Expertise
and insight
for the future

Abel Onditi

Implementation of LoRaWAN for Metropolia University of Applied Sciences

Metropolia University of Applied Sciences

Bachelor of Engineering

Information and Communication Technology

Bachelor's Thesis

2 February 2019

Author Title Number of Pages Date	Abel Onditi Implementation of LoRaWAN for Metropolia University of Applied Sciences 40 pages + 2 appendices 23 September 2019
Degree	Bachelor of Engineering
Degree Programme	Information and Communication Technology
Professional Major	Smart Systems
Instructors	Keijo Länsikunnas, Senior Lecturer

The Internet of Things (IoT) field and the development of new IoT devices and solutions are growing rapidly. Many A lot of IoT related projects are also being conducted in Metropolia University of Applied Science.

However, the IoT end-devices are often low-powered devices and as such, they often use communication protocols that require low power to function. One communication technology that has gained a lot of popularity in the field of low power data transport is LoRa and the higher-level implementation of the technology, LoRaWAN (LoRa wide area network) is already widely used in most countries around the world.

The primary goal of this thesis was to set up a private LoRaWAN, a low power, wide area network, on one or more Metropolia UAS campuses which could be used by Metropolia students in their projects and studies. A secondary goal of the thesis was to also implement an authentication method to the network which utilizes Metropolia's user database to authenticate and authorize the users. Another goal was to test the network using possibly several different evaluation boards to see which board would require the least amount of work to use the network with.

The network was implemented using LoRa Server, an open-source LoRaWAN network server, to create the backend and LORIX One LoRaWAN gateways to create the physical network. The LoRa Server was implemented on CentOS 7 running in Metropolia's educloud-platform.

Keywords	LoRa, LoRaWAN, IoT, Network

Tekijä Otsikko Sivumäärä Aika	Abel Onditi LoRaWAN-verkon toteuttaminen Metropolian Ammattikorkeakoululle 40 sivua + 2 liitettä 23.9.2019
Tutkinto	Insinööri (AMK)
Tutkinto-ohjelma	Tieto- ja viestintätekniikka
Ammatillinen pääaine	Älykkäät järjestelmät
Ohjaajat	Lehtori Keijo Länsikunnas

Esineiden internet on nopeasti kasvava ala ja tuotekehitys sen parissa lisääntyy jatkuvasti. Myös Metropolian Ammattikorkeakoulussa tehdään useita esineiden internetiin liittyviä projekteja.

Esineiden internetiin liittyvät laitteet ovat hyvin usein vähävirtaisia, joten ne myös usein hyödyntävät energiatehokkaita kommunikointimenetelmiä. Energiatehokkaiden kommunikointimenetelmien parissa erityistä huomiota on saanut LoRa-teknologia, ja sen korkeamman tason ilmentymä LoRaWAN on jo maailmanlaajuisesti hyödynnettynä esineiden internetin applikaatioissa.

Tämä insinöörityön päätavoitteena oli toteuttaa yksityinen LoRaWAN-verkko yhdelle tai useammalle Metropolian AMK:n kampukselle, jota opiskelijat voivat hyödyntää projekteissaan ja opinnoissaan. Sekundäärinen tavoite projektille, oli toteuttaa verkko siten, että siihen tunnistautuminen käyttäjänä onnistuu Metropolian yleisiä kirjautumistunnuksia käyttäen. Tämän lisäksi tavoitteena oli testata verkkoa eri kehityslautoja käyttäen, tarkoituksena löytää mahdollisimman helposti käyttöönotettava kehityslauta.

LoRaWAN verkko toteutettiin hyödyntämällä avoimen lähdekoodin alustaa, LoRa Serveriä. Yhdyskäytävänä päätelaitteen ja verkon välillä käytettiin LORIX One yhdyskäytävää. Palvelin, jolle LoRa Server asennettiin, toimii CentOS 7 pohjaisella virtuaalikoneella Metropolian educloud-pilvipalvelussa.

Avainsanat	LoRa, LoRaWAN, IoT, Verkot

Contents

List of Abbreviations

ABP	Activation by Personalization. An activation scheme, where an end-device is activated on a network using pre-defined hard-coded credentials.
AES	Advanced Encryption Standard. A data encryption specification standardized by NIST.
CMAC	Cipher-based Message Authentication Code. Ambiguous to OMAC
CTR	Counter. A mode that turns a block cipher into a stream cipher.
FUOTA	Firmware-update-over-the-air.
gRPC	An open source remote procedure call system. Among other things, it simplifies the complex remote calls between a server and a client.
JSON	JavaScript Object Notation. An open-standard lightweight data-interchange format.
LDAP	Lightweight Directory Access Protocol for accessing directory services between a client and a server.
LoRa	A spread spectrum modulation used in the long range, low power LoRa Technology in the field of Internet of Things.
LoRaWAN	LoRa Wide Area Networking protocol. A protocol that is used to connect low-power devices to a larger network.
MAC	Message authentication code is used to authenticate a message. It is often substituted by MIC to distinguish it from media access control (MAC).
MAC layer	Media access control layer. A sublayer of the data link layer of OSI model, responsible for moving data packets from one network interface card to another.
MIC	Message integrity code. Often used interchangeably with MAC.

MQTT	Message Queuing Telemetry Transport. A publish-subscribe-based lightweight messaging protocol.
NIST	National Institute of Standards and Technology. An American institute operating that aims to promote innovation and industrial competitiveness.
OMAC	One-key MAC. A message authentication code constructed from a block cipher.
OS	Operating system. A system software that manages all the hardware and other application programs in a computer.
OSI model	Open Systems Interconnection model. A model that standardizes the communication between computers.
OTAA	Over-the-air activation. An activation scheme, where an end-device joins the network by performing a join-procedure.
PostgreSQL	A free, open-source relational database management system.
REST	Representational State Transfer. An application programming interface protocol that defines a set of rules to be used when creating web services.
SSH	Secure Shell. A protocol that allows a secure connection to a computer over an unsecure network.
TCP	Transmission Control Protocol. A standard that defines how two hosts connect to each other and exchange data.
TLS	Transport Layer Security. An encryption protocol that secures communication on the internet.
TTN	The Things Network. A project that aims to build a global network for the Internet of Things.
TTI	The Things Industries. An IoT company that provides solutions and assistance for other companies in their LoRaWAN implementations.

