$

M6 →

Gateway:

$MR = GwEUI_{Gwi}||DevEUI_{ENi}||GwSKey_{ENi}||[STATUS] = SDec(GrpKey_{Grpia}, M6)$

if MR contains GRANTED in [STATUS]

  Retrieves {$M1$, $GwDevId_{ENi}$, $GwSKey_{ENi}$} by using $GwSKey_{ENi}$

  calculates $DevAddr=SDec(GwDevId_{ENi}, DevAddr_{ENi})$

  calculates $MIC\_P_{ENi}'=aes\_cmac(GwSKey_{ENi}, msg||GwDevId_{ENi}||FCntUp)[0..3]$

  if $(MIC\_P_{ENi}' == MIC\_P_{ENi})$

    Stores {$GwDevId_{ENi}$, $GwSKey_{ENi}$, $DevEUI_{ENi}$ }

    $MP1 = MHDR||FHDR||FPort||FRMPayload||MIC||MICPy$

    Forwards $MP2 = SEnc(GrpKey_{Grpia} , MP1)$

  else

    Abort process

MP2 →

Network Server:

$MP1 = SDec(GrpKey_{Grpia}, MP2)$

Validates MIC

Forwards MP1

MP1 →

Application Server:

Calculates $MICPy = aes\_cmac(AppSKey_{ENi}, FRMPayload) [0..3]$

if $(MICPy' == MICPy)$

  $Payload=SDec(AppSKey_{ENi},FRMPayload)$

  Decode Payload

Else

  Abort process

M4= *AEnc(Pubkey$_{NS}$,GwEUI$_{Gwi}$||DevEUI$_{EN_i}$||UNAUTHORIZED)*. Then **M4** is sent back to *NS*.

NS receives *M4* and then asymmetrically decrypts it by executing *ADec(Pubkey$_{NS}$, M4)* to obtain *M5* to calculate **M6**=*SEnc(GrpKey$_{GrpId}$,M5)* and then sents it back to *Gw$_i$*.

Gw$_i$ receives *M6* and symetrically dercrypts by executing *SDec(GrpKey$_{GrpId}$,M6)* to obtain *MR = GwEUI$_{Gw_i}$'||DevEUI$_{EN_i}$'||GwSkey$_{EN_i}$'||STATUS*. Then, *Gw$_i$* validates if *MR* contains GRANTED response, if so, it then retrieves M1,*GwDevId$_{EN_i}$* ,*GwSkey$_{EN_i}$* from TempDB by using *GwSkey$_{EN_i}$* and calculates **MIC_P$_{ENi}$**' = *aes_cmac(GwSKey$_{ENi}$, msg||GwDevId$_{EN_i}$||FCntUp)[0..3]*. It compares if **MIC_P$_{ENi}$**' is equal to **MIC_P$_{ENi}$** obtained from M1, stores (*GwDevId$_{EN_i}$* , *GwSkey$_{EN_i}$*, *DevEUI$_{EN_i}$*), calculates **DevAddr** = *SDec(**GwDevId$_{ENi}$**, DevAddr$_{ENi}$)*, builds **MP1**=*MHDR||FHDR||FPort||FRMPayload||MIC||**MICPy***, calculates **MP2**=*SEnc(GrpKey$_{GrpId}$, MP1)* and forwards it to *NS* .

Then, *NS* calculates *SDec(GrpKey$_{GrpId}$,**MP2**)* to obtain **M2**. It then calculates **cmacS** and **cmacF** to validate **MIC** according to [22], if such validation is true **M2** is then forwarded to the Application Server *(AS)*.

Finally, after *AS* receives *M2*, it calculates **MIC_Py**'=*aes_cmac(AppSKey$_{ENi}$, FRMPayload)[0..3]* and validates against **MIC_Py** from *M2* to verify that FRMpayload has not been altered whilst in transit. If that validation was succesfull it then executes *SDec(AppSKey$_{ENi}$,FRMPayload)* to obtain Payload in plain text and then decodes it; otherwise, *AS* aborts the process.

3.4   Security issues to be addressed

The previous designed protocols are mean to address authentication issues between End-Nodes and Gateways, as well as Gateways with Network Servers. Currently, LoRaWAN specification does not specify authentication issues of the gateway. With the designed protocols a gateway will have to be registered before connecting to a LoRaWAN network. In addition, this trusting relation that we are proposing between gateway and End-Nodes will allow secure message delivery for Class B devices which are configured to receive messages directly from the gateway to attemp some tasks for future works.

Chapter 4

EVALUATION

In this chapter, we evaluate the proposed protocols from the security and the performance perspectives. Section 4.1 performs the analysis from the security perspective by using the following approaches:

1. BAN Logic to formally analyze the security of The Gateway Registration Protocol (GRP).

2. Scyther Tool to evaluate the security level of GRP and the other protocols: Gateway Session Key Derivation Protocol (GSKDP), UMOAEG and UMOUEG.

3. An informal analysis to evaluate possible attacks over the proposed protocols.

4. Analysis of the number and types of cryptographic operations included.

Section 4.2 evaluates the impact in terms of perfomance of the solution over the End-Node by evaluating RAM, CPU and Power Consumption. Finally in Section 4.3 we discuss the results obtained to determine the impact generated over EN_i devices.

4.1 Security Analysis

4.1.1 Formal Analysis

In this section, we demonstrate the security of the Gateway Registration Protocol by using BAN logic which is a group of rules, created by Burrows–Abadi–Needham, used to evaluate protocols that exchange information. It is used to determine if information is trusted and secure against eavesdropping. This logic contains notations, rules, goals, idealized forms of messages and assumptions which are used to proof the security of the protocol that is being examined [65].

BAN logic notations

The following Table 4.1 presents the notations used for BAN logic.

Table 4.1: Notations used in BAN logic.

Notation	Description
$X \mid\equiv Y$	*X* believes a statement *Y*
#(Y)	*X* is updated and fresh
$X \triangleleft Y$	*X* sees that *Y*
$X \mid\sim Y$	*X* once said the statement *Y*
$X \Rightarrow Y$	*X* controls that *Y*
$X \xleftrightarrow{K} Y$	*K* is a secret shared key between *X* and *Y*
$\xmapsto{K} X$	*X* has *K* as a public key
$\{Y\}_K$	*Y* is encrypted with *K*
$\langle Y \rangle_K$	*Y* is combined with *K*

BAN logic rules

BAN logic uses some rules to verify the security of a protocol. These rules are described below:

- **Message meaning rule:** this rule concerns message interpretation when shared keys are used. It states that if X believes in shared key with Z and itself and X sees a message Y encrypted with K, then X believes that Z once said Y.

- **Nonce verification rule:** it allows to determine that a message is recently generated (fresh) and that a sender believes that message is fresh.

- **Jurisdiction rule:** describes that a principal X is going to trust beliefs where Z has control over (jurisdiction).

- **Feshness rule:** determines that a message has been created recently.

- **Belief rule:** represents things that can be believed but have not necessarily been sent.

The following are the rules of BAN logic:

1. Message meaning rule: $\frac{X\mid\equiv X\xleftrightarrow{K}Z,\, X\triangleleft\{Y\}_K}{X\mid\equiv Z\mid\sim Y}$

2. Nonce verification rule: $\frac{X\mid\equiv\#(Y),\, X\mid\equiv Z\mid\sim Y}{X\mid\equiv Z\mid\equiv Y}$

3. Jurisdiction rule: $\frac{X\mid\equiv Z\Rightarrow Y,\, X\mid\equiv Z\mid\equiv Y}{X\mid\equiv Y}$

4. Freshness rule: $\frac{X|\equiv\#(Y)}{X|\equiv\#(Y,W)}$

5. Belief rule: $\frac{X|\equiv(Y,W)}{X|\equiv Y}$

Security Goals

The following are the goals defined for the Gateway registration protocol:

Goal 1: $Gw_i \mathrel{|\equiv} (Gw_i \xleftrightarrow{GrpKey_{\mathsf{GrpId}}} fNS)$

Goal 2: $fNS \mathrel{|\equiv} (Gw_i \xleftrightarrow{GrpKey_{\mathsf{GrpId}}} fNS)$

Goal 3: $Gw_i \mathrel{|\equiv} fNS \mathrel{|\equiv} (Gw_i \xleftrightarrow{GrpKey_{\mathsf{GrpId}}} fNS)$

Goal 4: $fNS \mathrel{|\equiv} Gw_i \mathrel{|\equiv} (Gw_i \xleftrightarrow{GrpKey_{\mathsf{GrpId}}} fNS)$

Idealizaed forms of messages

The idealized form of the messages of our protocol are shown below:

Msg1: $Gw_i \rightarrow fNS : \{GwSKa\}_{h(ID_{\mathsf{Ui}}||h(PW_{\mathsf{Ui}}))}, ID_{\mathsf{Ui}}, \{RN1, GwEUI_{\mathsf{Gwi}}\}_{GwSKa}$

Msg2: $fNS \rightarrow Gw_i : \{RN1, RN2, Gw_i \xleftrightarrow{GrpKey_{\mathsf{GrpId}}} fNS\}_{GwSKa}$

Msg3: $Gw_i \rightarrow fNS : \{RN2, MICGw, h(GwEUI_{\mathsf{Gw_i}})\}_{Gw_i \xleftrightarrow{GrpKey_{\mathsf{GrpId}}} fNS}$

Assumptions

The assumptions are listed below:

A$_1$: $fNS \mathrel{|\equiv} (Gw_i \xleftrightarrow{h(ID_{\mathsf{Ui}})||h(PW_{\mathsf{Ui}})} fNS)$

A$_2$: $fNS \mathrel{|\equiv} \#(RN1)$

A$_3$: $Gw_i \mathrel{|\equiv} (Gw_i \xleftrightarrow{h(ID_{\mathsf{Ui}})||h(PW_{\mathsf{Ui}})} fNS)$

A$_4$: $Gw_i \mathrel{|\equiv} \#(RN2)$

Proof using BAN logic

1. According to **Msg1**, the following is obtained:

$(S_1) : fNS \triangleleft GwSKa_{h(ID_{\mathsf{Ui}} || h(PW_{\mathsf{Ui}}))}, ID_{\mathsf{Ui}}, \{RN1, GwEUI_{\mathsf{Gw_i}}\}_{GwSKa}$

43

2. By using S_1 and A_1 with the message meaning rule, we obtain:

$$(S_2) : fNS \mid\equiv GWi \mid\sim \{(h(ID_{\mathsf{Ui}}||h(PW_{\mathsf{Ui}})), RN1, GwEUI_{\mathsf{Gw_i}}\}_{GwSKa}$$

3. Using S_2 and A_2, with the freshness rule, the following is obtained:

$$(S_3) : fNS \mid\equiv \#(GWSKa_{(h(ID_{\mathsf{Ui}} \,||\, h(PW_{\mathsf{Ui}}))}), ID_{\mathsf{Ui}}, \{RN1, GwEUI_{\mathsf{Gw_i}}\}_{GwSKa}$$

4. By using S_1 and A_1 with the message meaning rule, we obtain:

$$(S_4) : fNS \mid\equiv GW_{\mathsf{i}} \mid\sim GWSKa$$

5. By using Nonce Verification Rules, S_3 and S_4, we obtain:

$$(S_5) : fNS \mid\equiv GW_{\mathsf{i}} \mid\equiv GWSKa$$

6. According to S_5 and Jurisdiction rule, we obtain:

$$(S_6) : fNS \mid\equiv GWSKa$$

7. According to **Msg2**, we obtain:

$$(S_7) : Gw_{\mathsf{i}} \lhd \{RN1, RN2, Gw_{\mathsf{i}} \xleftarrow{\; GrpKey_{\mathsf{Gprld}} \;} fNS\}_{GwSKa}$$

8. Using S_6 and S_7 with the message meaning rule, we obtained:

$$(S_8) : Gw_{\mathsf{i}} \mid\equiv fNS \mid\sim Gw_{\mathsf{i}} \xleftarrow{\; GrpKey_{\mathsf{Gprld}} \;} fNS$$

9. Using S_8, A_2 and A_4 with the nonce verification rule, we obtained:

$$(S_9) : Gw_{\mathsf{i}} \mid\equiv fNS \mid\equiv Gw_{\mathsf{i}} \xleftarrow{\; GrpKey_{\mathsf{Gprld}} \;} fNS$$

$$(\mathbf{Goal3})$$

10. Using S_9 and jurisdiction rule, the next is obtained:

$$(S_{10}) : Gw_i \mathrel{|}\equiv Gw_i \xleftrightarrow{GrpKey_{\mathsf{Gprld}}} fNS$$

$$(\mathbf{Goal1})$$

11. By using the Key Generation Algorithm of the Protocol (Since $Gw_i \xleftrightarrow{GrpKey_{\mathsf{Gprld}}} fNS$ is generated by *fNS*)

$$(S_{11}) : fNS \mathrel{|}\equiv Gw_i \xleftrightarrow{GrpKey_{\mathsf{Gprld}}} fNS$$

$$(\mathbf{Goal2})$$

12. According to **Msg3**, the following is obtained:

$$(S_{12}) \quad : \quad fNS \lhd \{RN2, MICGw, h(GwEUI_{\mathsf{Gw_i}})\}_{Gw_i \xleftrightarrow{GrpKey_{\mathsf{Gprld}}} fNS}$$

13. By using S_{11}, S_{12}, and message meaning rule, we obtain:

$$(S_{13}) : fNS \mathrel{|}\equiv GWi \mathrel{|}\sim (RN2, MICGw, h(GwEUI_{\mathsf{Gw_i}}))$$

14. Using S_{13}, A_2 and A_4 with the nonce verification rule

$$(S_{14}) \quad : \quad GWi \quad \mathrel{|}\equiv \quad fNS \quad \mathrel{|}\equiv \quad (RN2, MICGw, h(GwEUI_{\mathsf{Gw_i}}))$$

15. By using the validation of the returned RN2, MICGw

$$(S_{15}) : fNS \mathrel{|}\equiv GWi \mathrel{|}\equiv Gw_i \xleftrightarrow{GrpKey_{\mathsf{Gprld}}} fNS$$

$$(\mathbf{Goal4})$$

Scyther tool

To formally validate the security of a protocol, there are several tools like AVISPA, Proverif, Tamarin and Scyther. We chose Scyther as it is good in verifying multi-protocol attacks, it also verifies protocols by bounded and unbounded number of sesions, it uses SPDL language, it also assumes that every protocol runs in the same network as described in [66]. Scyther is a tool that performs formal security analysis of protocols considering the assumption of perfect cryptography. It means that the adversary cannot learn from an encrypted message unless he possesses the key for decryption [67]. According to the authors, this tool helps finding problems when building protocols. This tool uses Security Protocol Description Language (SPDL), which has a programming syntax similar to C or Java.

45

Scyther is able to evaluate security properties such as; i. *Aliveness* ensures that partners are live, ii. *Weakagree* assures that a partner is communicating with each other rather than an intruder, iii. *Niagree* which means that the parties shall agree on the value of variables after a protocol has been executed, iv. *Nisynch* validates that everything is executed by triggers, occurs in order and contents are preserved, v. *SKR* refers to secrecy of session keys, vi. *Secret* refers to the secrecy of a particular parameter as stated in [67].

Scyther is developed over python and has a Graphic User Interface (GUI) and CLI interface, both of them can be used to analyze protocols and show claims. The results shown in tables 4.2 and 4.3 are taken from the GUI and are able to show a "Failed" statement in the "Status" column when there is a security issue and will display all attacks found with the help of a button that will launch a new window containing a graphic that denotes the attack. On the other hand, if everything goes well the "Status" column will show an "OK" statement combined with the "No attacks" words meaning that there are no attacks that affect the analyzed claim.

In terms of data types, Scyther is flexible and any type could be defined in order to represent a variable. It is important to clarify that Scyther does not analyze data types, it views the state of security of the whole protocol rather than checking for robustness of keys or algorithms used.

From the literature review, Scyther has been used to perform a formal analysis of the security protocols of LoRaWAN as described in [20]. In that work, authors prove the security of the OTAA Join-Procedure process by designing the protocol from scratch according to the specification. The results

obtained showed that V1.0 is susceptible to attacks as there is a weak relation between the End-Node and the NS/AS particular during the join process. The tests performed were focused on Non-injective agreement and Non-injective synchronization.

To perform the analysis of the proposed protocols, we took the designs established in the previous section and translated them to the SPDL language following all the steps designed.

Analysis of Gateway Registration Protocol

First of all, this protocol was coded including all variables as described in Figure 3.3. For this scenario, the Gw_i is authenticated against NS. Then NS, calculates a group key that includes all previous symmetric keys received from other gateways that have been registered already. Every time a gateway arrives or leaves this $GrpKey_{GrpId}$ is recalculated. The Scyther analysis of this protocol is shown in Table 4.2. The secrecy of $GrpKey_{GrpId}$ remains intact by showing no attacks, likewise all claims (Alive, Weakagree, Niagree, Nisynch) are marked with status *OK* showing that no attacks are possible. This validates that the proposed protocol is secure.