UDP	User Datagram Protocol. A communication protocol used mainly to establish low-latency and loss-tolerating communications between hosts. Due to its simplicity, the protocol is prone to packet losses, errors and duplication.
VM	Virtual Machine. A software that emulates the behavior of a separate computer.
XOR	Exclusive OR. A logical operation that outputs true if the inputs differ.

1 Introduction

This thesis examines ways of setting up a private LoRaWAN network with user authentication. LoRaWAN is a higher-level network architecture utilizing LoRa protocol and adding on top of it a structure with which IoT devices can efficiently communicate over to end-user interfaces. LoRa is a fairly recently emerged low power, long range wireless communication protocol developed to meet the needs of the rapidly expanding IoT infrastructure.

The thesis was carried out for Metropolia University of Applied Sciences with a fundamental purpose of enabling easier and faster development of devices utilizing LoRa technology in the Metropolia UAS premises. A secondary goal of the thesis was to enable an authentication method on the network which uses Metropolia user credentials as a valid way of authenticating.

This thesis is divided into six sections. The first section introduces the topic of the thesis along with a brief explanation of the technologies utilized in it. The second section describes in detail the LoRaWAN network protocol with all its components while section three introduces in detail the network server project which is used in the thesis to set up a private LoRaWAN.

The fourth section gives an explanation on how the thesis was planned and which meetings were held and which discussions were had. Section five focuses on describing the steps taken in the development and testing of the network. In section six, the conclusion of the thesis is discussed, and possible future steps are considered.

2 LoRaWAN network protocol

This chapter describes the LoRaWAN network protocol in detail. LoRaWAN is considered to be equivalent to the data link and network layer (layers 2 and 3) of the OSI model and it operates on top of LoRa modulation technology which is the equivalent of the physical layer (layer 1) of the OSI model. If not otherwise specified, this chapter describes LoRaWAN v1.1, which has some major upgrades in security and otherwise over the first stable version LoRaWAN v1.0.

2.1 Overview

LoRaWAN network usually uses the star-of-stars topology where each end-device is connected to one or more gateways over a single-hop LoRa connection. The network itself usually consists of a cluster of servers that often reside at the same location in the network. The servers can, however, also be distributed across the LoRaWAN network.

The usual way of implementing a LoRaWAN network is shown in the following figure:

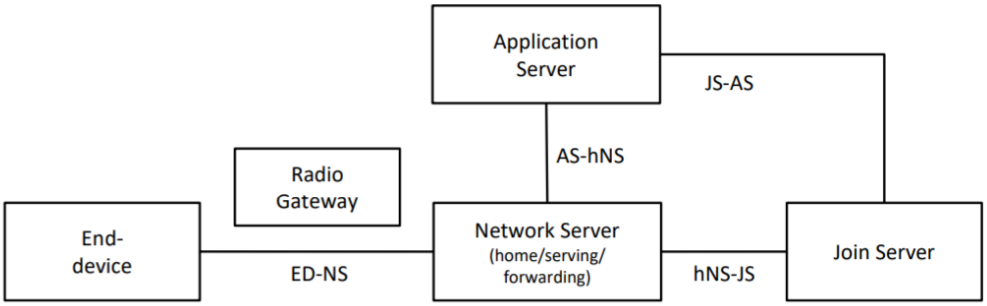


Figure 1. Example of a LoRaWan network device at home. [2, p. 8.]

The message flow of the network is bidirectional. In up-link messages, the end-device transmits a message over the LoRa network. One or more gateways relay this message to a Network Server over a secured standard IP connection. The Network Server then routes it to the correct Application Server. The Application Server provides the end user interface, such as a mobile application or a web page.

In case the Network Server receives a Join-request from an end-device initiating the connection, it forwards the end-device information to a Join Server. The Join Server

takes care of the Over-The-Air (OTA) -activation of the end-device. It associates the end-device with its Network Server and some specific Application Server. This scheme known as Join Procedure is described in detail in [2, p. 8.]

The communication between the end-devices and the gateways is spread out on multiple frequency channels and data rates. Any end-device may use any desired channel or data rate as long as they switch their channel on each transmission randomly and respect the maximum transmit duty cycle and duration dictated by the local regulations.

As specified in LoRaWAN 1.1 Specification by LoRa Alliance [1, p. 8.], LoRaWAN uses symmetric cryptography with session keys to ensure a secure radio transmission. This is discussed in more detail in chapter 2.4.1. The session keys are derived from unique device root keys. Both of the root keys and the session keys are stored in a Join Server.

2.2 LoRaWAN components

All LoRaWAN networks fundamentally consist of the same building blocks: the end-device sending or receiving data over LoRa radio frequencies, one or more gateways relaying the data between the end device and the server, and finally the back-end servers handling the data.

The different protocols over which the LoRaWAN connects, are illustrated in the following figure:

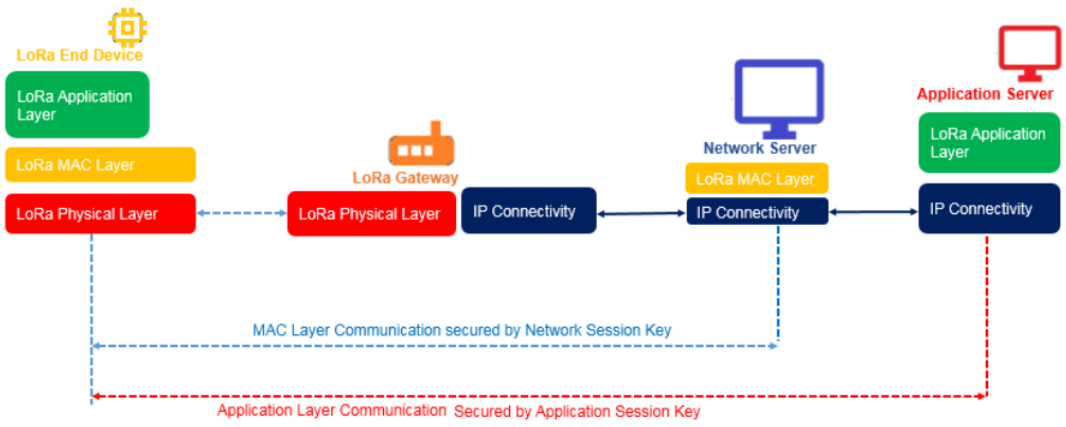


Figure 2. LoRaWAN network protocols [7].

2.2.1 End-device

The end-devices, also known as motes, along with the gateways are the components of the network that communicate over radio frequencies specified by the LoRa Alliance.

All end-devices have an application layer and a MAC layer. The application layer is connected to an Application Server over the network, while the MAC layer is terminated at the Network Server. The MAC layer communication is secured by a Network Session Key, NwkSKey and the Application Layer communication by an Application Session Key, AppSKey.

End-devices can be activated on the network in two ways. Activation-by-Personalization (ABP) activates the end-device to a specific network with preconfigured server information. In this method, the end-device will use the same sessions keys during its lifetime.

Over-the-Air (OTA) activation makes the end-device perform a Join Procedure with the corresponding Join Server to get activated in the network as specified by [2, p. 12]. This is a more flexible and usually the preferred approach to device activation as it has less security implications and allows for re-keying when compromised according to [10, p. 9.].

LoRaWAN end-devices can be divided into three classes based on their functionality: Class A, B and C. All the end-devices allow for bi-directional communication. The difference between the classes lie in the way they receive downlink messages.

Class A end-devices are the very basic, lowest power devices. The downlink messages in this class are only received in two short windows right after an uplink message has been transmitted. The following figure by The Things Network illustrates how a server can respond in either the first window, or the second one or in neither one, but not in both of the receive windows.

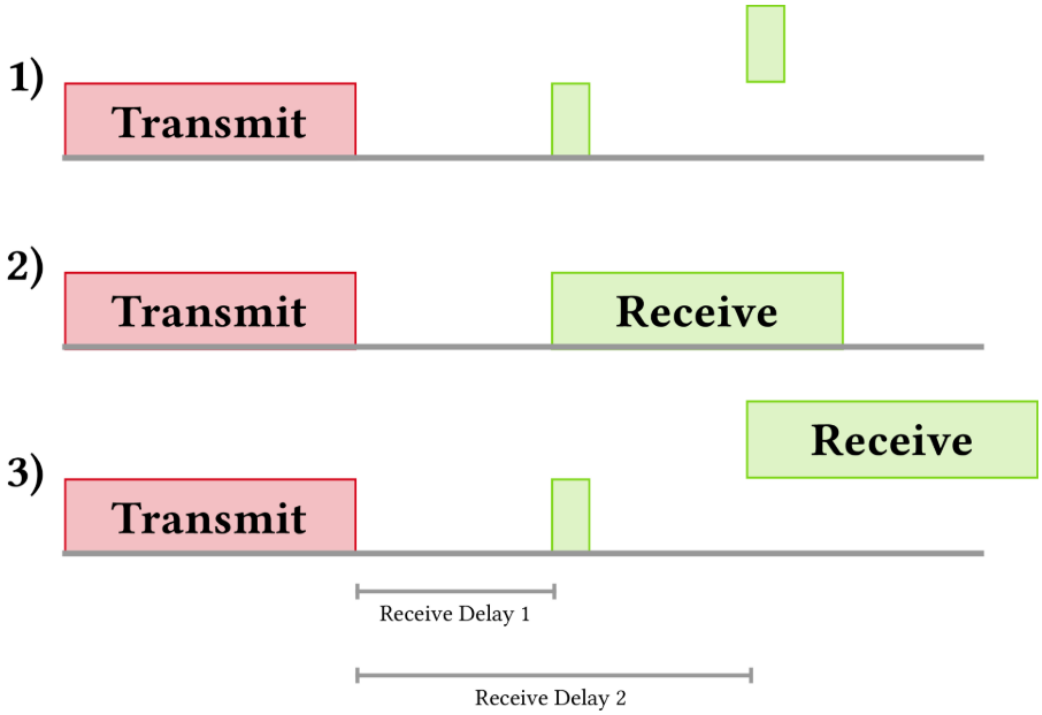


Figure 3. Transmit/receive scenarios of a Class A end-device [3].

The Class B devices allow for flexibility in the receive scenarios by enabling a scheduled receive window. These devices can open one or more extra receive windows allowing the gateway to send time-synchronized beacons to the end-device.

The Class C devices are the most power consuming with the lowest latency. Devices of this class always have their receive window open when they are not transmitting.

2.2.2 Gateway

Gateways function as a bridge between the end-device and the Network Server of the LoRaWAN network. Different gateways can vary in complexity of the software running on them, but all gateways have at least the same fundamental functions.

The gateways are routers with a LoRa concentrator running on them. The concentrator is capable of decoding LoRa modulation variants on different frequencies in parallel utilizing a LoRa demodulator. The gateways use higher bandwidth networks to relay the data over to the Network Server as described in [4] and [5].

The very basic gateway on a minimal firmware only decodes incoming data and relays it, thus acting only as a packet forwarder. Gateways can, however, run on an operating system, which allows them to have more sophisticated data handling software running on them, such as authentication and authorization. Such a gateway is used in this project.