Table 4.2: Scyther Results for Proposed Protocols I

Protocol	Role	Claim	Status	Attack patterns
Gateway Registration Protocol	Gateway	Alive	*OK*	No attacks
		Weakagree	*OK*	No attacks
		Niagree	*OK*	No attacks
		Nisynch	*OK*	No attacks
		Secret $GrpKey_{GrpId}$	*OK*	No attacks
	Network Server	Alive	*OK*	No attacks
		Weakagree	*OK*	No attacks
		Niagree	*OK*	No attacks
		Nisynch	*OK*	No attacks
		Secret $GrpKey_{GrpId}$	*OK*	No attacks
Gateway Session Key Derivation Protocol	End-Node	Alive	*OK*	No attacks
		Weakagree	*OK*	No attacks
		Niagree	*OK*	No attacks
		Nisynch	*OK*	No attacks
		SKR $GwSkey_{EN_i}$	*OK*	No attacks
	Gateway	Alive	*OK*	No attacks
		Weakagree	*OK*	No attacks
		Niagree	*OK*	No attacks
		Nisynch	*OK*	No attacks
		Secret $GrpKey_{GrpId}$	*OK*	No attacks
		Secret $GwSkey_{EN_i}$	*OK*	No attacks
	Network Server	Alive	*OK*	No attacks
		Weakagree	*OK*	No attacks
		Niagree	*OK*	No attacks
		Nisynch	*OK*	No attacks
		Secret $GrpKey_{GrpId}$	*OK*	No attacks
		Secret $GwSkey_{EN_i}$	*OK*	No attacks
	Join Server	Alive	*OK*	No attacks
		Weakagree	*OK*	No attacks
		Niagree	*OK*	No attacks
		Nisynch	*OK*	No attacks
		Secret $GwSkey_{EN_i}$	*OK*	No attacks

Table 4.3: Scyther Results for Proposed Protocols II

Protocol	Role	Claim	Status	Attack patterns
UMOAEG Protocol	End-Node	Alive	*OK*	No attacks
		Weakagree	*OK*	No attacks
		Niagree	*OK*	No attacks
		Nisynch	*OK*	No attacks
		Secret $GwSkey_{EN_i}$	*OK*	No attacks
		Secret *AppSKey*	*OK*	No attacks
	Gateway	Alive	*OK*	No attacks
		Weakagree	*OK*	No attacks
		Niagree	*OK*	No attacks
		Nisynch	*OK*	No attacks
		Secret $GwSkey_{EN_i}$	*OK*	No attacks
		Secret *AppSKey*	*OK*	No attacks
	NS	Alive	*OK*	No attacks
		Weakagree	*OK*	No attacks
		Niagree	*OK*	No attacks
		Nisynch	*OK*	No attacks
		Secret $GwSkey_{EN_i}$	*OK*	No attacks
		Secret *AppSKey*	*OK*	No attacks
	AS	Alive	*OK*	No attacks
		Weakagree	*OK*	No attacks
		Niagree	*OK*	No attacks
		Nisynch	*OK*	No attacks
		Secret $GwSkey_{EN_i}$	*OK*	No attacks
		Secret *AppSKey*	*OK*	No attacks
UMOUAEG Protocol	End-Node	Alive	*OK*	No attacks
		Weakagree	*OK*	No attacks
		Niagree	*OK*	No attacks
		Nisynch	*OK*	No attacks
		SKR *AppSKey*	*OK*	No attacks
		SKR $GwSkey_{EN_i}$	*OK*	No attacks
	Gateway	Alive	*OK*	No attacks
		Weakagree	*OK*	No attacks
		Niagree	*OK*	No attacks
		Nisynch	*OK*	No attacks
		Secret $GrpKey_{GrpId}$	*OK*	No attacks
		Secret $GwSkey_{EN_i}$	*OK*	No attacks
	Network Server	Alive	*OK*	No attacks
		Weakagree	*OK*	No attacks
		Niagree	*OK*	No attacks
		Nisynch	*OK*	No attacks
		Secret $GrpKey_{GrpId}$	*OK*	No attacks
		Secret $GwSkey_{EN_i}$	*OK*	No attacks
	Join Server	Alive	*OK*	No attacks
		Weakagree	*OK*	No attacks
		Niagree	*OK*	No attacks
		Nisynch	*OK*	No attacks
		Secret *AppSKey*	*OK*	No attacks
		Secret $GwSkey_{EN_i}$	*OK*	No attacks

Analysis of Gateway Session Key Derivation Protocol

The Gateway Session Key Derivation Protocol was coded by including all variables involved during the OTAA procedure as described in the specification of LoRaWAN 1.1. This protocol is composed of three main roles: End-Node (Dev), Gateway, Network Server and Join Server. The results shown that Alive, Weakagree, Niagree and Nisynch are OK and are not susceptible to attacks. In addition, the secrecy of all the sessions keys generated is preserved among all the roles. The new introduced key **$GwSKey_{ENi}$** represented by $aes\_cmac(NAKey_{EN_i}, h(DevNonce_{ENi}||JoinNonce_{ENi}||JoinEUI_{ENi}))$ shows no attacks, meaning that the OTAA Join Procedure is not feasible to other attacks due to its inclusion. It is important to mention that this new session key preservers the same length (16 bytes) as other derived keys during Join Procedure.

This new key is shared to the Gateway through the Network Server, which uses a pre calculated group key ($GrpKey_{GrpId}$) to multicast this session key ($GwSkey_{EN_i}$) to other gateways that are connected to the same Network Server. Therefore, only the gateway or its group of gateways that belong to the same Network Server can decrypt messages sent by EN_i and then forward payloads to the backend infrastructure. The results of Scyther execution are displayed in Table 4.2.

Analysis of protocol UMOAEG

According to the LoRaWAN 1.1 specification once the Join Procedure has performed and End-Node device would be able to send data to the Application Server. As stated before, the design was translated to a SPDL file to reflect all interactions between the involved roles. In this case the participants were: End-Node (Dev), Gateway, Network Server (NS) and Application Server (AS).

All the variables used in the protocol where declared as String for testing purposes. String was defined as a userType variable as it is not a common data type of Scyther.

As shown in, Table 4.2 there are no potential vulnerabilities in the proposed protocol. It means that as long as an End-Node uses a valid *GwSkey*, a message will be delivered to the Application Server, otherwise, it will be discarded by the Gateway before sending it to the backend infrastructure.

The secrecy of session keys is preserved according to the results shown by Scyther as well as MIC and MIC_P_{EN_i} validation fields to protect the FRMPayload from bit-flipping attacks.

Analysis of protocol UMOUEG

In our proposed scenario, we have identified that if an EN_i has gone through a Join procedure using a different *Gateway.* It is possible to re-generate the $GwSkey_{EN_i}$ and pass it to the Gateway so that it could deliver messages to the back-end infrastructure no matter if this is a newly authenticated Gateway over the platform. In case a rogue gateway aims to forward a message to the AS, it will not be able to determine the $GrpKey_{GrpId}$ required to forward the payload to the NS /fNS .

For this scenario, there are five roles participating in the communication Dev, Gateway, NS, Join Server(JS) and Application Server (AS). Each of them is in charge of encrypting/decrypting particular parts of the message.

The results displayed by Scyther (see Table 4.3) showed that the implementation does not have potential attacks and it could be considered as a secure protocol. All claims are marked with the *OK* word and the *Verified Niagree, Nisynch*, *Alive, Weekagree* and session keys.

4.1.2 Informal Analysis

This part examines the security of the proposed set of protocols by reviewing possible attacks [59].

Man in the middle attack. This attack is not possible as the uplink messages dispatched are using secure encryption functions. When EN_i sends a message to Gw_i, it uses the symmetric session key ($GwSKey$) derived during the Join-Procedure. And when the Gw_i wants to send a message to fNS_i, it uses the symmetric key $GrpKey_{GrpId}$. Likewise, when fNS_i wants to communicate with Gw_i, it uses the calculated symmetric key $GrpKey_{GrpId}$. Using secure encryption functions, let proposed protocols to maintain confidentiality and integrity of messages.

Replay attack. During the gateway registration protocol phase, random nonces are used to avoid replay attacks. Even if the attacker grabs the random nonce, he needs to have gateway credentials to perform a full registration procedure. Also, the attacker will not be able to generate valid messages to the gateway as the session key used to cipher it is calculated during Join and Rejoin procedures respectively.

Password guessing attack. PW_{Ui} is not stored and is only known by the user in charge of performing registration procedure. A variant of it this value $h(PW_{Ui})$ is used to validate a user. It is important to consider that $h(.)$ is a one way has function that cannot reversed to obtain original credentials.

Privileged-insider attack. In the proposed solution, the network administrator (U_i) only have credentials for registering gateways and could not be able to capture other credentials because they are transmitted with a one-way hash function $h(PW_{Ui})$.

Brute force attack. The attacker might try to decrypt the uplink message generated by an end-node. However, the message is protected by symmetric key of 128-bit length that could be changed on demand.

Separation of responsibilities. A gateway (Gw) will only handle a pre-calculated temporary root key ($GwSKa$) and every end-node session key ($GwSKey_{ENi}$). A gateway will not be able to derive $GwSKey_{ENi}$ as it does not store parameters for such purpose.

4.1.3 Cryptographic Operations

In order to determine a potential performance affectation, it is important to analyze and identify the number of additional cryptographical operations that will take place with the current proposed solution. This cryptographical operations comprise hashing, simple XOR, symmetric encryption, symmetric decryption, asymmetric encryption, and asymmetric decryption.

First of all, the current operation of the protocol already includes some cryptographic operations according to the specification [22] that are listed in the table below. The considered operations were taken from the Join-Procedure activation and the Uplink message delivery. Table 4.4 contains the operation name, the number of cryptographic operations, the entity that performs such operation and the phase where that operation takes place (Join-Procedure or UplinkMSG delivery). For the analysis it is important not to overload the End-Node as it has limited computational and power resources.

Table 4.4: LoRaWAN cryptographic operations

LoRaWAN Operation	Entity	Cryptographic Operations						
		XOR	Hash	CMAC	SEnc	SDec	AEnc	ADec
Session key derivation	EN	-	-	-	5	-	-	-
	JS	-	-	-	5	-	-	-
Uplink message	EN	-	-	3	2	-	-	-
	NS	-	-	2	-	-	-	-
	AS	-	-	-	-	1	-	-

The following table 4.5 shows the total number of additional cryptographic operations to be executed by every entity considering the new protocols proposed.

The following Figure 4.1 provides a summary of the additional effort to be made by all participant entities to implement the protocols of the proposed solution. According to the results shown in table 4.5 and figure 4.1, there are more encryption and decryption functions to be executed; however, none of them belong to the end-node. As mentioned before, the End-Node should not be overloaded as that is the entity with the lowest computational capacity, the other devices provide more computational resources so that the inclusion of new

Table 4.5: Table Cryptographic operations of the proposed solutions

Proposed Protocols	Entity	Cryptographic Operations						
		XOR	Hash	CMAC	SEnc	SDec	AEnc	ADec
Gateway Registration	*Gw*	1	4	1	2	1	-	-
	fNS	1	5	1	1	2	-	-
GwSKey Derivation	*JS*	1	2	1	-	-	1	-
	fNS	-	-	-	1	-	-	1
	Gw	-	1	1	-	-	-	-
	EN	1	1	1	-	-	-	-
UMOAEG	*EN*	-	1	2	-	-	-	-
	Gw	-	-	1	1	-	-	-
	fNS	-	-	-	-	1	-	-
	AS	-	-	1	-	-	-	-
UMOUEG	*EN*	-	1	2	-	-	-	-
	GW	-	-	1	2	1	-	-
	fNS	-	-	-	1	2	1	1
	JS	1	1	1	-	-	1	1
	AS	-	-	1	-	-	-	-

cryptographic functions would not affect its overall performance. Devices like the gateway are able to run over robust devices. The whole back-end infrastructure (Join-Server, Application Server and Newtork Server) are able to run over servers, virtual servers or containers in cloud infrastructures.

To have a better understanding on the impact over the End-Node the following Figures 4.2, 4.3 show a comparison of the proposed solution with current LoRaWAN version in terms of cryptographic operations during the Session Key Derivation (Join-Procedure) and Uplink Message Delivery. In blue are all the new cryptographic functions added by the proposed solution whilst in orange are the current LoRaWAN cryptography operations. As showed one new type of operation is XOR. Also, another operation that comes from our proposal is hashing. CMAC operations refer to actions for calculating Message Integrity codes to guarantee message integrity. The proposed solution does not add new decryption functions or asymmetric operations.
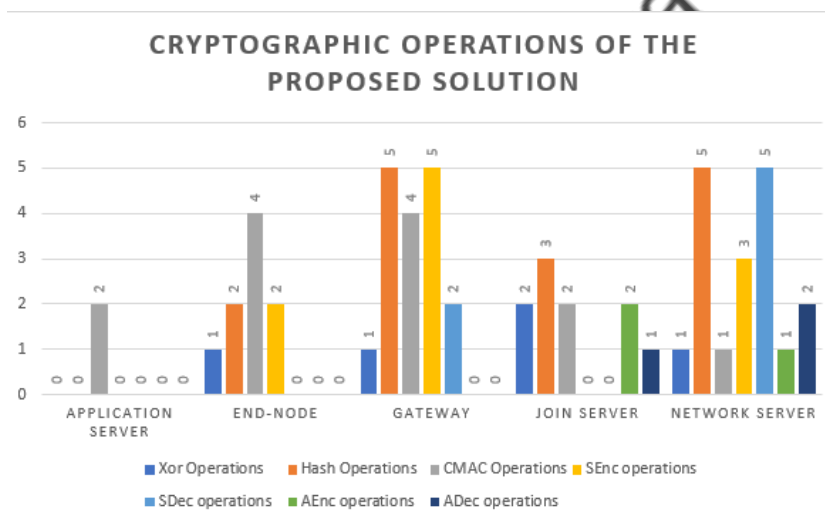
Figure 4.1: Cryptographic operations of the proposed solution per role

The proposed protocols do not aim to implement new symmetric algorithms or to increase their encryption level (i.e. changing to AES-256). The solution is tied to the specification and although it will perform more cryptographic operations, their complexity will remain which means that the current computational resources would be enough to process the new operations.

4.2 Prototype performance evaluation

As part of the validation that is to be performed according to DS methodology. We conduct some experiments to validate the effects in terms of computing performance(RAM and CPU) and power consumption performance over LoRaWAN end-node devices. To achieve so, we
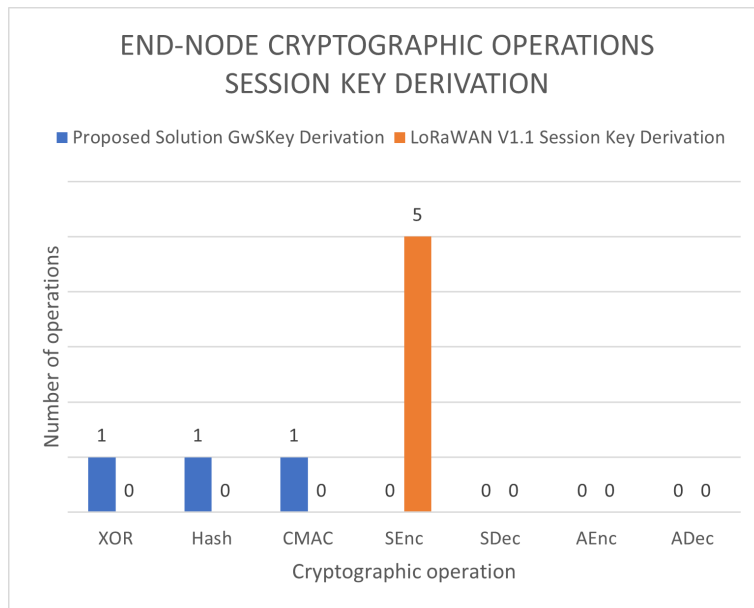
Figure 4.2: Cryptographic operations for End-Node Session Key Derivation
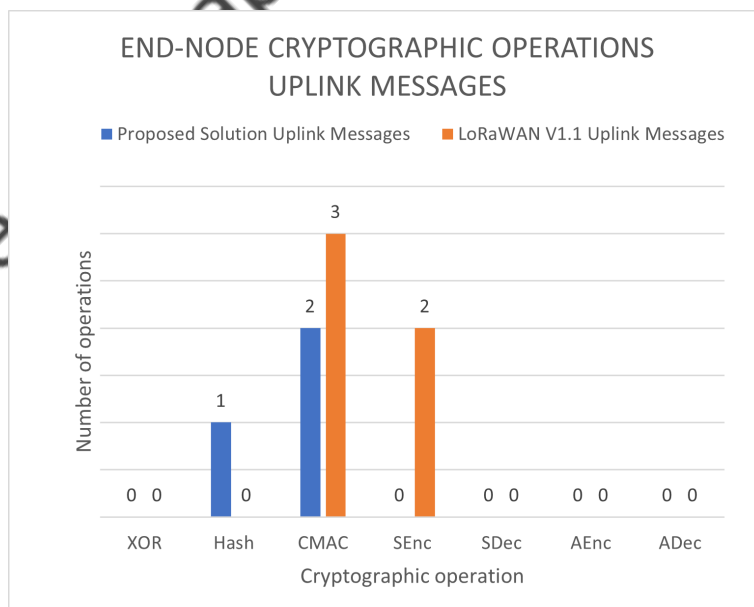
Figure 4.3: Cryptographic operations for End-Node on Uplink Message Delivery

have designed an architecture based on LoRaWAN (see figure 4.6).

We will perform the following steps (see figure 4.4) to conduct our expriments as based in [68].
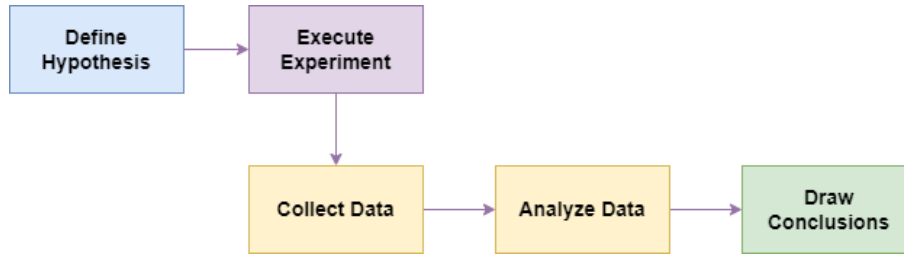

Figure 4.4: Experiment Stages based on Scientific Method

4.2.1 Hypothesis Definition

Research question: Does a lightweight secure communication protocol allows gateways to be authenticated over a LoRaWAN affects performance and power consumption over end-node devices?

Null Hypothesis (H_0): Gateway authentication protocol does not affect performance and power consumption in end-node devices.

Alternative Hypothesis (H_a): Gateway authentication protocol affects performance and power consumption in end-node devices.

Variables to be analyzed

- RAM measured in bytes (B).

- CPU measured in Hertz (Hz).

- Current consumption measured in Mili amperes (mA).

- Battery Voltage measured in Volts (V).

- Time to generate packets measured in seconds (s).

4.2.2 Experiment Setup and Execution

In this stage, we will conduct the experiments and take measurments related to performance in terms of power and computational resources. The proposed solution mainly introduces changes over two stages of LoRaWAN communication. First of all, during the activation process a third key is derived and is therefore required to measure the impact of such task as the end-node will be dealing with it. The other phase is uplink messages where messages

properly encrypted are meant to be delivered to LoRaWAN backend infrastructure. In this phase, we have enhanced the security to protect the message between End-Nodes and Gateway, there cryptographic operations that will take place in the end-node. In summary, the following scenarios will be considered to conduct the experiments:

- Over the air activation (OTAA)

- Uplink Messages

During the execution of the experiments we will use predefined payloads combined with information of power consumption and data of the sensor connected to the end-node as shown in figure 4.5. We tested our solution under US915 which is the licensed frequency for this contry, with the following Spread factor(SF) configuration and Bandwidth as we have estimated in [69] that our solution might not be supported in all configurations. The following table 4.6 shows the supported transmision configurations.

Figure 4.5: Arduino Device Log

Experiment assumptions and constraints

In order to conduct the experiments, there are some assumptions / restrictions that are in place due to factors like geographic location, technology limitations, hardware features, among others. These assumptions / constraints are documented below.

Table 4.6: Supported frequency plans for uplink *EN_i* messages under US915 Band

Spread Factor (SF)	Bandwidth in kHz	Status	Tested
SF7	125	Supported	✓
SF8	125	Supported	✓
SF9	125	Supported	✓
SF10	125	Not Supported in our solution	-
SF11	125	Not supported by US915 region	-
SF12	125	Not supported by US915 region	-
SF7	500	Supported	-
SF8	500	Supported	-
SF9	500	Supported	-
SF10	500	Supported	-
SF11	500	Supported	-
SF12	500	Not supported in our solution	-

- **Device Classes:** *EN_i* must be configured to work in Class A mode, as this approach does not consider yet downlink messages (Classes B and C *EN_i*).

- **Frequency Band:** Due to technology limitations, compatibility and geographic location *EN_i* devices only support US915 frequency (902-928 MHz). From 902.3 MHz to 914.9 MHz with 64 channels and 200 KHz spacing for uplink messages, 923.3 MHz to 927.5 MHz with 8 channels for downlink messages with 600 KHz spacing, and 903.3 MHz to 914.2 MHz with 1.6 MHz spacing.

- **Gateway features:** Due to technology limitations, gateway is not able to be deployed outdoors for collecting information outside lab perimeter.

Testbed architecture componentes

In this section the following architecture (see figure 4.6) has been designed and deployed to perform experiments for the two scenarios described in section 4.2.2. This architecture is based on LoRaWAN specification [22, 34]. We have developed and modified our version of Chirpstack to support new security features proposed in this approach to authenticate Gateways. Changes has been made over Appplication Server, Network Server, Join Server, Chirpstack API, Chirpstack Bridge.

The following tables 4.7 and 4.8 describe the elements used to build the architecture described in previous section. Table 4.7 describes the hardware used for deploying arduino sketch code, to support both types two versions of the sketch have been created as ESP32 devices use libraries that are not fully compatible with Arduino devices.
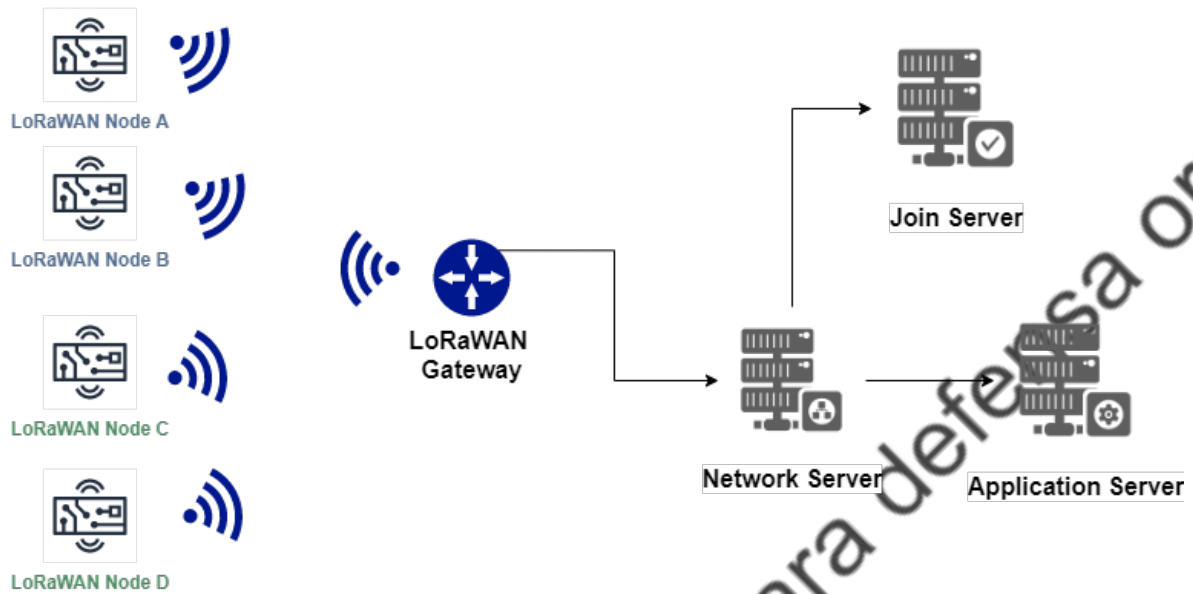
Figure 4.6: Testbed Architecture

Role	Hardware	Sensors/Add-ons
Gateway	Raspberry Pi 3 B+	RAK 2245 Ver A. Pi Hat
End-node A	Heltech ESP 32 Wifi LoRa	Photosensitive Sensor
End-node B	Heltech ESP 32 Wifi LoRa V2	Photosensitive Sensor
End-node C	Arduino UNO Elego	Photosensitive Sensor
End-node D	Arduino UNO R3	Photosensitive Sensor

Table 4.7: Devices, roles and sensors used

Role	Software
Network Server	Chirpstack OS - Network Server
Application Server	Chirpstack OS - Application Server
Gateway Bridger	Chirpstack OS - Gateway Bridge

Table 4.8: Backend Roles and software tools

4.2.3 Data Collection

The following code 1 is used to collect statistics from End-Nodes, this code registers information about CPU usage, RAM usage and Battery Voltage over Arduino IDE Serial Monitor. The output obtained is later downloaded to an excel file to be processed for further analysis. The table shown in figure 4.7 was used to process information collected from log files. This table has 3 columns: i) *Original Message*, corresponds to the original message captured from the Arduino IDE Serial Monitor. ii) *Time* column, extracts the time when the message was captured. iii) *Event*, collects all information and metrics generated by the EN_i whilst sending packages. The following diagram describe steps followed to collect data.

A Original Message	B Time	C Event
12:05:00.825 -> 438563231: engineUpdate, opmode=0x908	12:05:00.825	-> 438563231: engineUpdate, opmode=0x908
12:05:00.825 -> 438563371: EV_TXSTART	12:05:00.825	-> 438563371: EV_TXSTART
12:05:00.825 -> 903900000	12:05:00.825	-> 903900000
12:05:00.825 -> 3	12:05:00.825	-> 3
12:05:00.825 -> 1	12:05:00.825	-> 1
12:05:00.825 -> Freq: 903900000, DevNonce: 4, SF: 3	12:05:00.825	-> Freq: 903900000, DevNonce: 4, SF: 3
12:05:00.825 -> 4095.00V 1567 LUM	12:05:00.825	-> 4095.00V 1567 LUM
12:05:00.825 -> 438564133: TXMODE, freq=903900000, len=8(	12:05:00.825	-> 438564133: TXMODE, freq=903900000, len=
12:05:00.825 -> LoRaWAN 1.1 PhD SF7 ->4095.00V 1467 LUM	12:05:00.825	-> LoRaWAN 1.1 PhD SF7 ->4095.00V 1467 LUM
12:05:00.825 -> Packet queued	12:05:00.825	-> Packet queued
12:05:01.975 -> 438635507: setupRx1 txrxFlags 0x20 --> 01	12:05:01.975	-> 438635507: setupRx1 txrxFlags 0x20 --> 01
12:05:01.975 -> start single rx: now-rxtime: 4	12:05:01.975	-> start single rx: now-rxtime: 4
12:05:01.975 -> 438636140: RXMODE_SINGLE, freq=92330000	12:05:01.975	-> 438636140: RXMODE_SINGLE, freq=9233000(
12:05:01.975 -> rxtimeout: entry: 438636367 rxtime: 4386361	12:05:01.975	-> rxtimeout: entry: 438636367 rxtime: 43863(
12:05:02.957 -> 438698007: setupRx2 txrxFlags 0x1 --> 02	12:05:02.957	-> 438698007: setupRx2 txrxFlags 0x1 --> 02
12:05:02.957 -> start single rx: now-rxtime: 4	12:05:02.957	-> start single rx: now-rxtime: 4

Figure 4.7: Data table collection template

To collect information about milli amperes consumed, we will be using a multimeter connected in series as shown in figure 4.8. Device information will be collected for 5 minutes.

Security Keys

During the process of data collection, we verify that keys are derived according to our protocol design and specification. This section compiles that are used to authenticate devices and enhance security between EN_i <-> Gw_i <-> NS . Besides pointing out derived keys, we also determine the time taken to perform such tak measured in seconds (s).

Gateway Session Key Derivation

The following figures 4.9 and 4.10, shows the results of introducing the new key for the End-Node to secure communication between EN_i <-> Gw_i .
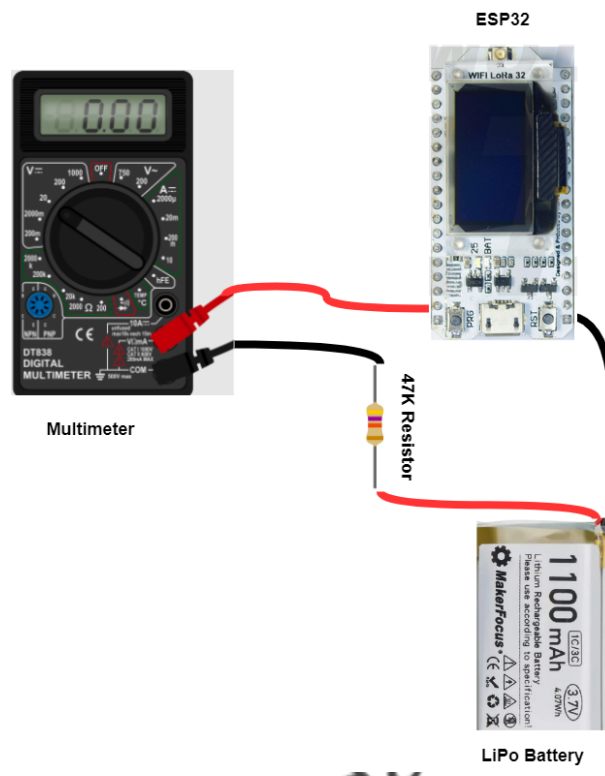
Figure 4.8: Current measurement with multimeter

```
1    void getDeviceStatistics(){
2        //Hardware Statistics
3        long T0 = millis();
4        uint32_t startCycle = ESP.getCycleCount();
5        while (millis() < T0 + 1000) {}
6        uint32_t endCycle = ESP.getCycleCount();
7        Serial.printf("CPU Usage Hz: %u\n", endCycle - startCycle);
8        startCycle = ESP.getCycleCount();
9        delay(1000);
10       endCycle = ESP.getCycleCount();
11       Serial.printf("CPU Usage Hz: %u\n", ((endCycle - startCycle)));
12       Serial.println("Heap Free Bytes: "+String(ESP.getFreeHeap()));
13
14       //Power Consumption Statistics (Volts)
15       Serial.println("Battery : "+String(ReadVoltage(voltagePin))+" V");
16
17       //Sensor Data
18       Serial.println("Photo Sensor : "+GetSensorData(sensorPin));
19   }
```

Code 1: IoT Device Data Collection code

Figure 4.9: Gateway Session Key (GwSKey) derivation process in IoT device.

Figure 4.10: Gateway Session Key (GwSKey) derivation process in server side (Chirpstack infrastructure).

In figure 4.9 highlighted in yellow appears the new derived key. In light a message to define the star time when the device built the Join Request and sent to the server. In green, the picture shows the final time when the Join Accept was received and then derived GwSKey. It took approximately 5.08 seconds (s). From previous tests we have identified that there is a mean time of 5.084 seconds (s) to derive this new key compared with the previous LoRaWAN specification.

4.2.4 Data Analysis

4.2.5 Scenario 1 - OTAA Activation

For this scenario we have not configured specific bandwidth to deliver Join Request messages as this is handled internally by the End-Node.

LoRaWAN 1.1

The time taken to build every Join request is denoted in figure 4.11.
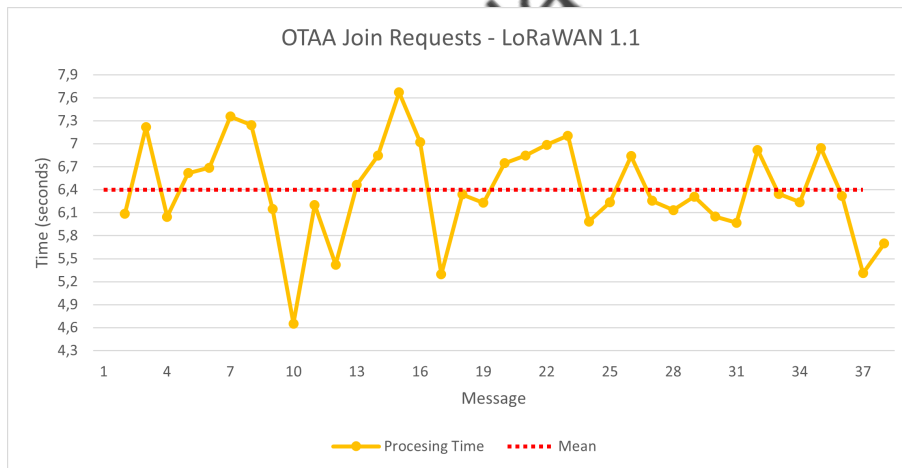
Figure 4.11: OTAA Processing Time

The following figure 4.11 denotes power consumption measured in mili amperes (mA) for the generation of Join Requests during Over the Air Activation Procedure.

LoRaWAN Enhanced

Figure 4.13 shows the time taken for a Join Request Message to be generated. On the other hand, figure 4.14 shows power consumption measurements in terms of milli amperes(mA) and Battery Voltage (V).

Figure 4.15 shows a comparision of processing times between the two versions. In this figure, it can be noted that processing times are greater for the LoRaWAN enhanced version,
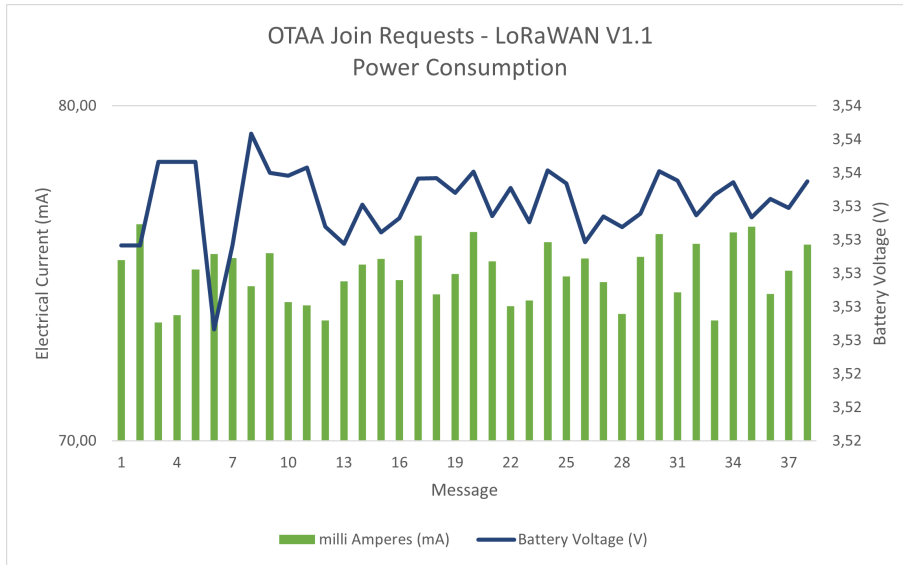
Figure 4.12: OTAA Power Consumption
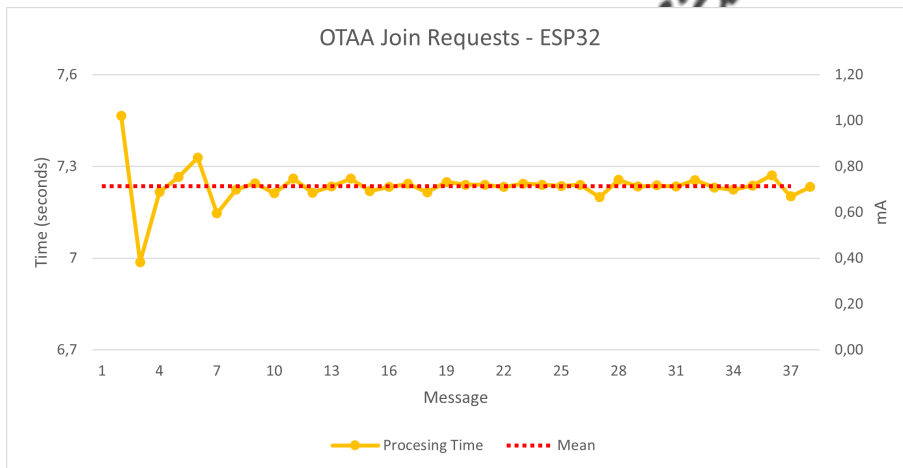
Figure 4.13: OTAA Processing Time
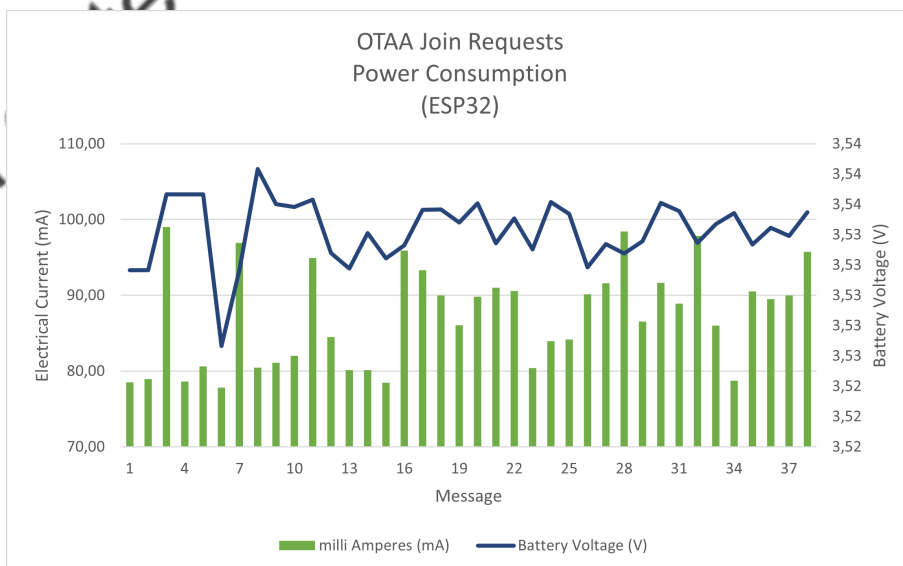
Figure 4.14: OTAA Power Consumption

it took around 1 more second to generate a Join Request compared to the Original OTAA activation procedure of LoRaWAN version 1.1.
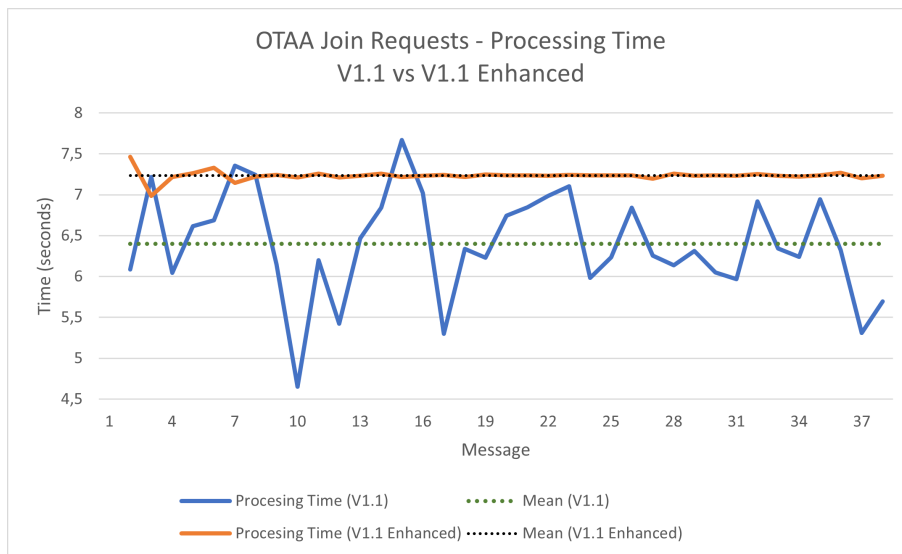
Figure 4.15: OTAA Processing Time Comparison V1.1 vs V1.1 Enhanced

4.2.6 Scenario 2 - Uplink Messages

LoRaWAN 1.1

The following figures denote power consumption 4.16 and CPU usage 4.17 for LoRaWAN version 1.1. The next figure shows power consumption measured in mA according to different frequency configurations.