2.2.3 Network Server

The Network Server is at the center of the star topology and is responsible for some of the core features of LoRaWAN. Its functionality varies depending on the architecture of the network. The following figure shows a standard non-roaming implementation of a LoRaWAN. The gateway–end-device connection is simplified to simply being an end-device.

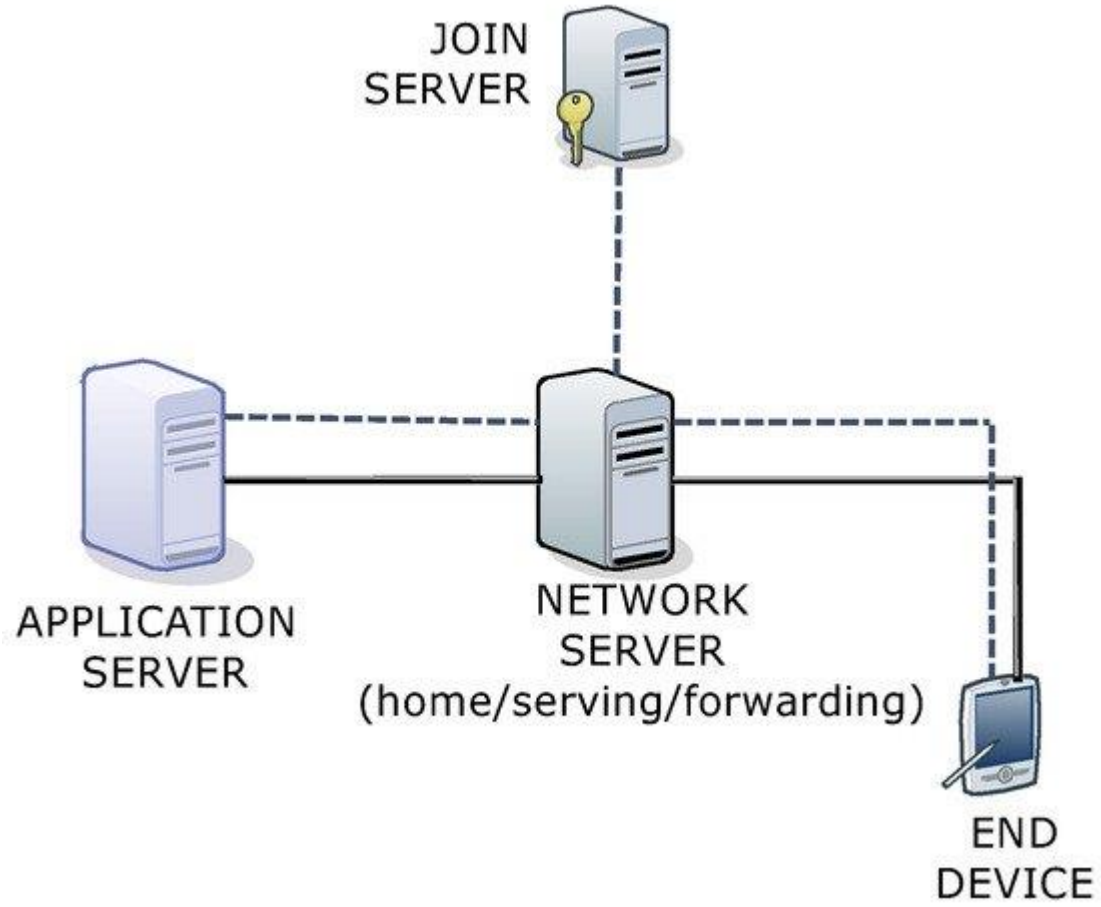


Figure 4. Simple LoRaWAN 1.1 architecture [6, p. 27].

The Network Server checks the end-device address. It forwards all the uplink messages sent by end-devices to their corresponding Application Server and queues the downlink messages originating from the Application Server to an end-device. It also forwards the messages between an end-device and a Join Server in case of an OTA end-device activation. [2, p. 9.]

2.2.4 Application Server

The Application Server is responsible for the interface accessed by the end-user. It handles the uplink messages from end-devices and generates the downlink payloads to end-devices as described in [2, p. 11.]. The Application Server used in this project provides a web-interface for the end-user. It can also be interfaced over RESTful or gRPC APIs.

2.2.5 Join Server

The Join Server handles all the OTA end-device activations. It holds all the necessary information about the end-device in order to handle the activation process.

The Join Server sends the Network Session Key of the end-device to the Network Server and the Application Session Key to the corresponding Application Server [2, p. 10]. By doing this, it makes the corresponding servers acknowledge the presence of the end-device, thus allowing the end-device to connect to the servers.

In addition to the end-device, the Join Server is the only place the AppKey and the NwkKey of the end-device are stored in addition to the end-device itself [2, p. 10].

2.3 Roaming

The LoRaWAN supports roaming and there are two types of roaming schemes for the network, Passive Roaming and Handover Roaming.

In Passive Roaming, the end-device connected to a gateway in one LoRaWAN network can use the services of a Network Server in another network. In this scheme, the foreign Network Server is referred to as the Serving Network Server, and it handles the LoRa session and the MAC-layer control of the end-device. The Network Server that is in the same network as the end-device is called the Forwarding Network Server, and as the name suggests, its responsibility is to forward the frames between the end-device and the Serving Network Server.

The Forwarding Network Server can be configured to be stateful or stateless. When stateful, the Network Server creates context for end-devices and utilizes that context to process all subsequent packets of the corresponding end-devices. If the Network Server is configured to be stateless, it will handle each packet independent of each other. [2, p. 22.]