This figure 4.16 denotes the usage of CPU when generating uplink messages to the LoRaWAN backend infrastructure. The next table 4.9 shows a summary of the satistics collected in terms of power consumption of the LoRaWAN 1.1 version of the procotol.

Frequency	Mean (mA)	Min (mA)	Max (mA)	Average (mA)
SF7 - BW125	62,12	58,53	66,47	62,33
SF8 - BW125	64,37	60,40	70,16	65,06
SF9 - BW125	75,04	73,53	76,47	75,06

Table 4.9: Uplink Messages statistics for LoRaWAN Enhanced version

CPU usage is measured in percentage as denoted in the following figure 4.17.

In table 4.10, it can be seen that the average CPU usage for the three configurations bandiwdths is 0.04%.

In the previous figures, it can be seen that the SF9 configuration demands more power consumption and CPU usage.
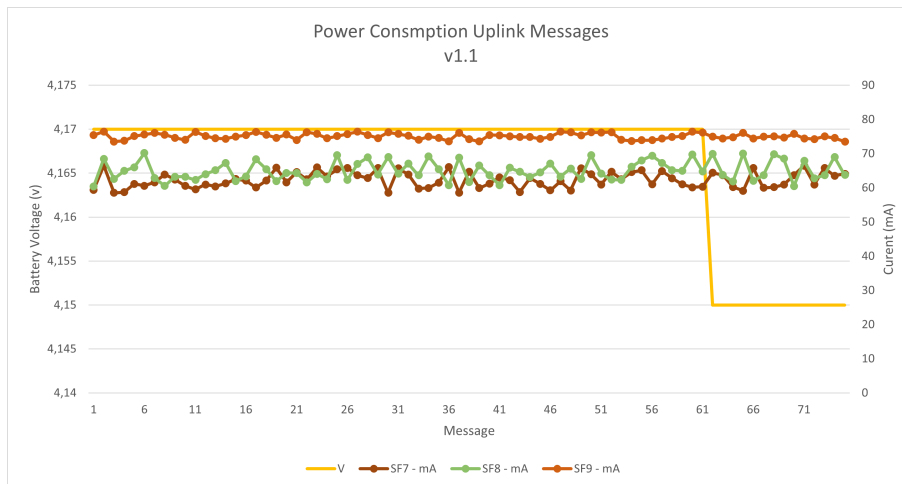
Figure 4.16: Power Consumption to send uplink messages in LoRaWAN 1.1 enhanced version.
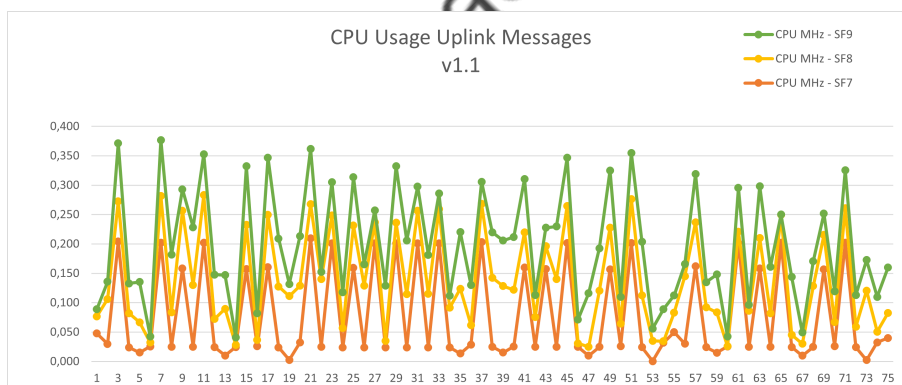
Figure 4.17: CPU Usage to send uplink messages in LoRaWAN 1.1 enhanced version.

Frequency	Mean	Min	Max	Average
SF7 - BW125	0,04	0,04	0,05	0,04
SF8 - BW125	0,04	0,03	0,04	0,04
SF9 - BW125	0,04	0,01	0,08	0,04

Table 4.10: Uplink Messages CPU usage statistics for LoRaWAN version 1.1

LoRaWAN Enhanced

In figure 4.18 we can see that the proposed solution consumes more power with SF9 (78.57 mA approximately) configuration whilst with SF7 it requires approximately 67,58 mA for transmitting information. We can also identify that Voltage consumption is dropping as battery is discharging, it has reduced from 4.17V to 4,15V while sending around 75 messagews every five seconds. The following table 4.11 shows statistics values of the results obtained.
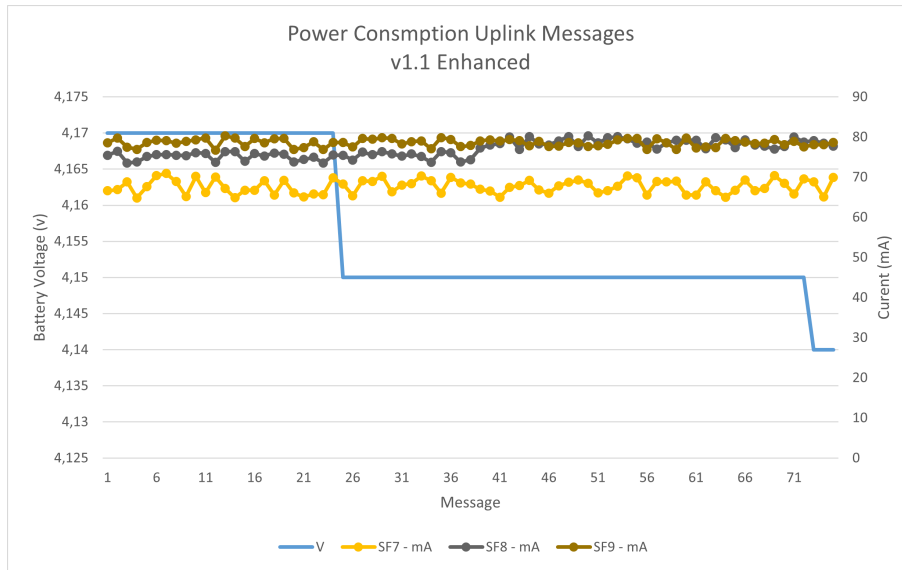
Figure 4.18: Power Consumption to send uplink messages in LoRaWAN 1.1 enhanced version.

Frequency	Mean (mA)	Min (mA)	Max (mA)	Average (mA)
SF7 - BW125	67,41	64,80	70,90	67,58
SF8 - BW125	76,47	73,53	80,23	76,87
SF9 - BW125	78,62	76,71	80,33	78,57

Table 4.11: Uplink Messages statistics for LoRaWAN Enhanced version

The following information (see Figure 4.19 and Table 4.12) shows CPU usage statistics for the LoRaWAN Enhanced version. This values are represented in percentage of usage. According to the figure SF9 is the configuration that demands more computational resources.

Frequency	Mean	Min	Max	Average
SF7 - BW125	0,04	0,04	0,05	0,04
SF8 - BW125	0,08	0,08	0,08	0,08
SF9 - BW125	0,07	0,04	0,09	0,07

Table 4.12: Uplink Messages statistics for LoRaWAN Enhanced version - CPU Usage in usage percent

66
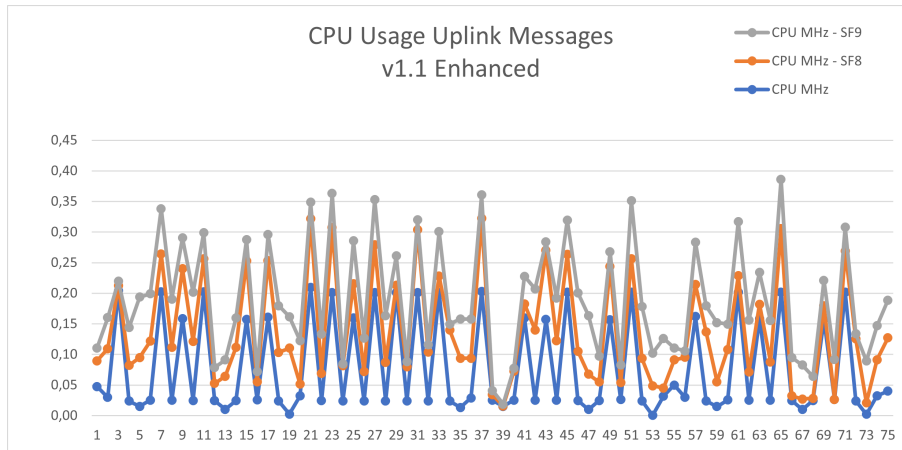
Figure 4.19: CPU Usage to send uplink messages in LoRaWAN 1.1 enhanced version.

RAM Analysis

For LoRaWAN v1.1, the sketch file (program) uses 272304 bytes (20%) of storage, the total amount is 1310720 bytes. Global variables uses 15624 bytes (4%) of dynamic memory, leaving 312056 bytes free for local variables. The maximum size is 327680 bytes.

For LoRaWAN v1.1 enhanced version,the sketch file (program) uses 291472 bytes (22%) of storage, the total amount is 1310720 bytes. Global variables uses 15660 bytes (4%) of dynamic memory, leaving 312056 bytes free for local variables. The maximum size is 327680 bytes.

In terms of RAM memory, for LoRaWAN 1.1 version the solution has a left heap space of 368252 bytes whilst the enhanced version has 368468 bytes left of heap, according to the log.

4.2.7 Hypothesis validation

To validate our hypothesis we have used wilcoxon test. From those results, it was determined that the values obtained are lower than 0.05 which indicates the null hypothesis is rejected and hence alternative hypothesis is accepted by confirming that **Gateway authentication protocol affects performance and power consumption in end-node devices**.

4.3 Discussion

In the Over the Air Activation Scenario, have identified that our approach requires approximately 87.19 mA to build and send Join Requests, and LoRaWAN 1.1 implementation uses 75.03 mA. It means that our solution requires more power to achieve such task (12.16 mA) which represents 16% extra power effort. However, this amount of power does not represent an

issue as it is small and could be satisfied with rechargeable batteries.

From the results compiled in previos section 4.2, it can be seen that our solution demands more power consumption but it is not that siginifcant as shown in table 4.13. We have coded LoRaWAN 1.1 as **"Lv11"** and then enhanced version as **"LEv11"** to make table columns smaller. According to the results obtained there would be an extra effort of around 18% to send uplink packets over our solution compared to version 1.1. However, this does not represent an issue for the use of our solution as the bandwidth configuration depends on the size of the payload being sent. For our testing we decided to validate our solution through several scenarios.

Frequency	Lv11 (mA)	LEv11 (mA)	Difference (mA)	Extra Power Required
SF7 - BW125	62,33	67,58	5,25	8,42%
SF8 - BW125	65,06	76,87	11,81	18,15%
SF9 - BW125	75,06	78,57	3,51	4,67%

Table 4.13: Uplink Messages power consumption for different bandwidth configurations

The following table 4.14 shows the a summary of the results obtained in terms of CPU usage for the diffente bandwidth configurations. The value of the columns is expressed in percentage. This table also shows a considerable demand of hardware resources (CPU usage). However, the demand of resources is not quite considerable as the increase is less than 1% of usage. Therefore, the proposed approach will not affect the performance of the devices.

Frequency	Lv11	LEv11	Difference	Extra CPU Required
SF7 - BW125	0,04	0,04	0,00	0,00%
SF8 - BW125	0,04	0,08	0,04	122,55%
SF9 - BW125	0,04	0,07	0,03	56,32

Table 4.14: Uplink Messages CPU usage for different bandwidth configurations

On the other hand, in terms of RAM and storage our proposed approach demands an extra 2% of storage. This, does not represent an issue for deploying this solution in similar devices with limited resources.

From the results obtained it can be said that demands more power and computational resources. However, as it has been described the required amounts can be handled with the inclusion of batteries with higher capacity. For instance, with such amount of constant consumption, assuming that we will have a 1100mA battery, we could be able to send messages for around 16 hours to drain the battery totally, This scenario implies that the device will be working all day sending messages every 5 seconds. However, this is not a use case for LoRaWAN as Class A devices are intended to sense information in large periods of time. As mentioned before, this approach is not intended yet to Class B or C devices as they are more power demanding and might need an unlimited power source to work efficiently.

Chapter 5

CONCLUSIONS AND FUTURE WORKS

This chapter presents the conclusions as well as future works of the thesis. Section 5.1 presents the conclusions while Section 5.2 describes limitations and future works concerning LoRaWAN networks.

5.1 Conclusions

In this work, we have proposed a secure and lightweight protocol to address gateway authentication over LoRaWAN infrastructure. Within IoT architectures, authentication is a major security concern. In most cases, it is mostly oriented to End-Devices as they are information producers; however, there are other devices like gateways that are assumed to be "trusted". Authentication task is performed by including third-party elements (like CA or Blockchain) or is an implementation that is part of a specific piece of software. This turns out in several types of deployments that might lead to compromise architectures by opening more vulnerabilities. Gateway authentication is a feature that is out of the LoRaWAN scope, and hence this has led to empirically make the best effort to address such issue. Our solution provides a design that takes advantage of LoRaWAN architecture without including further components or roles.

The literature points out several types of vulnerabilities as shown in Table 2.2. Most of the research have focused in improving LoRaWAN v1.0.x. and have not deeply analyzed v1.1 as it seems to fix most security vulnerabilities reported in versions 1.0.x. However, this assumption has not been confirmed since few works analyzing v1.1 have found that are still vulnerabilities to be addressed. It called our attention that during the review of the literature we found that the gateway is an element qualified as "trusted" and within the specification there is no mechanism to ensure that statement. In fact, this lack of assurance led us to

understand that this device, despite being critical, is not authenticated within the LoRaWAN network. Hence, this non-existence of authentication is vaguely or almost nully discussed and analyzed. Only few works discuss this availability issue [15, 21, 46].

Lighweight cryptographic operations provide an appropriate level of security to accomplish tasks like authentication. These operations are oriented to be deployed over devices with few computational resources. Operations like XOR combined with other techniques such as hashing provide an adequate level of protection when they are oriented to protect small amounts of information. Likewise, symmetric encryption is not a consumer of computational resources if there are no huge keys to be calculated. Moreover, CMAC (Cipher-based message authentication codes) are a mechanism to verify the autenticity of a message and are generally based on symmetric cryptography. The combinataion of XOR, hashing, symmetric cryptography and CMAC allowed to design a set of lightweight security protocols to properly handle the lack of gateway authentication over LoRaWAN network. This design can address authentication on two sides: End-Node with gateways, and gateways with LoRaWAN back-end infrastructure (Network Server). The proposed approach, related to other works like [21], does not include new elements that may generate new security breaches.

BAN Logic and Scyther Tool formally verify the level of security of communication protocols. These tools are popular among research community to point out security issues or denote great strength in security aspects. These techniques have allowed to determine that, from a formal security perspective, including new protocols in LoRaWAN infrastructure does not open breaches. Likewise, this set of protocols does not represent a problem of energy and computational consumption for the End-Node. Although the primary target of this approach was gateways, we cannot ignore collateral effects that can be generated on elements with low energy consumption and computational cost (End-Nodes). We have measured RAM, CPU, Time, Battery Voltage and Energy current over an IoT monitoring scenario (light sensing), to evaluate the impact of this solution over the End-Node. The results obtained in section 4.2 have shown that despite requiring additional capacities, this solution is not a high consuming resource one. It generates a small impact but it is not representative and does not represent a threat to the performance of the equipment or its power source. The approach used for this evaluation contemplate IoT devices with limited and similar hardware features as other smart sensing devices. Therefore, these protocols can be implemented over IoT sensing devices and gateways with similar or higher hardware and energy resources.

The major milestone obtained with this research, was the development of LoRaWAN library to support V1.1 [70] available in github [1]. To the best of our knowledge, there are no open and public frameworks that currently handle that version. This library can be deployed over IoT Arduino like devices supporting LoRaWAN network connectivity and are agnostic to the type of application deployed.

---

[1]Available at: https://github.com/JuanSulca/arduino-lmic

5.2 Future Works and perspectives

IoT devices leveraged on LPWAN protocols and using LoRaWAN are growing and predictions state there will be billions in few years. This exponential growth opens greatly involved the participation of industry and academia to develop new applications. However, security issues are a latent concern that is not necessarily addressed to the same extent as the new solutions that appear. This opens the door to face security problems that might compromises confidentialiy, integrity and availability (CIA triad) of information and systems based on this technologies. Therefore, it is necessary to profundize in solutions that enhances authentication and key exchange by reducing human intervention to prevent misconfigurations that might lead to open new vulnerabilities.

One of the limitations that appear in this research is that current protocols are based on cryptographic libraries that would be deprecated and hence providing such support would affect and demand more power consumption. The current source code shall be optimized to adapt to other scenarios where IoT devices would have less hardware or energy resources.

Another limitation found during our research is related to data transmission rates. The devices used, operated over 915Mhz but LoRaWAN can operate over 433Mhz and 868Mhz depending on the country where the test is performed. Although the library is adaptable to any type of frequency, it would be useful to test over such frequencies to analyze the behaviour of the proposed solution under such conditions.

LoRaWAN has several types of configurations, it is important to extend the advantages of the proposed protocols to authenticate downlink messages so that End-Nodes will only be able to receive trusted messages from gateways, for a fully authenticated architecture ensuring end-to-end security.

The inclusion of AI would improve authentication schemes by recognizing other elements of the infrastructure. Also, it would allow to build proactive tools to identify potential attacks like DDoS over this type of technologies. It would allow not only to develop protections over TCP/IP layered protocols but also LPWAN schemes.

Finally, zero-trust is a novel approach that allows to define a policy for accessing resources based on needs to work. The inclusion of initiatives like Zero-Trust over LoRaWAN networks would allow to have a fine granularity model where devices can share specific information with other members of the back-end infrastructure or share information among other End-Node devices.

Bibliography

[1] Y. Qian, "Towards decentralized iot security enhancement: A blockchain approach," *Comput. Electr. Eng*, vol. 72, pp. 266–273, 11 2018.

[2] E. Clarissa, F.-B. Gregor, and H. David, "From facets to a universal definition-an analysis of iot usage in retail," in *14th International Conference on Wirtschaftsinformatik Proceedings*, Siegen, Germany, february 2019.

[3] R. Sinha, Y. Wei, and S.-H. Hwang, "A survey on lpwa technology: Lora and nb-iot," *ICT Express*, vol. 3, pp. 14–21, 03 2017.

[4] S. R. Department, "Iot: number of connected devices worldwide 2012-2025 | statista," p. 11, 11 2016. [Online]. Available: https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/

[5] G. Inc, "Leading the iot," Gartner Inc, Tech. Rep., 2017. [Online]. Available: https://www.gartner.com/imagesrv/books/iot/iotEbook_digital.pdf

[6] J. J. Barriga, J. Sulca, J. L. León, A. Ulloa, D. Portero, R. Andrade, and S. G. Yoo, "Smart parking: A literature review from the technological perspective," *Applied Sciences*, vol. 9, no. 21, 2019. [Online]. Available: https://www.mdpi.com/2076-3417/9/21/4569

[7] J. P. Shanmuga Sundaram, W. Du, and Z. Zhao, "A survey on lora networking: Research problems, current solutions, and open issues," *IEEE Communications Surveys and Tutorials*, vol. 22, no. 1, pp. 371–388, 2020. [Online]. Available: https://ieeexplore.ieee.org/document/8883217

[8] K. Mekki, E. Bajic, F. Chaxel, and F. Meyer, "A comparative study of LPWAN technologies for large-scale IoT deployment," *ICT Express*, vol. 5, no. 1, pp. 1–7, Mar. 2019. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01670379

[9] M. A. Ertürk, M. A. Aydın, M. T. Büyükakkaşlar, and H. Evirgen, "A survey on lorawan architecture, protocol and technologies," *Future Internet*, vol. 11, no. 10, 2019. [Online]. Available: https://www.mdpi.com/1999-5903/11/10/216

[10] R. O. Andrade and S. G. Yoo, "A comprehensive study of the use of lora in the development of smart cities," *Applied Sciences*, vol. 9, no. 22, 2019. [Online]. Available: https://www.mdpi.com/2076-3417/9/22/4753

[11] K. O. Adefemi Alimi, K. Ouahada, A. M. Abu-Mahfouz, and S. Rimer, "A survey on the security of low power wide area networks: Threats, challenges, and potential solutions," *Sensors*, vol. 20, no. 20, 2020. [Online]. Available: https://www.mdpi.com/1424-8220/20/20/5800

[12] J. J. Barriga and G. Yoo S, "Internet of things: A security survey review on long range wide area network (lorawan)," *J. Eng. Appl. Sci*, vol. 14, pp. 9774–9787, 09 2019.

[13] J. Lee, D. Hwang, J. Park, and K.-H. Kim, "Risk analysis and countermeasure for bit-flipping attack in lorawan," *2017 International Conference on Information Networking (ICOIN)*, pp. 549–551, 2017.

[14] J. Kim and J. Song, "A dual key-based activation scheme for secure lorawan," *Wirel. Commun. Mob. Comput*, vol. 2017, 2017.

[15] L. Jae Young, "A study on gateway authentication protocol in iot," *Journal of Convergence for Information Technology*, vol. 7, no. 3, p. 91–96, 2017.

[16] X. Yang, E. Karampatzakis, C. Doerr, and F. Kuipers, "Security vulnerabilities in lorawan," *2018 IEEE/ACM Third International Conference on Internet-of-Things Design and Implementation (IoTDI)*, pp. 129–140, 2018.

[17] I. Butun, N. Pereira, and M. Gidlund, "Analysis of lorawan v1.1 security," *Proceedings of the 4th ACM MobiHoc Workshop on Experiences with the Design and Implementation of Smart Objects -SMARTOBJECTS '18*, pp. 1–6, 2018.

[18] T. Dönmez and E. Nigussie, "Security of lorawan v1.1 in backward compatibility scenarios," *Procedia Computer Science*, vol. 134, pp. 51–58, 2018.

[19] E. Gresak and M. Voznak, "Protecting gateway from abp replay attack on lorawan," in *AETA 2018 - Recent Advances in Electrical Engineering and Related Sciences: Theory and Application*, I. Zelinka, P. Brandstetter, T. Trong Dao, V. Hoang Duy, and S. B. Kim, Eds. Cham: Springer International Publishing, 2018, pp. 400–408.

[20] M. Eldefrawy, I. Butun, N. Pereira, and G. M, "Formal security analysis of lorawan," *Comput. Networks*, vol. 148, pp. 328–339, 01 2019.

[21] A. Mohamed, F. Wang, I. Butun, J. Qadir, R. Lagerström, P. Gastaldo, and D. D. Caviglia, "Enhancing cyber security of lorawan gateways under adversarial attacks," *Sensors*, vol. 22, no. 9, 2022. [Online]. Available: https://www.mdpi.com/1424-8220/22/9/3498

[22] L. Alliance, "Lorawan tm 1.1 specification," p. 97331, 2017. [Online]. Available: https://lora-alliance.org/wp-content/uploads/2020/11/lorawantm_specification_-v1.1.pdf

[23] A. Lora, "Ts2-1.1.0 lorawan backend interfaces specification," p. 85, 2020. [Online]. Available: https://resources.lora-alliance.org/technical-specifications/ts002-1-1-0-lorawan-backend-interfaces

[24] R. H. Alan, T. M. Salvatore, P. Jinsoo, and R. Sudha, "Design science in information systems research," *MIS Quarterly*, vol. 28, no. 1, pp. 75–105, 2004. [Online]. Available: http://www.jstor.org/stable/25148625

[25] A. Hevner and S. Chatterjee, *Design Science Research Frameworks*. Boston, MA: Springer US, 2010, pp. 23–31. [Online]. Available: https://doi.org/10.1007/978-1-4419-5653-8_3

[26] G. A. Abdul, "Cryptonet generic security framework for cloud computing environments," Ph.D. dissertation, School of Information and Communication Technologies , 2011. [Online]. Available: https://www.diva-portal.org/smash/get/diva2:411892/FULLTEXT01.pdf

[27] A. Dennis, R. Jones, D. Kildare, and C. Barclay, "Design science approach to developing and evaluating a national cybersecurity framework for jamaica," *THE ELECTRONIC JOURNAL OF INFORMATION SYSTEMS IN DEVELOPING COUNTRIES*, vol. 62, no. 1, pp. 1–18, 2014. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/j.1681-4835.2014.tb00444.x

[28] M. Steven and C. Dan, "A design science approach to constructing critical infrastructure and communicating cybersecurity risks," *Technology Innovation Management Review*, vol. 5, pp. 6–16, 06/2015 2015. [Online]. Available: http://timreview.ca/article/902

[29] R. Jyri and P. Rauno, "Design science research towards resilient cyber-physical ehealth systems," *FinJeHeW*, vol. 9, p. 203, 2017. [Online]. Available: http://www.psc-europe.eu

[30] A. Alexei, "Implementing design science research method to develop a cyber security framework for heis in moldova," in *The 11th International Conference on Electronics, Communications and Computing*, 03 2021, pp. 228–231.

[31] W. Group, "What makes sigfox so secure for the internet of things," WND Group, Tech. Rep., 09 2017. [Online]. Available: https://www.wndgroup.io/2017/09/18/sigfox-security-internet-things/

[32] L. Alliance, "A technical overview of lora ® and lorawan tm what is it?" San Ramon, CA, 2015. [Online]. Available: https://lora-developers.semtech.com/uploads/documents/files/LoRa_and_LoRaWAN-A_Tech_Overview-Downloadable.pdf

[33] J. J. Barriga, J. Sulca, J. León, A. Ulloa, D. Portero, J. García, and S. G. Yoo, "A smart parking solution architecture based on lorawan and kubernetes," *Applied Sciences*, vol. 10, no. 13, 2020. [Online]. Available: https://www.mdpi.com/2076-3417/10/13/4674

[34] L. Alliance, "Lorawan remote multicast setup specification v1.0.0 | lora alliance tm," 06 2019. [Online]. Available: https://lora-alliance.org/wp-content/uploads/2020/11/remote_multicast_setup_v1.0.0.pdf

[35] S. Iskhakov, R. Meshcheryakov, and A. Iskhakova, "Analysis of vulnerabilities in low-power wide-area networks by example of the lorawan *," *IV International Research Conference "Information Technologies in Science, Management, Social Sphere and Medicine*, vol. 72, pp. 334–338, 2017.

[36] W. Sung, H. Ahn, J. Kim, and S. Choi, "Protecting enddevice from replay attack on lorawan," *20th International Conference on Advanced Communication Technology (ICACT)*, p. 1, 2018.

[37] S. Tomasin, S. Zulian, and L. Vangelista, "Security analysis of lorawan join procedure for internet of things networks," *2017 IEEE Wireless Communications and Networking Conference Workshops, WCNCW 2017*, pp. 1–6, 2017.

[38] S. Naoui, M. Elhdhili, and L. Saidane, "Enhancing the security of the iot lorawan architecture," *2016 International Conference on Performance Evaluation and Modeling in Wired and Wireless Networks (PEMWN)*, pp. 1–7, 2016.

[39] E. Aras, N. Small, G. Ramachandran, S. Delbruel, W. Joosen, and D. Hughes, "Selective jamming of lorawan using commodity hardware," *Proceedings of the 14th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services*, pp. 363–372, 2017.

[40] E. Aras, G. S. Ramachandran, P. Lawrence, and D. Hughes, "Exploring the security vulnerabilities of lora," in *2017 3rd IEEE International Conference on Cybernetics (CYBCONF)*, 2017, pp. 1–6.

[41] S.-I. R, "Enhancing lorawan security through a lightweight and authenticated key management approach," *Sensors*, vol. 18, p. 1833, 06 2018.

[42] G. Avoine and L. Ferreira, "Rescuing lorawan 1.0," *Financial Cryptography and Data Security*, vol. 3, pp. 779–787, 2018.

[43] K. Tsai, Y. Huang, F. Leu, I. You, Y. Huang, and C. Tsai, "Aes-128 based secure low power communication for lorawan iot environments," *IEEE Access*, 2018.

[44] J. Kim and J. Song, "A secure device-to-device link establishment scheme for lorawan," *IEEE Sensors Journal*, vol. 18, pp. 2153–2160, 2018.

[45] Cispa, A. Nasir, H. Qureshi, A. Ashfaq, S. Mumtaz, and J. Rodriguez, "Network intrusion detection system for jamming attack in lorawan join procedure," *2018 IEEE International Conference on Communications (ICC)*, pp. 1–6, 06 2014.

[46] M. Olof, T. Rikard, W. Joakim, and K. Stig Arne, "A survey on attacks and defences on lorawan gateways," *Decision Support Systems and Industrial IoT in Smart Grid, Factories, and Cities*, pp. 19–38, 6 2021.

[47] L. S. Laufenberg, "Impersonating lorawan gateways using semtech packet forwarder," *CoRR*, vol. abs/1904.10728, 2019. [Online]. Available: http://arxiv.org/abs/1904.10728

[48] F. Coman, K. Malarski, M. Petersen, and R. S, "Security issues in internet of things: Vulnerability analysis of lorawan, sigfox and nb-iot," *Glob. IoT Summit*, pp. 1–6, 2019.

[49] M. Ralambotiana, "Key management with a trusted third party using lorawan protocol : A study case for e2e security," *Pdfs.Semanticscholar.Org*, p. 78, 2018.

[50] S.-Y. Gao, X.-H. Li, and M.-D. Ma, "A malicious behavior awareness and defense countermeasure based on lorawan protocol," *Sensors*, vol. 19, p. 5122, 11 2019.

[51] M. Van Leent, "An improved key distribution and updating mechanism for low power wide area networks (lpwan)," Master's thesis, Universitei Leiden, 2017.

[52] J. Han and J. Wang, "An enhanced key management scheme for lorawan," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, pp. 407–416, 2018.

[53] V. Ribeiro, R. Holanda, A. Ramos, and J. J. P. C. Rodrigues, "Enhancing key management in lorawan with permissioned blockchain," *Sensors*, vol. 20, no. 11, 2020. [Online]. Available: https://www.mdpi.com/1424-8220/20/11/3068

[54] T. Dönmez and E. Nigussie, "Security of join procedure and its delegation in lorawan v1.1," in *The 15th International Conference on Mobile Systems and Pervasive Computing (MobiSPC 2018) / The 13th International Conference on Future Networks and Communications (FNC-2018) / Affiliated Workshops, Gran Canaria, Spain, August 13-15, 2018*, ser. Procedia Computer Science, A. Yasar and E. M. Shakshuki, Eds., vol. 134.   Elsevier, 2018, pp. 204–211. [Online]. Available: https://doi.org/10.1016/j.procs.2018.07.202

[55] J. Kim and J. Song, "A simple and efficient replay attack prevention scheme for lorawan," in *Proceedings of the 2017 the 7th International Conference on Communication and Network Security*, ser. ICCNS 2017.   New York, NY, USA: Association for Computing Machinery, 2017, p. 32–36. [Online]. Available: https://doi.org/10.1145/3163058.3163064

[56] S. Na, D. Hwang, W. Shin, and K. Kim, "Scenario and countermeasure for replay attack using join request messages in lorawan," *International Conference on Information Networking*, pp. 718–720, 2017.

[57] S. M. Danish, M. Lestas, H. K. Qureshi, K. Zhang, W. Asif, and M. Rajarajan, "Securing the lorawan join procedure using blockchains," *Cluster Computing*, vol. 23, no. 3, pp. 2123–2138, Sep 2020. [Online]. Available: https://doi.org/10.1007/s10586-020-03064-8

[58] M. Ingham, J. Marchang, and D. Bhowmik, "Iot security vulnerabilities and predictive signal jamming attack analysis in lorawan," *IET Inf. Secur*, vol. 14, pp. 368–379, 2020.

[59] S. G. Yoo and J. J. Barriga, "Privacy-aware authentication for wi-fi based indoor positioning systems," in *Applications and Techniques in Information Security*, L. Batten, D. S. Kim, X. Zhang, and G. Li, Eds.   Singapore: Springer Singapore, 2017, pp. 201–213.

[60] J. J. Barriga A., S. G. Yoo, and J. C. Polo, "Enhancement to the privacy-aware authentication for wi-fi based indoor positioning systems," in *Applied Cryptography and Network Security Workshops*, J. Zhou, R. Deng, Z. Li, S. Majumdar, W. Meng, L. Wang, and K. Zhang, Eds.   Cham: Springer International Publishing, 2019, pp. 143–155.

[61] M. T. Hammi, E. Livolant, P. Bellot, A. Serhrouchni, P. Minet, and P. A. Minet, "A lightweight iot security protocol," 2017. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01640510

[62] V. Varadharajan, U. Tupakula, and K. Karmakar, "Lightweight authentication mechanism and oauth protocol for iot devices," Advanced Cyber Security Engineering Research Centre, Tech. Rep., 2018. [Online]. Available: https://www.newcastle.edu.au/__data/assets/pdf_file/0003/552018/TR2-ISIF-ASIA.pdf

[63] R. Amin, N. Kumar, G. Biswas, R. Iqbal, and V. Chang, "A light weight authentication protocol for iot-enabled devices in distributed cloud computing environment," *Future Generation Computer Systems*, vol. 78, pp. 1005–1019, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X1630824X

[64] J. Oh, S. Yu, J. Lee, S. Son, M. Kim, and Y. Park, "A secure and lightweight authentication protocol for iot-based smart homes," *Sensors*, vol. 21, pp. 1–24, 2 2021. [Online]. Available: https://www.mdpi.com/1424-8220/21/4/1488

[65] M. Burrows, M. Abadi, and R. Needham, "A logic of authentication," *ACM Trans. Comput. Syst.*, vol. 8, no. 1, p. 18–36, feb 1990. [Online]. Available: https://doi.org/10.1145/77648.77649

[66] S. S. Ahamad and A. S. K. Pathan, "Trusted service manager (tsm) based privacy preserving and secure mobile commerce framework with formal verification," *Complex Adaptive Systems Modeling*, vol. 7, pp. 1–18, 12 2019. [Online]. Available: https://casmodeling.springeropen.com/articles/10.1186/s40294-019-0064-z

[67] C. Cremers, "Scyther: Semantics and verification of security protocols," Ph.D. dissertation, Technische Universiteit Eindhoven, 2006. [Online]. Available: https://pure.tue.nl/ws/files/2425555/200612074.pdf

[68] S. Peisert and M. Bishop, "How to design computer security experiments," in *Fifth World Conference on Information Security Education*, L. Futcher and R. Dodge, Eds. New York, NY: Springer US, 2007, pp. 141–148.

[69] J. J. Barriga and S. G. Yoo, "Securing end-node to gateway communication in lorawan with a lightweight security protocol," *IEEE Access*, vol. 10, pp. 96 672–96 694, 2022. [Online]. Available: https://ieeexplore.ieee.org/document/9875282

[70] J. M. S. Coral, "Adaptación de la biblioteca lmic en arduino para soportar la especificación lorawan versión 1.1." 7 2021. [Online]. Available: http://bibdigital.epn.edu.ec/handle/15000/21728

[71] I. Butun, N. Pereira, and M. Gidlund, "Security risk analysis of lorawan and future directions," *Futur. Internet*, vol. 11, pp. 1–22, 2018.

[72] T. Mundt, J. Bauer, A. Gladisch, J. Goltz, S. Rietschel, and S. Wiedenmann, "General security considerations of lorawan version 1.1 infrastructures," *MobiWac 2018 -Proc. 16th ACM Int. Symp. Mobil. Manag. Wirel. Access*, pp. 118–123, 2018.

[73] D. Basu, T. Gu, and P. Mohapatra, "Security issues of low power wide area networks in the context of lora networks," *CoRR*, vol. abs/2006.16554, 2020. [Online]. Available: https://arxiv.org/abs/2006.16554

[74] C. Cremers, "The scyther tool: Verification, falsification, and analysis of security protocols -tool paper," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial and Lecture Notes in Bioinformatics*, vol. 5123, pp. 414–418, 2008.

[75] A. Acosta, E. Tena-Sánchez, C. Jiménez, and J. Mora, "Power and energy issues on lightweight cryptography," *J. Low Power Electron*, vol. 13, pp. 326–337, 09 2017.

[76] K. Olha, "An investigation of lightweight cryptography and using the key derivation function for a hybrid scheme for security in iot," Master's thesis, Blekinge Institute of Technology, 2017.

[77] J.-P. Kaps and B. Sunar, "Energy comparison of aes and sha-1 for ubiquitous computing," in *Emerging Directions in Embedded and Ubiquitous Computing*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 372–381.