In the Handover Roaming scheme, the home Network Server of an end-device can transfer MAC-control from one Network Server to another. In this scheme, the home Network Server handing over the control will stay the same during the whole session. These two

roaming schemes can also be combined to create a more flexible network as seen in the figure below:

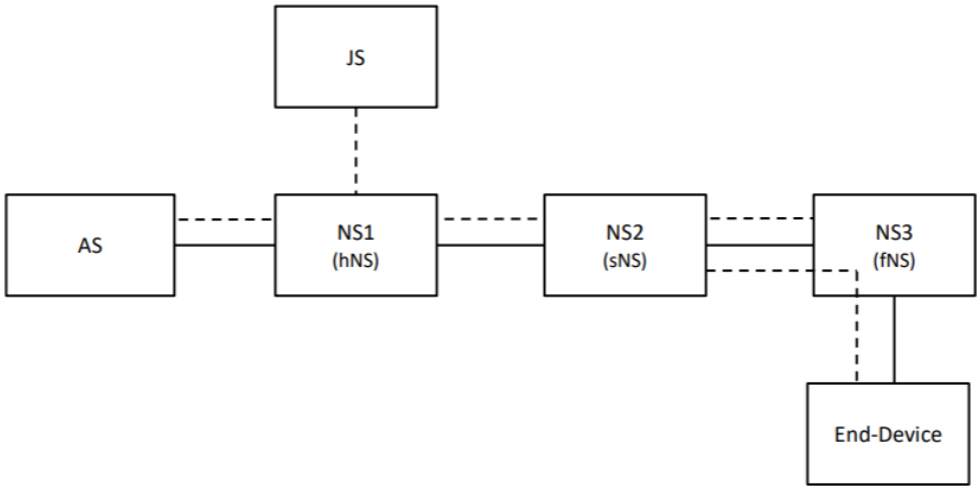


Figure 5. Example use-case of Handover and Passive Roaming together [2, p. 23].

In this thesis, no roaming functionality is implemented not only due to it being obsolete at this time, but also due to the LoRa Server not having support for roaming at the time of writing.

2.4 LoRaWAN Cryptography

LoRaWAN uses a cryptography based on AES encryption algorithm with several operation modes. The encryption of payloads is carried out with AES-CTR and the Message Integrity Code (MIC) is computed utilizing AES-CMAC. These aforementioned abbreviations will be discussed in a bit more detail in this chapter.

2.4.1 AES encryption algorithm

AES (Advanced Encryption Standard) is a specification for data encryption by National Institute of Standards and Technology (NIST). AES is a subset of a block cipher, Rijndael, which was the winner of the AES selection process conducted by NIST to come up with a successor for the Data Encryption Standard (DES). AES differs from Rijndael

only by its number of supported block lengths and cipher key lengths as described in [19, p. 31].

AES is a is symmetric cipher which means that it uses the same key to encrypt and decrypt the data. This key must be known both by the sender and the receiver. The data to be encrypted is first placed in a byte array. This array is then manipulated with a transformation specific to the standard several times. The amount of transformation rotations varies from 10 to 14 depending on the AES key size (128-, 196- or 256-bits). [18.]

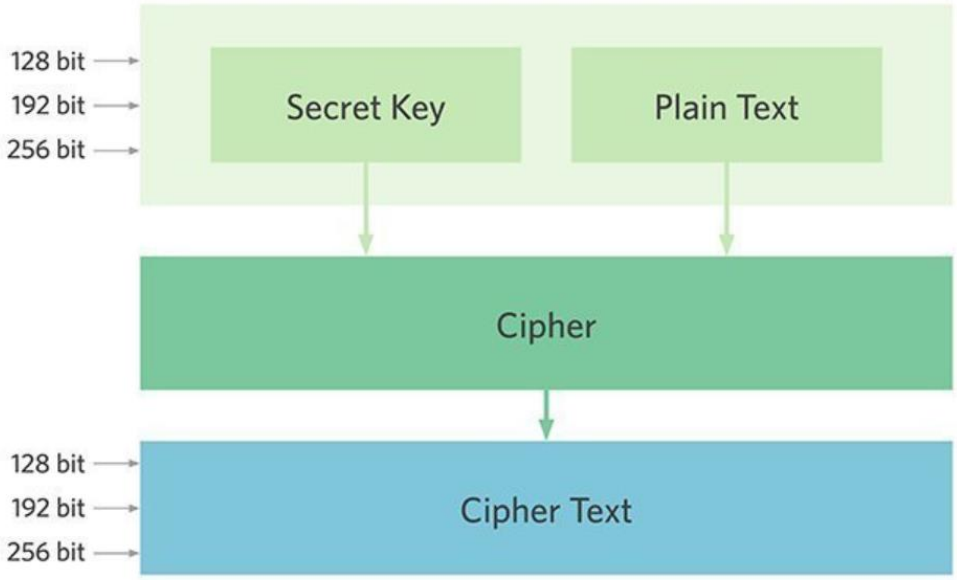


Figure 6. AES Design [18].

In the Cipher-block of the illustration above, the actual algorithm is applied. A single rotation of transformations consists of four steps:

- Substitution: The array of bytes is substituted with a non-linear array with as little input-output correlation and difference propagation probability as possible.
- Shift data rows: The data rows of the array are cyclically shifted with each of the rows having a different shift offset.
- Mix columns: The columns of the data array are mixed with efficiency.

- Key addition: Lastly a XOR-operation is performed on each column of the data with the encryption key [18]. A XOR- or exclusive or -operation is a logical operation that outputs true when the inputs are different, i.e. *1 XOR 0 = TRUE.*

General design guidelines for each of these steps are defined by Daemen and Rijmen in [19, p. 34-41].

2.4.2 AES-CTR

AES-CTR (Counter-Mode) is an AES operation mode defined by NIST. A total of 5 different AES operation modes are defined by NIST at the time of writing out of which AES-CTR is considered to be the best according to [20] and numerous other sources.