[78] M. Kumar, "Single bit full adder design using 8 transistors with novel 3 transistors xnor gate," *Int. J. VLSI Des. Commun. Syst*, vol. 2, pp. 47–59, 12 2011.

[79] K. Luke E, C. Jiaming James, T. Rebecca, L. Vicky, and M. Matthew, "Security and performance in iot: A balancing act," *IEEE Access*, vol. 8, pp. 121 969–121 986, 2020.

[80] K. Peter, B. Lena, L. Leandro, S. Thomas C, and W. A. Matthias, "A performance study of crypto-hardware in the low-end iot," in *Proceedings of the 2021 International Conference on Embedded Wireless Systems and Networks*. ACM, 2 2021, power Consumption for SHA-256 in mA. [Online]. Available: https://github.com/

[81] T. T. Industries, "The things network - lorawan airtime calculator," 2016. [Online]. Available: https://www.thethingsnetwork.org/airtime-calculator

[82] L. A. T. C. R. P. Workgroup, "Rp002-1.0.3 lorawan® regional parameters," San Ramon, CA, 5 2021. [Online]. Available: https://hz137b.p3cdn1.secureserver.net/wp-content/uploads/2021/05/RP002-1.0.3-FINAL-1.pdf?time=1672853176

[83] L. Alliance, "Lora basics™ station | developer portal," LoRa Alliance. [Online]. Available: https://lora-developers.semtech.com/build/software/lora-basics/lora-basics-for-gateways

[84] I. Chatzigiannakis, V. Liagkou, and L. &, "Brief announcement: Providing end-to-end secure communication in low-power wide area networks," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) (Vol. 10879 LNCS*, pp. 101–104, 2018.

[85] E. van Es, "Lorawan vulnerability analysis:(in) validation of possible vulnerabilities in the lorawan protocolspecification." Master's thesis, OpenUniversity of the Netherlands, 2018.

[86] Gartner, "Gartner says 4.9 billion connected "things" will be in use in 2015. retrieved," Gartner Inc., Tech. Rep., 11 2014. [Online]. Available: https://www.gartner.com/en/newsroom/press-releases/2017-02-07-gartner-says-8-billion-connected-things-will-be-in-use-in-2017-\up-31-percent-from-2016

[87] M. Knight and B. Seeber, "Decoding lora: Realizing a modern lpwan with sdr," *Proceedings of the GNU Radio Conference*, vol. 1, no. 1, 2016. [Online]. Available: https://pubs.gnuradio.org/index.php/grcon/article/view/8

[88] R. Krejčí, O. Hujňák, and M. Švepeš, "Security survey of the iot wireless protocols," *2017 25th Telecommunication Forum (TELFOR)*, pp. 1–4, 2017.

[89] C. Kuo, V. Chang, and C. Lei, "A feasibility analysis for edge computing fusion in lpwa iot environment with sdn structure," *Proceedings of 2017 International Conference on Engineering and Technology, ICET 2017*, pp. 1–6, 2018.

[90] K. Ema and S. Mark, "Security-related research in ubiquitous computing - results of a systematic literature review," *CoRR*, vol. abs/1701.00773, 2017. [Online]. Available: http://arxiv.org/abs/1701.00773

[91] V. Liagkou, C. Stylios, and D. Salmas, "Vr training model for exploiting security in lpwan," *Procedia CIRP*, vol. 79, pp. 724–729, 2019.

[92] M. Kais, B. Eddy, C. Frederic, and M. Fernand, "A comparative study of lpwan technologies for large-scale iot deployment," *ICT Express*, vol. 5, no. 1, pp. 1–7, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2405959517302953

[93] R, Hwang, D. Shin, W. Kim, and K, "Scenario and countermeasure for replay attack using join request messages in lorawan," *Mendeley -Reference Management Software & Researcher Network*, pp. 718–720, 08 2016.

[94] K. Olsson, S. Finnsson, V. Dadarlat, E. De Poorter, and A. Munteanu, "Exploring lora and lorawan: A suitable protocol for iot weather stations," *Proceedings of IEEE Sensors*, 2017.

[95] B. Oniga, V. Dadarlat, E. De Poorter, and A. Munteanu, "Analysis, design and implementation of secure lorawan sensor networks," *Proceedings -2017 IEEE 13th International Conference on Intelligent Computer Communication and Processing*, pp. 421–428, 2017.

[96] U. Raza, P. Kulkarni, and M. Sooriyabandara, "Low power wide area networks: An overview," *IEEE Communications Surveys and Tutorials*, vol. 19, pp. 855–873, 2017.

[97] Eef, H. Vranken, and A. Hommersom, "Denial-of-service attacks on lorawan," *Proceedings of the*, 08 2018.

[98] J. J. Barriga A and S. G. Yoo, "Security over smart home automation systems: A survey," in *Developments and Advances in Defense and Security*, Á. Rocha and T. Guarda, Eds. Cham: Springer International Publishing, 2018, pp. 87–96.

[99] T. Yang, F. Zhai, H. Xu, and W. Li, "Design of a secure and efficient authentication protocol for real-time accesses of multiple users in piot-oriented multi-gateway wsns," *Energy Reports*, vol. 8, pp. 1200–1211, 2022, 2021

International Conference on New Energy and Power Engineering. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2352484722003080

[100] J. Oh, S. Yu, J. Lee, S. Son, M. Kim, and Y. Park, "A secure and lightweight authentication protocol for iot-based smart homes," *Sensors*, vol. 21, no. 4, 2021. [Online]. Available: https://www.mdpi.com/1424-8220/21/4/1488

ANNEXES

Annex 1: Arduino Sketch LoRaWAN Enhanced Version

```cpp
/***************************************************************************
 * LoRaWAN v1.1 with Security Enhancementes for ESP32 Devices
 * @author Jhonattan Barriga
 * v1.0 - gwSKey derivation and sensor metrics
 *
 ***************************************************************************/

#include <Arduino.h>
#include <lmic.h>
#include <hal/hal.h>
#include <SPI.h>
#include "heltec.h"

#ifdef COMPILE_REGRESSION_TEST
# define FILLMEIN 0
#else
# warning "You must replace the values marked FILLMEIN with real values from the TTN control
    panel!"
# define FILLMEIN (#dont edit this, edit the lines that use FILLMEIN)
#endif
#define CFG_us915 1

//LoRaWAN initial security features
static const u1_t PROGMEM APPEUI[8]={ 0xF9, 0x51, 0x02, 0xD0, 0x7E, 0xD5, 0xB3, 0x70 };
void os_getArtEui (u1_t* buf) { memcpy_P(buf, APPEUI, 8);}
//Dev EUI
static const u1_t PROGMEM DEVEUI[8]={ 0x63, 0x24, 0xBC, 0x18, 0x84, 0xF7, 0x8A, 0x4E };
//JoinEUI
static const u1_t PROGMEM JoinEui[8] = { 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08 };
void os_getJoinEui (u1_t* buf) { memcpy_P(buf, JoinEui, 8);}

// This key should be in big endian format.
//AppKey
static const u1_t PROGMEM APPKEY[16] = { 0xD8, 0x9E, 0x6E, 0xD1, 0x37, 0x76, 0x86, 0x8D,
    0x7B, 0xAE, 0xCB, 0x3E, 0x40, 0x46, 0x30, 0x47 };
void os_getAppKey (u1_t* buf) {  memcpy_P(buf, APPKEY, 16);}
//NwkKey
static const u1_t PROGMEM NwkKey[16] = { 0xA1, 0xB2, 0xC3, 0xD4, 0xE5, 0xF6, 0xA7, 0xB8,
    0xC1, 0xD2, 0xE3, 0xF4, 0xA5, 0xB6, 0xC7, 0xD8 };
void os_getNwkKey (u1_t* buf) { memcpy_P(buf, NwkKey, 16);}

static uint8_t mydata[] = "Node A - V1.1 Modified "; //Part of payload to be sent
static osjob_t sendjob;

//Empty functions
void os_getDevEui (u1_t* buf) { memcpy_P(buf, DEVEUI, 8);}
void os_saveFCntUp (u4_t cntr){}
```

```
46  void os_getFCntUp (u4_t* buf){}
47  void os_saveNFCntDown (u4_t buf){}
48  void os_getNFCntDown (u4_t* buf){}
49  void os_saveAFCntDown (u4_t buf){}
50  void os_getAFCntDown (u4_t* buf){}
51  void os_saveDevNonce (u2_t buf){}
52  void os_getDevNonce (u2_t* buf){}
53  void os_saveJoinNonce (u2_t buf){}
54  void os_getJoinNonce (u2_t* buf){}
55  u1_t os_deviceRestarted (){return 0;}
56
57  // Schedule TX every this many seconds (might become longer due to duty cycle limitations).
58  const unsigned TX_INTERVAL = 5;
59
60  // Pin mapping
61  const lmic_pinmap lmic_pins = {
62    .nss = 18,
63    .rxtx = LMIC_UNUSED_PIN,
64    .rst = 14,
65    .dio = {26, 33, 32}
66  };
67
68
69  //Sensor variables.
70  float Celsius, Fahrenheit, Kelvin;
71  float currentTemp;
72  float currentHumidity;
73  int sensorPin = 34; // select the input pin for the LDR
74  int sensorValue = 0; // variable to store the value coming from the sensor
75  int voltagePin = 37;
76
77  //Sensor Ultrasonic
78  #define trigPin 17
79  #define echoPin 2
80
81  float duration;
82  float distance;
83  String totalDistance;
84
85  //Convert bytes to Hex Format
86  void printHex2(unsigned v) {
87      v &= 0xff;
88      if (v < 16)
89          Serial.print('0');
90      Serial.print(v, HEX);
91  }
92
93  //Event Handler
94  void onEvent (ev_t ev) {
95      Serial.print(os_getTime());
96      Serial.print(": ");
```

```
 97        switch(ev) {
 98            case EV_SCAN_TIMEOUT:
 99                Serial.println(F("EV_SCAN_TIMEOUT"));
100                break;
101            case EV_BEACON_FOUND:
102                Serial.println(F("EV_BEACON_FOUND"));
103                break;
104            case EV_BEACON_MISSED:
105                Serial.println(F("EV_BEACON_MISSED"));
106                break;
107            case EV_BEACON_TRACKED:
108                Serial.println(F("EV_BEACON_TRACKED"));
109                break;
110            case EV_JOINING:
111                Serial.println(F("EV_JOINING"));
112                Heltec.display -> clear();
113                Heltec.display -> drawString(5,0, "EV_JOINING");
114                Heltec.display -> display();
115                break;
116            case EV_JOINED:  //Device Joined, print session keys
117                Heltec.display -> clear();
118                Heltec.display -> drawString(5,0, "EV_JOINED_DEVICE");
119                Heltec.display -> display();
120                Serial.println(F("EV_JOINED"));
121                {
122                  u4_t netid = 0;
123                  devaddr_t devaddr = 0;
124                  u1_t nwkKey[16];
125                  u1_t artKey[16];
126                  u1_t gwSKey[16];
127                  LMIC_getSessionKeys(&netid, &devaddr, nwkKey, artKey, gwSKey);
128                  Serial.print("netid: ");
129                  Serial.println(netid, DEC);
130                  Serial.print("devaddr: ");
131                  Serial.println(devaddr, HEX);
132                  Serial.print("AppSKey: ");
133                  for (size_t i=0; i<sizeof(artKey); ++i) {
134                    if (i != 0)
135                      Serial.print("-");
136                    printHex2(artKey[i]);
137                  }
138                  Serial.println("");
139                  Serial.print("NwkSKey: ");
140                  for (size_t i=0; i<sizeof(nwkKey); ++i) {
141                          if (i != 0)
142                                  Serial.print("-");
143                          printHex2(nwkKey[i]);
144                  }
145                  Serial.println();
146                  Serial.print("GwSKey: ");
147                  for (size_t i=0; i<sizeof(gwSKey); ++i) {
```

```
148                    if (i != 0)
149                        Serial.print("-");
150                    printHex2(gwSKey[i]);
151            }
152            Serial.println();
153
154        }
155        // Disable link check validation (automatically enabled
156        // during join, but because slow data rates change max TX
157        // size, we don't use it in this example.
158        LMIC_setLinkCheckMode(0);
159        break;
160    case EV_JOIN_FAILED:
161        Serial.println(F("EV_JOIN_FAILED"));
162        break;
163    case EV_REJOIN_FAILED:
164        Serial.println(F("EV_REJOIN_FAILED"));
165        break;
166    case EV_TXCOMPLETE:
167        Serial.println(F("EV_TXCOMPLETE (includes waiting for RX windows)"));
168        if (LMIC.txrxFlags & TXRX_ACK)
169          Serial.println(F("Received ack"));
170        if (LMIC.dataLen) {
171          Serial.print(F("Received "));
172          Serial.print(LMIC.dataLen);
173          Serial.println(F(" bytes of payload"));
174        }
175        // Schedule next transmission
176        os_setTimedCallback(&sendjob, os_getTime()+sec2osticks(TX_INTERVAL), do_send);
177        break;
178    case EV_LOST_TSYNC:
179        Serial.println(F("EV_LOST_TSYNC"));
180        break;
181    case EV_RESET:
182        Serial.println(F("EV_RESET"));
183        break;
184    case EV_RXCOMPLETE:
185        // data received in ping slot
186        Serial.println(F("EV_RXCOMPLETE"));
187        break;
188    case EV_LINK_DEAD:
189        Serial.println(F("EV_LINK_DEAD"));
190        break;
191    case EV_LINK_ALIVE:
192        Serial.println(F("EV_LINK_ALIVE"));
193        break;
194    case EV_TXSTART:                //Start transmitting LoRaWAN packets
195        Serial.println(F("EV_TXSTART"));
196        Heltec.display -> clear();
197        Heltec.display -> drawString(5,0, "EV_TXSTART");
198        Heltec.display -> drawString(5,10, "Freq: "+String(LMIC.freq));
```

```
199            Heltec.display -> drawString(5,22, "DevNonce: "+String(LMIC.devNonce));
200            Heltec.display -> drawString(5,32, "Node A - V1.1 Modified");
201            Heltec.display -> drawString(5,42,
    ↪ String(ReadVoltage(voltagePin))+"V-"+GetSensorData(sensorPin));
202            Heltec.display -> display();
203            //Log message - device information
204            Serial.println("Freq: "+String(LMIC.freq)+", DevNonce:
    ↪ "+String(LMIC.devNonce)+", SF: "+String(LMIC.datarate)+", RSSI: "+String(LMIC.rssi)+",
    ↪ JoinNonce: "+String(LMIC.joinNonce));
205            Serial.print("JoinEUI: ");
206            for (size_t i=0; i<sizeof(JoinEui); ++i) {
207                if (i != 0)
208                    Serial.print("-");
209                printHex2(JoinEui[i]);
210             }
211             Serial.println("");
212            break;
213        case EV_TXCANCELED:
214            Serial.println(F("EV_TXCANCELED"));
215            break;
216        case EV_RXSTART:
217            break;
218        case EV_JOIN_TXCOMPLETE:
219            Serial.println(F("EV_JOIN_TXCOMPLETE: no JoinAccept"));
220            Heltec.display -> clear();
221            Heltec.display -> clear();
222            Heltec.display -> drawStringMaxWidth(5,0,128, "EV_JOIN_TXCOMPLETE: no
    ↪ JoinAccept");
223            Heltec.display -> display();
224            break;
225
226        default:
227            Serial.print(F("Unknown event: "));
228            Serial.println((unsigned) ev);
229            break;
230    }
231 }
232
233 void do_send(osjob_t* j){
234    LMIC_setDrTxpow(DR_SF7,14);
235    // Check if there is not a current TX/RX job running
236    if (LMIC.opmode & OP_TXRXPEND) {
237        Serial.println(F("OP_TXRXPEND, not sending"));
238    } else {
239        // Prepare upstream data transmission at the next possible time.
240
241        String strSensorData = "Node A - V1.1 Modified SF7 ->
    ↪ "+String(ReadVoltage(voltagePin)) + "V - " +GetSensorData(sensorPin);
242        char charMsgToSend[46];
243        strSensorData.toCharArray(charMsgToSend, strSensorData.length()+1);
244        LMIC_setTxData2(1, (uint8_t *)charMsgToSend, sizeof(charMsgToSend)-1, 0);
```

```
245         Serial.println(strSensorData);
246         Serial.println(F("Packet queued"));
247     }
248     // Next TX is scheduled after TX_COMPLETE event.
249 }
250
251 void setup() {
252     pinMode(trigPin, OUTPUT); // Sets the trigPin as an Output
253     pinMode(echoPin, INPUT); // Sets the echoPin as an Input
254     Serial.begin(115200);
255
256     Serial.println(F("Starting"));
257     Heltec.begin(true /*DisplayEnable Enable*/, true /*LoRa Disable*/, true /*Serial
    ↪   Enable*/, true /*PABOOST Enable*/, 470E6 /**/);
258     Heltec.display -> drawString(0,0, "Node A - V1.1 Modified");
259     Heltec.display -> clear();
260     Heltec.display -> display();
261
262     // LMIC init
263     os_init();
264
265     LMIC_reset();
266
267     #if defined(CFG_us915)
268     LMIC_selectSubBand(1);  //903.1 - 904.9MHz
269     //LMIC_selectSubBand(7);  //903.1 - 904.9MHz
270     #endif
271
272     LMIC_setLinkCheckMode(0);
273     // Set data rate and transmit power for uplink (note: txpow seems to be ignored by the
    ↪   library)
274     //Specify the SF to transmit
275     //DR_SF7 - 125 KhZ
276     //DR_SF8 - 125 KhZ
277     //DR_SF9 - 125 KhZ
278     LMIC_setDrTxpow(DR_SF7,14);
279
280     // Start job
281     do_send(&sendjob);
282
283 }
284
285
286 void loop() {
287     os_runloop_once();
288     getDeviceStatistics();
289
290 }
291
292 void getDeviceStatistics(){
293     //Hardware Statistics
```

```
294    long T0 = millis();
295    uint32_t startCycle = ESP.getCycleCount();
296    while (millis() < T0 + 1000) {}
297    uint32_t endCycle = ESP.getCycleCount();
298    Serial.printf("CPU Usage Hz: %u\n", endCycle - startCycle);
299    startCycle = ESP.getCycleCount();
300    delay(1000);
301    endCycle = ESP.getCycleCount();
302    Serial.printf("CPU Usage Hz: %u\n", ((endCycle - startCycle)));
303    Serial.println("Heap Free Bytes: "+String(ESP.getFreeHeap()));
304
305    //Power Consumption Statistics (Volts)
306    Serial.println("Battery : "+String(ReadVoltage(voltagePin))+" V");
307
308    //Sensor Data
309    Serial.println("Photo Sensor : "+GetSensorData(sensorPin));
310  }
311
312  double ReadVoltage(byte pin){
313    double sensorValue = analogRead(pin); // Reference voltage is 3.7 so maximum reading is
    ↪ 3.7 = 4095 in range 0 to 4095
314    float voltage = ((float) sensorValue/4095)*3.7;
315    Serial.println("Volts:"+String(voltage)+ "Sensor value: "+sensorValue);
316    return voltage;
317  }
318
319  String GetSensorData(byte sensorPin)
320  {
321
322    sensorValue = analogRead(sensorPin);
323
324    float voltage = sensorValue * (5.0 / 1024.0);
325    String message = "";
326
327    // the lower the voltage, the brighter it is
328    if ((voltage >= 0) && (voltage <= 0.4)) {
329      message = "it is light - ";
330    } else if ((voltage > 0.4) && (voltage <= 2)) {
331      message = "it is bright - ";
332    } else {
333      message = "it is dark - ";
334    }
335    // print out the value you read:
336    Serial.println("Value read:"+String(voltage)+" "+message+" sensorvalue: "+sensorValue);
337    return " "+String(sensorValue)+" LUM";
338
339  }
340
341  String GetDistance() {
342    // Clears the trigPin
343    digitalWrite(trigPin, LOW);
```

```
344    delayMicroseconds(2);
345    // Sets the trigPin on HIGH state for 10 micro seconds
346    digitalWrite(trigPin, HIGH);
347    delayMicroseconds(5);
348    digitalWrite(trigPin, LOW);
349    // Reads the echoPin, returns the sound wave travel time in microsecondduration =
   ↪    pulseIn(echoPin, HIGH);
350    duration = pulseIn(echoPin, HIGH);
351    // Calculating the distance
352    distance= duration*0.034/2;
353    // Prints the distance on the Serial Monitor
354    Serial.print("Distance: ");
355    Serial.println(distance);
356    delay(50);
357    return ("D: "+String(distance)+" cm.");
358    }
```