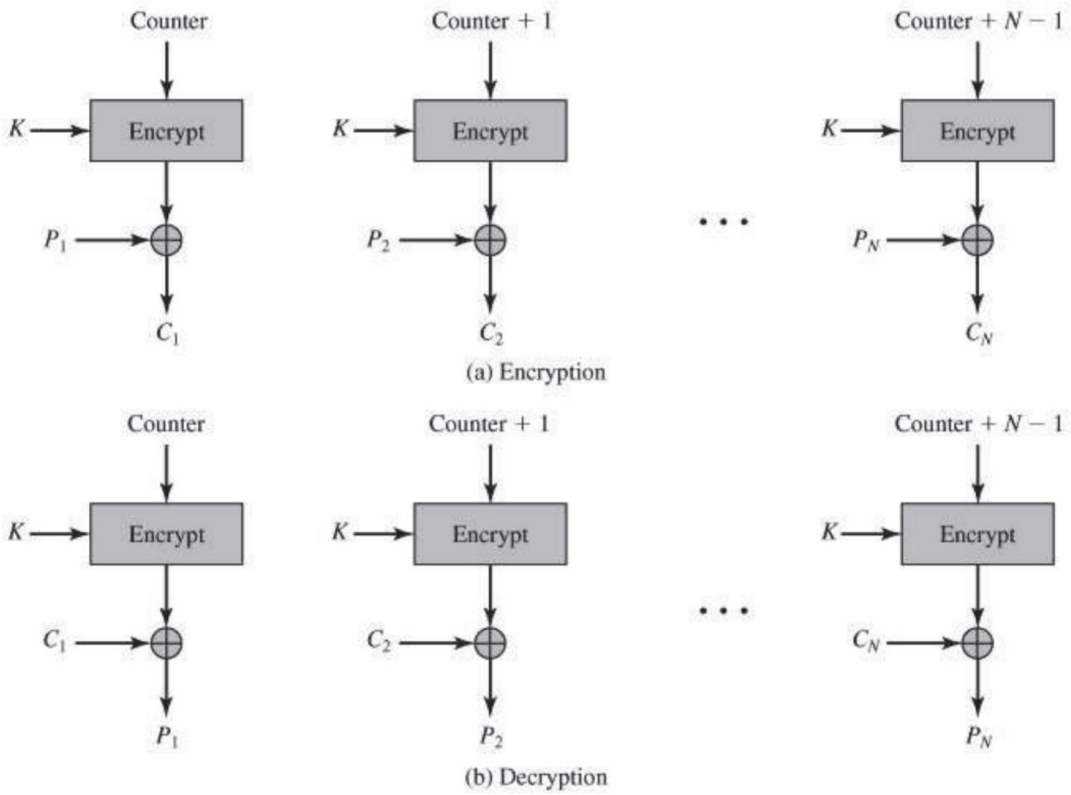


Figure 7. CTR mode of operation scheme [20].

In the figure above the operation of the CTR mode is described. To define the components in the figure: (*Counter, Counter +1, …, Counter + N - 1)* is the input or initialization vector of the block cipher in AES-CTR mode. In the scheme above, the counter is a

randomly generated, non-repeatable number that is on the order of 96 bits, according to [20]. In [21, p.1] it is, however, pointed out that this is just one of many ways to implement this encryption scheme. *K* is the encryption key used with the block cipher and it is the same for all the blocks. A XOR-operation is performed on the output of the block cipher using a plaintext message ($P_1, P_2, \ldots, P_N$). This message is the actual data to be encrypted.

The output of the XOR-operation ($C_1, C_2, \ldots, C_N$) and the Counter are the parts that are typically transmitted to the decrypting end. The receiving end knows the encryption key K and performs the same exact encryption process using the Counter it received. The only difference is that the XOR-operation is performed on the output of the block cipher using the output of the XOR-operation performed in the encryption process ($C_1, C_2, \ldots, C_N$). The output of this operation is the original message in plain text.

AES-CTR provides a simple, yet powerful encryption method which at the time of writing is still unbreakable if implemented correctly. It does not, however, provide any form of authentication.

2.4.3 AES-CMAC

For authentication, LoRaWAN uses AES-CMAC. This is an authentication protocol fundamentally based on a technique called *cipher block chaining message authentication code* (CBC-MAC).

CBC is a method where a block of a message is encrypted with a block cipher algorithm. This encrypted block is then forwarded as a parameter to encrypt the next block of the message. Hence the name *cipher block chaining.* Because each block relies on the proper encryption of the previous block, this method effectively ensures the authenticity of the message, because if one the encryption of a single block along the way is somehow altered, the resulting message will change in an unpredictable way.

CBC-MAC is almost identical to CBC except for a couple of details. CBC-MAC uses a static initialization vector, usually consisting of zeroes, and only outputs one result as shown in the figure below.

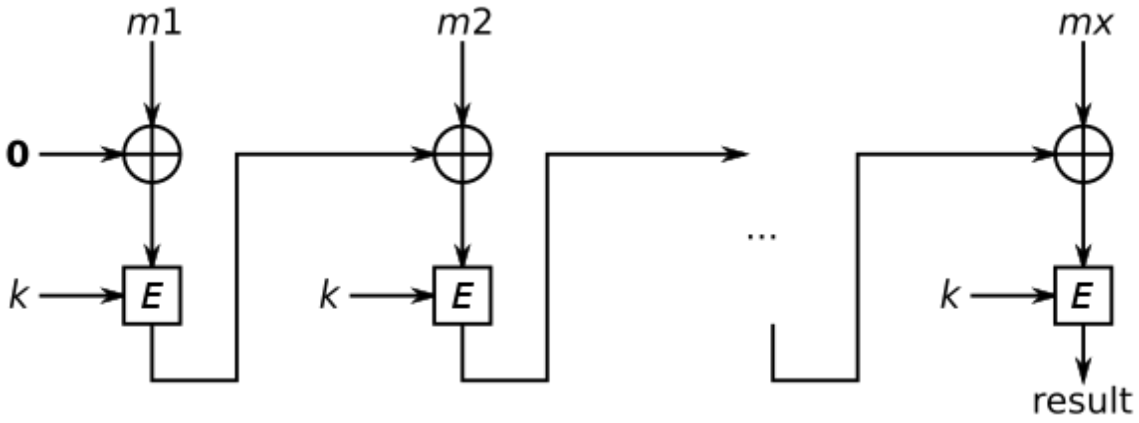


Figure 8. CBC-MAC algorithm [22].

CBC-MAC and CBC are, however, only secure when the message is of fixed length. For this flaw a several fixes were introduced, one of which is CMAC, also known as OMAC (One-key MAC). CMAC improves the CBC-MAC by adding an extra layer of security to the last XOR-operation of the block chain [23, p.3].

AES-CMAC is thus simply CMAC which uses AES encryption as its block cipher and performs the XOR-operation on the last block with an encryption key that is generated utilizing AES [24].

2.4.4 LoRaWAN packet

Each LoRaWAN packet is secured utilizing all the previously mentioned encryption schemes. The figure below illustrates the encryption scheme.

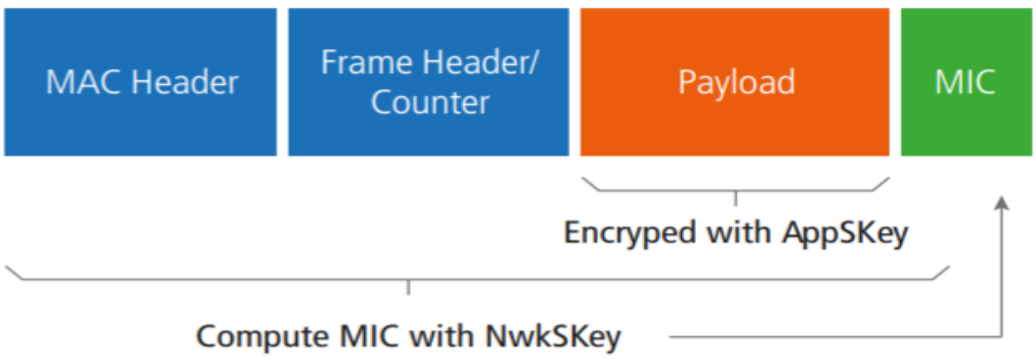


Figure 9. LoRaWAN packet structure [17].

The payload within the packet is encrypted by AES-CTR. As described in [1, p. 25] , the encryption key used in the AES-CTR encryption process is either AppSKey or NwkSKey depending on the endpoints of the communication. The different LoRaWAN keys are discussed in detail in chapter 2.6.

The message integrity code (MIC) is calculated over all the fields of the packet frame using the NwkSKey. This encryption is carried out by AES-CMAC. The MIC comprises the encrypted payload and the encrypted frame, which means that both the integrity and the authenticity of the frame can be verified by simply checking the MIC.

2.5 LoRaWAN security

The security aspects of the initial release of the LoRaWAN specification 1.0 have been reviewed on several papers and a lot of security faults were found. Most of these faults were, however, addressed in the newer version 1.1 of the LoRaWAN specification. In this chapter the security aspects of the updated LoRaWAN version 1.1 will be viewed.

2.5.1 Security challenges

LoRaWAN has several security challenges even in the updated version 1.1. Depending on the size of the network and the number of functionalities implemented, the amount of challenges varies.

Merely the sheer amount of different type of servers required to set up a fully functioning network makes it compilated to manage and to ensure its security. However, according to [10] the weakest link of LoRaWAN are its gateways. Because there are typically only a few gateways deployed to cover a rather large area, if a gateway was captured or tampered with, it could easily destroy a connection of quite many end-devices.

There are security implications regarding the security key distribution in LoRaWAN. As mentioned in more detail in section 2.2, the keys can be distributed by two different mechanisms, ABP or OTAA. While ABP is more straightforward and makes deployment slightly easier, it has a clear disadvantage over OTAA: ABP does not allow for re-keying, which means that if a device is compromised, there is very little to be done to fix the

situation. As mentioned in [10 p. 9], the root keys, AppKey and NwkKey, need to be securely stored. For this, it is also pointed out in [10] that a hardware-secure way of using tamper-proof or -resistant memory elements needs to be used.

Even though the possibility for re-keying is introduced for OTAA devices, no mechanism to perform a scheduled and organized re-keying is given which means that it is up to the provider to implement a re-keying scheme. Neglecting this may reduce security considerably.