Code 2: ESP32 Arduino Sketch Code

Annex 2: SPDL Code for formal verification security analysis

```
1    // The protocol is running between Gateway (Gateway) and Network Server (NS).
2    // The predefined shared key userCreds=h(IDui||h(PWui)) between Gateway and Server is
  ↪  k(Gateway,NS).
3    // dec models a decryption function that is invertible by an encryption function (enc)
4    usertype String;
5    usertype AEScmac;
6    usertype SymKey;
7    hashfunction h;
8
9    //const xor:Function;
10
11   const GwEUI: String;
12   const IDui: String;
13   const PWui: String;
14
15   macro userCreds=h(IDui,h(PWui));
16
17   protocol LoRaWAN-OTAA-v11-GWRegistration(Gateway,NS)
18   {
19   role Gateway {
20         fresh RNonceListG:Nonce;
21         fresh RN1: Nonce;
22
23         fresh MICGw: AEScmac;
24         fresh RSK,GwKey: SymKey;
25         var RN2,RN3: Nonce;
26         var GwKeyJ,GwKeyJ1,GwKeyJN:Nonce;
27
28         macro GwSKa=h(RSK,GwKey);
29
30         macro GrpKeyGrpId=h(GwKey,GwKeyJ,GwKeyJ1,GwKeyJN,RN3);
31
32         macro M1= {userCreds,RN1,GwEUI}k(GwSKa),IDui;
33         macro M2= {RN1,RN2,GrpKeyGrpId}k(GwSKa);
34         macro M3={RN2,MICGw}k(GrpKeyGrpId);
35
36         send_1(Gateway,NS,M1);
37         recv_2(NS,Gateway,M2);
38         match (RN2, RNonceListG);
39         macro RNonceListG=(RNonceListG, RN2);
40         send_3(Gateway,NS,M3);
41
42
43         claim_gw1(Gateway,Running,NS,RN1); //checks that Gateway agrees with NS on
  ↪  RN1
44         claim_gw2(Gateway,Alive); //assures the Aliveness of Gateway
45         claim_gw3(Gateway,Weakagree); //minimum agreement check between partners according
  ↪  to Gateway
```

91

```
46        claim_gw4(Gateway,Niagree); //validates the non-injective agreement according to
   ↪ Gateway
47        claim_gw5(Gateway,Nisynch); //validates the non-injective synchronization according
   ↪ to Gateway
48        claim_gw6(Gateway,Secret,GrpKeyGrpId); //validates the secrecy of GrpKeyGrpId
   ↪ according to Gateway
49
50        }
51
52  role NS {
53        fresh RNonceList, RN2,RN3:Nonce;
54        var RN1: Nonce;
55        var MICGw:AEScmac;
56        //var GwSKa:String;
57        var RSK,GwKey: SymKey;
58        fresh GwKeyJ,GwKeyJ1,GwKeyJN:Nonce;
59        secret GrpKeyGrpId:Nonce;
60
61        recv_1(Gateway,NS,M1);
62        match (RN1, RNonceList);
63        macro RNonceList=(RNonceList, RN1);
64
65        macro GrpKeyGrpId=h(GwKey,GwKeyJ,GwKeyJ1,GwKeyJN,RN3);
66
67        send_2(NS,Gateway,M2);
68        recv_3(Gateway,NS,M3);
69        match (RN2, RNonceList);
70        macro RNonceList=(RNonceList, RN2);
71
72
73        claim_j1(NS,Running,Gateway,RN2); //checks that NS agrees with Dev on RN2
74        claim_j2(NS,Alive); //assures the Aliveness of NS
75        claim_j3(NS,Weakagree); //minimum agreement check between partners according to NS
76        claim_j4(NS,Niagree); //validates the non-injective agreement according to NS
77        claim_j5(NS,Nisynch); //validates the non-injective synchronization according to NS
78        claim_j6(NS,SKR,GrpKeyGrpId); //validates the secrecy of GrpKeyGrpId according to
   ↪ NS
79
80        }
81  }
82
83
```

Code 3: SPDL Scyther verification Code for Gateway Authentication

[p]

```
1   // The protocol is running between End Device (Dev), Gateway and Network Server/Join Server
   ↪ (Join).
2   // The predefined shared key (NwkKey) between End Device and Server is k(Dev,Join).
3   // The predefined shared key (GwKey) between Gateway and Network/Join Server is
   ↪ k(Gateway,Join)
```

```
4    // dec models a decryption function that is invertible by an encryption function (enc)
5    const dec: Function;
6    usertype String;
7    const pad01,pad02,pad03,pad04,pad05,pad06,pad07,pad08: String;
8    const pad09,pad10,pad11,pad12,pad13,pad14,pad15,pad16: String;
9
10   secret Appkey,NonceList,GwKey: String;
11
12   hashfunction h;
13
14   protocol GwSKeyDerivation (Dev,Join,Gateway,NS)
15   {
16        role Dev {
17             fresh DevNonce: Nonce;
18             fresh MHDRDev: String;
19             var MHDRSrv: String;
20             var JoinNonce: Nonce;
21             var NetID: String;
22             var DevAddr: String;
23             var DLSettings: String;
24             var RxDelay: String;
25             var CFList: String;
26             var JoinReqType: String;
27             fresh JoinEUI:String;
28             fresh DevEUI:String;
29
30             macro JoinNonceMIC = {MHDRDev,JoinEUI,DevEUI,DevNonce}k(Dev,Join);
31             send_1(Dev,Join,(MHDRDev,JoinEUI,DevEUI,DevNonce),JoinNonceMIC);
32
33             macro JSIntKey={pad06,Dev,pad16 }k(Dev,Join);
34             macro
→    MIC={JoinReqType,Join,DevNonce,MHDRSrv,JoinNonce,NetID,DevAddr,DLSettings,RxDelay,CFList}JSIntKey;
35             recv_2 (Join,Dev,
→    MHDRSrv,{{JoinNonce,NetID,DevAddr,DLSettings,RxDelay,CFList,MIC}dec} k(Dev,Join),MIC);
36
37             /*LoRaWAN Session Derivation Keys*/
38             macro FNwkSIntKey={pad01,JoinNonce,Join,DevNonce,pad16}k(Dev,Join);
39             macro SNwkSIntKey={pad03,JoinNonce,Join,DevNonce,pad16}k(Dev,Join);
40             macro
→    NwkSEncKey={pad04,JoinNonce,Join,DevNonce,pad16}k(Dev,Join);
41             macro JSEncKey={pad05,Dev,pad16}k(Dev,Join);
42             macro AppSKey={pad02,JoinNonce,Join,DevNonce, pad16 }Appkey;
43
44             /*LoRaWAN Session Derivation Key for EN-GW*/
45             macro GwSKeyD = h((k(Dev,Join),Appkey),DevNonce,JoinNonce,Join);
46
47             claim(Dev,Running,Join,DevNonce); //checks that Dev agrees with Join on
→    DevNonce
48             claim(Dev,Alive); //assures the Aliveness of Dev
49             claim(Dev,Weakagree); //minimum agreement check between partners according
→    to Dev
```

```
50                    claim(Dev,Niagree); //validates the non-injective agreement according to
   ↪ Dev
51                    claim(Dev,Nisynch); //validates the non-injective synchronization according
   ↪ to Dev
52                    claim (Dev,SKR,FNwkSIntKey); //validates the secrecy of FNwkSIntKey
   ↪ according to Dev
53                    claim (Dev,SKR,SNwkSIntKey); //validates the secrecy of SNwkSIntKey
   ↪ according to Dev
54                    claim (Dev,SKR,NwkSEncKey); //validates the secrecy of NwkSEncKey according
   ↪ to Dev
55                    claim (Dev,SKR,AppSKey); //validates the secrecy of AppSKey according to
   ↪ Dev
56                    claim (Dev,SKR,JSEncKey); //validates the secrecy of JSEncKey according to
   ↪ Dev
57                    claim (Dev,SKR,JSIntKey); //validates the secrecy of JSIntKey according to
   ↪ Dev
58                    claim (Dev,SKR,GwSKeyD); //validates the secrecy of JSIntKey according to
   ↪ Dev
59          }
60      role Gateway{
61
62                    var DevEUI: String;
63                    var GwEUI:String;
64                    var DevNonce: Nonce;
65                    var JoinNonce: Nonce;
66
67                    /*Receives M1 from Join containing GwSessionKey*/
68                    macro GwSKeyGw = h(k(Dev,Join),Appkey,DevNonce,JoinNonce,Join);
69                    macro M2= {{GwEUI,GwSKeyGw,DevEUI}dec}k(Gateway,NS);
70                    recv_m2gw(NS, Gateway,M2);
71
72                    claim (Gateway,Secret,k(Join,Gateway)); //validates the secrecy of GwKey
   ↪ according to Gateway
73                    claim (Gateway,SKR,GwSKeyGw); //validates the secrecy of GwSKey according to
   ↪ Gateway
74                    claim(Gateway,Weakagree);
75                    claim(Gateway,Niagree);
76                    claim(Gateway,Nisynch);
77                    claim(Gateway,Alive);
78          }
79      role NS {
80                    var DevEUI: String;
81                    var GwEUI:String;
82                    var DevNonce: Nonce;
83                    var JoinNonce: Nonce;
84
85                    macro M1= {{GwEUI,GwSKeyGw,DevEUI}dec}k(Join,NS);
86                    recv_m1ns(Join, NS,M1);
87
88                    macro M2=  {{GwEUI,GwSKeyGw,DevEUI}dec}k(Gateway,NS);
89                    send_m2gw(NS, Gateway,M2);
```

```
90
91          }
92      role Join {
93              fresh JoinNonce: Nonce;
94              fresh MHDRSrv: String;
95              fresh NetID: String;
96              fresh DevAddr: String;
97              fresh DLSettings: String;
98              fresh RxDelay: String;
99              fresh CFList: String;
100             fresh NonceList: String;
101             fresh JoinReqType: String;
102             fresh DevEUI: String;
103             fresh GwEUI:String;
104             var DevNonce: Nonce;
105             var MHDRDev: String;
106             var JoinEUI:String;
107             var DevEUI:String;
108
109             macro JoinNonceMIC = {MHDRDev,JoinEUI,DevEUI,DevNonce}k(Dev,Join);
110             recv_1(Dev,Join,(MHDRDev,JoinEUI,DevEUI,DevNonce),JoinNonceMIC);
111             send_2
↪   (Join,Dev,MHDRSrv,{{JoinNonce,NetID,DevAddr,DLSettings,RxDelay,CFList,MIC}dec}k(Dev,Join),MIC);
112
113             /*Sends M1 to Gateway containing GwSessionKey*/
114             macro GwSKeyGw = h(k(Dev,Join),Appkey,DevNonce,JoinNonce,Join);
115             /*macro M1= {{GwEUI,GwSKeyGw,DevEUI}dec}k(Join,Gateway);
116             send_m1gw(Join, Gateway,M1);*/
117             macro M1= {{GwEUI,GwSKeyGw,DevEUI}dec}k(Join,NS);
118             send_m1ns(Join, NS,M1);
119
120             not match (DevNonce, NonceList);
121             macro NonceList=(NonceList, DevNonce);
122
123             claim(Join,Running,Dev,JoinNonce); //checks that Join agrees with Dev on
↪   JoinNonce
124             claim(Join,Alive); //assures the Aliveness of Join
125             claim(Join,Weakagree); //minimum agreement check between partners according
↪   to Join
126             claim(Join,Niagree); //validates the non-injective agreement according to
↪   Join
127             claim(Join,Nisynch); //validates the non-injective synchronization according
↪   to Join
128             claim(Join, SKR, FNwkSIntKey); //validates the secrecy of FNwkSIntKey
↪   according to Join
129             claim(Join, SKR, SNwkSIntKey); //validates the secrecy of SNwkSIntKey
↪   according to Join
130             claim(Join, SKR, NwkSEncKey); //validates the secrecy of NwkSEncKey
↪   according to Join
131             claim(Join, SKR, AppSKey); //validates the secrecy of AppSKey according to
↪   Join
```

```
132                    claim(Join, SKR, JSEncKey); //validates the secrecy of JSEncKey according to
    ↪   Join
133                    claim(Join, SKR, JSIntKey); //validates the secrecy of JSIntKey according to
    ↪   Join
134            }
135
136  }
```

Code 4: SPDL Scyther verification Code for GWSKey derivation

```
1    // The protocol is running between End Device (Dev), Gateway and Network Server/Join Server
    ↪   (Join).
2    // The predefined shared key (NwkKey) between End Device and Server is k(Dev,Join).
3    // The predefined shared key GrpKeyGrpId) between Gateway and Network Server is
    ↪   k(Gateway,NS)
4    // dec models a decryption function that is invertible by an encryption function (enc)
5    const dec: Function;
6    usertype String;
7    const pad01,pad02,pad03,pad04,pad05,pad06,pad07,pad08: String;
8    const pad09,pad10,pad11,pad12,pad13,pad14,pad15,pad16: String;
9    usertype TupleDB;
10
11   secret Appkey,NonceList,GwKey: String;
12
13   hashfunction h,cmac;
14
15   protocol UMOAEG (Dev,Join,Gateway,AS,NS)
16   {
17          role Dev {
18                  fresh DevNonce: Nonce;
19                  fresh MHDRDev: String;
20                  var MHDRSrv: String;
21                  var JoinNonce: Nonce;
22                  var NetID: String;
23                  var DevAddr: String;
24                  var DLSettings: String;
25                  var RxDelay: String;
26                  fresh CFList: String;
27                  var JoinReqType: String;
28                  fresh JoinEUI:String;
29                  fresh DevEUI:String;
30                  fresh FRMPayload:String;
31                  fresh FPort,FOpts,FCnt,FCtrl:String;
32
33                  macro JoinNonceMIC = {MHDRDev,JoinEUI,DevEUI,DevNonce}k(Dev,Join);
34                  send_1(Dev,Join,(MHDRDev,JoinEUI,DevEUI,DevNonce),JoinNonceMIC);
35
36                  macro JSIntKey={pad06,Dev,pad16 }k(Dev,Join);
37                  macro
    ↪   MIC={JoinReqType,Join,DevNonce,MHDRSrv,JoinNonce,NetID,DevAddr,DLSettings,RxDelay,CFList}JSIntKey;
```

```
38                      recv_2 (Join,Dev,
↪   MHDRSrv,{{JoinNonce,NetID,DevAddr,DLSettings,RxDelay,CFList,MIC}dec} k(Dev,Join),MIC);

39
40                      /*LoRaWAN Session Derivation Keys*/
41                      macro FNwkSIntKey={pad01,JoinNonce,Join,DevNonce,pad16}k(Dev,Join);
42                      macro SNwkSIntKey={pad03,JoinNonce,Join,DevNonce,pad16}k(Dev,Join);
43                      macro NwkSEncKey={pad04,JoinNonce,Join,DevNonce,pad16}k(Dev,Join);
44                      macro JSEncKey={pad05,Dev,pad16}k(Dev,Join);
45                      macro AppSKey={pad02,JoinNonce,Join,DevNonce, pad16 }Appkey;

46
47                      /*LoRaWAN Session Derivation Key for EN-GW*/
48                      macro GwSKeyD = h((k(Dev,Join),Appkey),DevNonce,JoinNonce,Join);

49
50                      /*Preparing MSG to Gw*/
51                      macro GwDevId = h(GwSKeyD,DevEUI);
52                      macro MDevice=(DevAddr,FCtrl,FCnt,FOpts,FPort,{FRMPayload}k(Dev,AS),MIC);
53                      macro MICPENi =cmac(GwSKeyD,MDevice,GwDevId,FCnt);
54                      macro MICPy= cmac(AppSKey,FRMPayload);
55                      macro M1GW=MDevice,MICPy,MICPENi,GwDevId;

56
57                      send_m1(Dev,Gateway,M1GW);

58
59                      claim(Dev,Running,Join,DevNonce); //checks that Dev agrees with Join on
↪   DevNonce
60                      claim(Dev,Alive); //assures the Aliveness of Dev
61                      claim(Dev,Weakagree); //minimum agreement check between partners according
↪   to Dev
62                      claim(Dev,Niagree); //validates the non-injective agreement according to
↪   Dev
63                      claim(Dev,Nisynch); //validates the non-injective synchronization according
↪   to Dev
64                      claim (Dev,SKR,AppSKey); //validates the secrecy of AppSKey according to
↪   Dev
65                      claim (Dev,SKR,GwSKeyD); //validates the secrecy of JSIntKey according to
↪   Dev
66              }
67          role Gateway{

68
69                      var JoinNonce: Nonce;
70                      fresh MHDRSrv: String;
71                      fresh NetID: String;
72                      fresh DevAddr: String;
73                      fresh DLSettings: String;
74                      fresh RxDelay: String;
75                      fresh CFList: String;
76                      fresh NonceList: String;
77                      fresh JoinReqType: String;
78                      fresh DevEUI: String;
79                      fresh GwEUI:String;
80                      var DevNonce: Nonce;
81                      fresh MHDRDev: String;
```

97

```
82              fresh JoinEUI:String;
83              var DevEUI:String;

84
85              /*Receives M1 from Join containing GwSessionKey*/
86              macro GwSKeyGw = h(k(Dev,Join),Appkey,DevNonce,JoinNonce,Join);
87              macro M1= {{GwEUI,GwSKeyGw,DevEUI}dec}k(Join,Gateway);
88              recv_m1gw(Join, Gateway,M1);

89
90              /*Receives M3 from End Node*/
91              fresh Mtype,EJP,Major: String;
92              var DevAddr, FCtrl, FCnt, FOpts,FPort: String;
93              var FRMPayload: String;

94
95              macro M1GW=MDevice,MICPy,MICPENi,GwDevId;

96
97              recv_m1(Dev,Gateway,M1GW);

98
99              match((GwDevId,GwSKeyGw),TupleDB);
100             macro GwDB=(GwDB, (GwDevId,GwSKeyGw));

101
102             macro MICPENi'=cmac(GwSKeyD,MDevice,GwDevId,FCnt);
103             match(MICPENi',MICPENi);

104
105             /*Sends M2 to NS*/
106             macro M3={M1GW}k(Gateway,NS);
107             send_m3(Gateway,NS,M3);

108
109             claim (Gateway,Secret,k(Join,Gateway)); //validates the secrecy of GwKey
     ↪   according to Gateway
110             claim (Gateway,SKR,GwSKeyGw); //validates the secrecy of GwSKey according to
     ↪   Gateway
111             claim(Gateway,Weakagree); //minimum agreement check between partners
     ↪   according to Gateway
112             claim(Gateway,Niagree); //validates the non-injective agreement according to
     ↪   Gateway
113             claim(Gateway,Nisynch); //validates the non-injective synchronization
     ↪   according to Gateway
114             claim(Gateway,Alive); //assures the Aliveness of Gateway

115
116         }
117     role Join {
118             fresh JoinNonce: Nonce;
119             fresh MHDRSrv: String;
120             fresh NetID: String;
121             fresh DevAddr: String;
122             fresh DLSettings: String;
123             fresh RxDelay: String;
124             fresh CFList: String;
125             fresh NonceList: String;
126             fresh JoinReqType: String;
127             fresh DevEUI: String;
```

```
128                 fresh GwEUI:String;
129                 var DevNonce: Nonce;
130                 var MHDRDev: String;
131                 var JoinEUI:String;
132                 var DevEUI:String;
133
134                 macro JoinNonceMIC = {MHDRDev,JoinEUI,DevEUI,DevNonce}k(Dev,Join);
135                 recv_1(Dev,Join,(MHDRDev,JoinEUI,DevEUI,DevNonce),JoinNonceMIC);
136                 send_2
↪ (Join,Dev,MHDRSrv,{{JoinNonce,NetID,DevAddr,DLSettings,RxDelay,CFList,MIC}dec}k(Dev,Join),MIC);
137
138                 /*Sends M1 to Gateway containing GwSessionKey*/
139                 macro GwSKeyGw = h(k(Dev,Join),Appkey,DevNonce,JoinNonce,Join);
140                 macro M1= {{GwEUI,GwSKeyGw,DevEUI}dec}k(Join,Gateway);
141                 send_m1gw(Join, Gateway,M1);
142
143                 not match (DevNonce, NonceList);
144                 macro NonceList=(NonceList, DevNonce);
145
146                 claim(Join,Running,Dev,JoinNonce); //checks that Join agrees with Dev on
↪ JoinNonce
147                 claim(Join,Alive); //assures the Aliveness of Join
148                 claim(Join,Weakagree); //minimum agreement check between partners according
↪ to Join
149                 claim(Join,Niagree); //validates the non-injective agreement according to
↪ Join
150                 claim(Join,Nisynch); //validates the non-injective synchronization according
↪ to Join
151                 claim(Join, SKR, AppSKey); //validates the secrecy of AppSKey according to
↪ Join
152                 claim(Join, SKR, GwSKeyGw); //validates the secrecy of GwSKeyGw according to
↪ Join
153           }
154
155       role NS{
156                 fresh JoinNonce: Nonce;
157                 fresh MHDRSrv: String;
158                 fresh NetID: String;
159                 fresh DevAddr: String;
160                 fresh DLSettings: String;
161                 fresh RxDelay: String;
162                 fresh CFList: String;
163                 fresh NonceList: String;
164                 fresh JoinReqType: String;
165                 fresh DevEUI: String;
166                 fresh GwEUI:String;
167                 var DevNonce: Nonce;
168                 fresh MHDRDev: String;
169                 fresh JoinEUI:String;
170                 var DevEUI:String;
171                 var DevAddr, FCtrl, FCnt, FOpts,FPort: String;
```

99

```
172                     var FRMPayload: String;

173

174                     recv_m3(Gateway,NS,{M1GW}k(Gateway,NS));

175

176                     /*Send M2 to AS*/
177                     send_m2(NS,AS,{M1GW}k(NS,AS));

178

179                     claim(NS,Alive); //assures the Aliveness of NS
180                     claim(NS,Weakagree); //minimum agreement check between partners according to
     ↪   NS
181                     claim(NS,Niagree); //validates the non-injective agreement according to NS
182                     claim(NS,Nisynch); //validates the non-injective synchronization according
     ↪   to NS

183

184             }
185         role AS{
186                     fresh JoinNonce: Nonce;
187                     fresh MHDRSrv: String;
188                     fresh NetID: String;
189                     fresh DevAddr: String;
190                     fresh DLSettings: String;
191                     fresh RxDelay: String;
192                     fresh CFList: String;
193                     fresh NonceList: String;
194                     fresh JoinReqType: String;
195                     fresh DevEUI: String;
196                     fresh GwEUI:String;
197                     var DevNonce: Nonce;
198                     fresh MHDRDev: String;
199                     fresh JoinEUI:String;
200                     var DevEUI:String;
201                     var DevAddr, FCtrl, FCnt, FOpts,FPort: String;
202                     var FRMPayload: String;

203

204                     recv_m2(NS,AS,{M1GW}k(NS,AS));
205                     macro MICPy'= cmac((Dev,AS),FRMPayload);
206                     match(MICPy',h(FRMPayload,(Dev,AS)));

207

208                     claim(AS,Alive); //assures the Aliveness of AS
209                     claim(AS,Weakagree); //minimum agreement check between partners according to
     ↪   AS

210                     claim(AS,Niagree); //validates the non-injective agreement according to AS
211                     claim(AS,Nisynch); //validates the non-injective synchronization according
     ↪   to AS

212

213             }

214

215

216     }
```

Code 5: SPDL Scyther verification Code for UMOAEG protocol

```
1    // The protocol is running between End Device (Dev), Gateway and Network Server/Join Server
 ↪    (Join).
2    // The predefined shared key (NwkKey) between End Device and Server is k(Dev,Join).
3    // The predefined shared key GrpKeyGrpId) between Gateway and Network Server is
 ↪    k(Gateway,NS)
4    // dec models a decryption function that is invertible by an encryption function (enc)
5    const dec: Function;
6    usertype String;
7    const pad01,pad02,pad03,pad04,pad05,pad06,pad07,pad08: String;
8    const pad09,pad10,pad11,pad12,pad13,pad14,pad15,pad16: String;
9    usertype TupleDB;
10
11   secret Appkey,NonceList,GwKey: String;
12
13   hashfunction h,cmac;
14
15   protocol UMOUAEG (Dev,Join,Gateway,AS,NS)
16   {
17         role Dev {
18               fresh DevNonce: Nonce;
19               fresh MHDRDev: String;
20               var MHDRSrv: String;
21               var JoinNonce: Nonce;
22               var NetID: String;
23               var DevAddr: String;
24               var DLSettings: String;
25               var RxDelay: String;
26               fresh CFList: String;
27               var JoinReqType: String;
28               fresh JoinEUI:String;
29               fresh DevEUI:String;
30               fresh FRMPayload:String;
31               fresh FPort,FOpts,FCnt,FCtrl:String;
32
33               macro JoinNonceMIC = {MHDRDev,JoinEUI,DevEUI,DevNonce}k(Dev,Join);
34               send_1(Dev,Join,(MHDRDev,JoinEUI,DevEUI,DevNonce),JoinNonceMIC);
35
36               macro JSIntKey={pad06,Dev,pad16 }k(Dev,Join);
37               macro
 ↪   MIC={JoinReqType,Join,DevNonce,MHDRSrv,JoinNonce,NetID,DevAddr,DLSettings,RxDelay,CFList}JSIntKey;
38               recv_2 (Join,Dev,
 ↪   MHDRSrv,{{JoinNonce,NetID,DevAddr,DLSettings,RxDelay,CFList,MIC}dec} k(Dev,Join),MIC);
39
40               /*LoRaWAN Session Derivation Keys*/
41               macro FNwkSIntKey={pad01,JoinNonce,Join,DevNonce,pad16}k(Dev,Join);
42               macro SNwkSIntKey={pad03,JoinNonce,Join,DevNonce,pad16}k(Dev,Join);
43               macro NwkSEncKey={pad04,JoinNonce,Join,DevNonce,pad16}k(Dev,Join);
44               macro JSEncKey={pad05,Dev,pad16}k(Dev,Join);
45               macro AppSKey={pad02,JoinNonce,Join,DevNonce, pad16 }Appkey;
46
```

```
47                      /*LoRaWAN Session Derivation Key for EN-GW*/
48                      macro GwSKeyD = h((k(Dev,Join),Appkey),DevNonce,JoinNonce,Join);
49
50                      /*Preparing MSG to Gw*/
51                      macro GwDevId = h(GwSKeyD,DevEUI);
52                      macro MDevice=(DevAddr,FCtrl,FCnt,FOpts,FPort,{FRMPayload}k(Dev,AS),MIC);
53                      macro MICPENi =cmac(GwSKeyD,MDevice,GwDevId,FCnt);
54                      macro MICPy= cmac(AppSKey,FRMPayload);
55                      macro M1GW=MDevice,MICPy,MICPENi,GwDevId;
56
57                      send_m1(Dev,Gateway,M1GW);
58
59                      claim(Dev,Running,Join,DevNonce); //checks that Dev agrees with Join on
   ↪   DevNonce
60                      claim(Dev,Alive); //assures the Aliveness of Dev
61                      claim(Dev,Weakagree); //minimum agreement check between partners according
   ↪   to Dev
62                      claim(Dev,Niagree); //validates the non-injective agreement according to
   ↪   Dev
63                      claim(Dev,Nisynch); //validates the non-injective synchronization according
   ↪   to Dev
64                      claim (Dev,SKR,AppSKey); //validates the secrecy of AppSKey according to
   ↪   Dev
65                      claim (Dev,SKR,GwSKeyD); //validates the secrecy of JSIntKey according to
   ↪   Dev
66              }
67          role Gateway{
68
69                      var JoinNonce: Nonce;
70                      fresh MHDRSrv: String;
71                      fresh NetID: String;
72                      fresh DevAddr: String;
73                      fresh DLSettings: String;
74                      fresh RxDelay: String;
75                      fresh CFList: String;
76                      fresh NonceList: String;
77                      fresh JoinReqType: String;
78                      fresh DevEUI: String;
79                      fresh GwEUI:String;
80                      var DevNonce: Nonce;
81                      fresh MHDRDev: String;
82                      fresh JoinEUI:String;
83                      var DevEUI:String;
84
85                      /*Receives  GwSessionKey from Join */
86                      macro GwSKeyGw = h(k(Dev,Join),Appkey,DevNonce,JoinNonce,Join);
87                      macro M1= {{GwEUI,GwSKeyGw,DevEUI}dec}k(NS,Gateway);
88                      recv_m1gw(Join, Gateway,M1);
89
90                      /*Receives Uplink Message from End Node*/
91                      fresh Mtype,EJP,Major: String;
```

```
 92                    var DevAddr, FCtrl, FCnt, FOpts,FPort: String;
 93                    var FRMPayload: String;
 94
 95                    macro M1GW=MDevice,MICPy,MICPENi,GwDevId;
 96
 97                    recv_m1(Dev,Gateway,M1GW);
 98
 99                    match((GwDevId,GwSKeyGw),TupleDB);
100                    macro GwDB=(GwDB, (GwDevId,GwSKeyGw));
101
102                    macro MICPENi'=cmac(GwSKeyD,MDevice,GwDevId,FCnt);
103                    not match(MICPENi',MICPENi);
104
105                    /*Sends M2 to NS*/
106                    macro M2={GwDevId,GwEUI,DevEUI}k(Gateway,NS);
107                    send_m2(Gateway,NS,M2);
108
109                    /*Receives M6 from NS*/
110                    macro M6={GwDevId,GwEUI,GwSKeyGw}k(NS,Gateway);
111                    recv_m6(NS,Gateway,M6);
112
113                    claim (Gateway,Secret,k(NS,Gateway)); //validates the secrecy of GrpKeyGrpId
    ↪  according to Gateway
114                    claim (Gateway,SKR,GwSKeyGw); //validates the secrecy of GwSKey according to
    ↪  Gateway
115                    claim(Gateway,Weakagree); //minimum agreement check between partners
    ↪  according to Gateway
116                    claim(Gateway,Niagree); //validates the non-injective agreement according to
    ↪  Gateway
117                    claim(Gateway,Nisynch); //validates the non-injective synchronization
    ↪  according to Gateway
118                    claim(Gateway,Alive); //assures the Aliveness of Gateway
119
120            }
121        role Join {
122                    fresh JoinNonce: Nonce;
123                    fresh MHDRSrv: String;
124                    fresh NetID: String;
125                    fresh DevAddr: String;
126                    fresh DLSettings: String;
127                    fresh RxDelay: String;
128                    fresh CFList: String;
129                    fresh NonceList,DeviceParms: String;
130                    fresh JoinReqType: String;
131                    fresh DevEUI: String;
132                    fresh GwEUI:String;
133                    var DevNonce: Nonce;
134                    var MHDRDev: String;
135                    var JoinEUI:String;
136                    var DevEUI:String;
137
```

```
138                        macro JoinNonceMIC = {MHDRDev,JoinEUI,DevEUI,DevNonce}k(Dev,Join);
139                        recv_1(Dev,Join,(MHDRDev,JoinEUI,DevEUI,DevNonce),JoinNonceMIC);
140                        send_2
↪  (Join,Dev,MHDRSrv,{{JoinNonce,NetID,DevAddr,DLSettings,RxDelay,CFList,MIC}dec}k(Dev,Join),MIC);
141
142                        /*Sends M1 to Gateway containing GwSessionKey*/
143                        macro GwSKeyGw = h(k(Dev,Join),Appkey,DevNonce,JoinNonce,Join);
144                        macro M1= {{GwEUI,GwSKeyGw,DevEUI}dec}k(NS,Gateway);
145                        send_m1gw(Join, Gateway,M1);
146
147                        /*Receives request from NS to provide GwSKey*/
148                        macro M3={GwDevId,GwEUI,DevEUI}sk(Join);
149                        recv_m3(NS,Join,M3);
150                        match(DeviceParms,(GwDevId,GwEUI,DevEUI));
151                        send_m4(Join,NS,{GwDevId,GwEUI,GwSKeyGw}sk(NS));
152
153                        not match (DevNonce, NonceList);
154                        macro NonceList=(NonceList, DevNonce);
155                        macro DeviceParams = (DeviceParms,(GwDevId,GwEUI,DevEUI));
156
157
158
159                        claim(Join,Running,Dev,JoinNonce); //checks that Join agrees with Dev on
↪  JoinNonce
160                        claim(Join,Alive); //assures the Aliveness of Join
161                        claim(Join,Weakagree); //minimum agreement check between partners according
↪  to Join
162                        claim(Join,Niagree); //validates the non-injective agreement according to
↪  Join
163                        claim(Join,Nisynch); //validates the non-injective synchronization according
↪  to Join
164                        claim(Join, SKR, AppSKey); //validates the secrecy of AppSKey according to
↪  Join
165                        claim(Join, SKR, GwSKeyGw); //validates the secrecy of GwSKeyGw according to
↪  Join
166                }
167
168        role NS{
169                fresh JoinNonce: Nonce;
170                fresh MHDRSrv: String;
171                fresh NetID: String;
172                fresh DevAddr: String;
173                fresh DLSettings: String;
174                fresh RxDelay: String;
175                fresh CFList: String;
176                fresh NonceList: String;
177                fresh JoinReqType: String;
178                fresh DevEUI: String;
179                fresh GwEUI:String;
180                var DevNonce: Nonce;
181                fresh MHDRDev: String;
```

```
182             fresh JoinEUI:String;
183             var DevEUI:String;
184             fresh DevAddr, FCtrl, FCnt, FOpts,FPort: String;
185             fresh FRMPayload: String;
186
187             recv_m2(Gateway,NS,{GwDevId,GwEUI,DevEUI}k(Gateway,NS));
188
189             /*Send M3 to JS*/
190             macro M3={GwDevId,GwEUI,DevEUI}sk(Join);
191             send_m3(NS,Join,M3);
192
193             /*Receive M4 from JS*/
194             recv_m4(Join,NS,{GwDevId,GwEUI,GwSKeyGw}sk(NS));
195             macro M6={GwDevId,GwEUI,GwSKeyGw}k(NS,Gateway);
196             send_m6(NS,Gateway,M6);
197
198             send_mp1(NS,AS,{M1GW}k(NS,AS));
199
200             claim(NS,Alive); //assures the Aliveness of NS
201             claim(NS,Weakagree); //minimum agreement check between partners according to
   ↪ NS
202             claim(NS,Niagree); //validates the non-injective agreement according to NS
203             claim(NS,Nisynch); //validates the non-injective synchronization according
   ↪ to NS
204
205         }
206     role AS{
207             fresh JoinNonce: Nonce;
208             fresh MHDRSrv: String;
209             fresh NetID: String;
210             fresh DevAddr: String;
211             fresh DLSettings: String;
212             fresh RxDelay: String;
213             fresh CFList: String;
214             fresh NonceList: String;
215             fresh JoinReqType: String;
216             fresh DevEUI: String;
217             fresh GwEUI:String;
218             var DevNonce: Nonce;
219             fresh MHDRDev: String;
220             fresh JoinEUI:String;
221             var DevEUI:String;
222             var DevAddr, FCtrl, FCnt, FOpts,FPort: String;
223             var FRMPayload: String;
224
225             recv_mp1(NS,AS,{M1GW}k(NS,AS));
226             macro MICPy'= cmac((Dev,AS),FRMPayload);
227             match(MICPy',h(FRMPayload,(Dev,AS)));
228
229             claim(AS,Alive); //assures the Aliveness of AS
230             claim(AS,Weakagree); //minimum agreement check between partners according to
   ↪ AS
```

105

```
231              claim(AS,Niagree); //validates the non-injective agreement according to AS
232              claim(AS,Nisynch); //validates the non-injective synchronization according
↪    to AS
233
234        }
235
236
237    }
```

Code 6: SPDL Scyther verification Code for UMOUAEG protocol