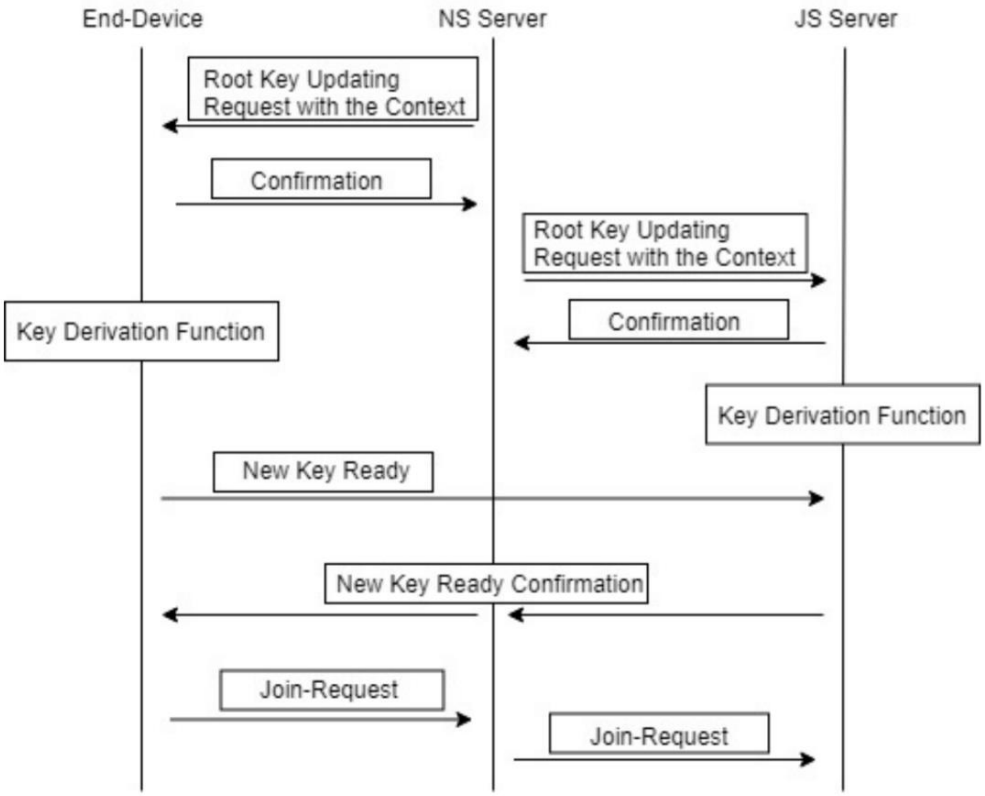


Figure 10. Root key re-keying scheme [15, p.9].

The figure above illustrates a possible re-keying scheme for the root key. In this scheme the update routine is carried out between the end-device and the Join Server over the Network Server, and the Application Server is not involved at all.

The re-keying challenge appears also in a situation where the JoinEUI of a Join Server requires modification. If the JoinEUI is modified, all the end-devices connected through

that Join Server will become useless, due there being currently no scheme for re-distributing a new JoinEUI to all the end-devices.

In this project, the OTAA method is used and the keys are stored in a PostgreSQL database protected by a strong password. The server in which the database is located resides in Metropolia's local network, and is only accessible over password protected SSH connection originating from within Metropolia's local network. Due to all these restrictions, the root keys can be considered safely stored. No re-keying scheme is implemented in this project.

In addition to the re-keying scheme, several other security aspects are left for the network provider to be solved. The exit procedure of end-devices is one of these. This procedure is only vaguely described in the description as mentioned in [10, p, 10], but is strongly suggested to be thoroughly implemented by the provider. This includes permanently deleting all information about the end-device from the Application Server after an end-device has exited.

2.5.2 Security threats

According to [10] LoRaWAN is more susceptible to physical attacks than higher layer attacks as illustrated in the following figure.

Name of threats		Likelihood			Impact on						Risk at			
		Technical difficulty	Motivation level	Likelihood	Scale	Detectability	Confidentiality	Integrity	Availability	Authentication & access control	Confidentiality	Integrity	Availability	Authentication & access control
	False join packets	None	Low	Unlikely	**LoRaWAN**	Medium	Minimal	Minimal	Significant	Minimal	Minor	Minor	Minor	Minor
MITM Attack	Bit-flipping / message forgery	None	Low	Unlikely	End-device	Low	Minimal	Moderate	Minimal	Minimal	Minor	Minor	Minor	Minor
	Frame payload attack	None	Low	Unlikely	End-device	Low	Minimal	Moderate	Minimal	Minimal	Minor	Minor	Minor	Minor
	Network flooding attack	None	Low	Unlikely	**LoRaWAN**	**High**	Minimal	Minimal	**Significant**	Minimal	Minor	Minor	Minor	Minor
	Network traffic analysis	None	Low	Unlikely	**LoRaWAN-EN**	Low	Moderate	Minimal	Minimal	Minimal	Minor	Minor	Minor	Minor
Physical Attacks	Destroy, remove or steal end-device	None	Medium	**Likely**	End-device	Medium	None	None	Moderate	None	None	None	**Major**	None
	Device cloning / Firmware replacement	None	High	**Likely**	End-device	Low	Moderate	Moderate	Minimal	**Significant**	**Major**	**Major**	Minor	**Critical**
	Security parameter extraction by phy. access	Solvable	Medium	Possible	End-device	Low	Moderate	Minimal	Minimal	**Significant**	**Major**	Minor	Minor	**Major**
	Plaintext key capture	None	Medium	**Likely**	End-device	Low	**Significant**	Moderate	Minimal	**Significant**	**Major**	**Major**	Minor	**Major**
	RF jamming	None	Low	Unlikely	**LoRaWAN**	Low	Minimal	Minimal	**Significant**	Minimal	Minor	Minor	Minor	Minor
	Self-Replay attack	Solvable	High	**Likely**	End-device	Low	Minimal	Minimal	**Significant**	Minimal	Minor	Minor	**Critical**	Minor
	Rogue end-device	Strong	High	**Likely**	End-device	Low	Moderate	Moderate	Moderate	**Significant**	**Major**	**Major**	**Major**	**Critical**
Rogue Gateway Attack	Beacon synchronization DoS attack	Solvable	Low	Unlikely	**LoRaWAN**	Medium	Minimal	Moderate	**Significant**	Minimal	Minor	Minor	**Major**	Minor
	Impersonation attack	Solvable	Low	Unlikely	**LoRaWAN**	Low	Minimal	Minimal	**Significant**	Minimal	Minor	Minor	**Major**	Minor
Routing Attacks	Selective forwarding attack	None	Low	Unlikely	**LoRaWAN**	Low	Minimal	Minimal	**Significant**	Minimal	Minor	Minor	Minor	Minor
	Sinkhole / Blackhole attack	None	Low	Unlikely	**LoRaWAN**	Low	Minimal	Minimal	Significant	Minimal	Minor	Minor	Minor	Minor

Figure 11. Security evaluation of LoRaWAN v1.1 [10, p. 17].

In the diagram above several different security threats are evaluated. Even though many of the known threats were eliminated in LoRaWAN 1.1 there are still many scenarios left where a part of the network is compromised.

Man-in-the-middle (MITM) attacks include bit-flipping and frame payload attacks. Bit-flipping is a scenario where an adversary changes the content of a message between servers, namely Application Server and Network Server. Frame payload attacks are more likely to be used in a network that supports roaming, because the FRMPayloads are transported between Network Servers which consider each other safe by default. [10, p. 14.]

Physical attacks include not only destroying or stealing a device but also cloning the firmware or replacing it altogether. To prevent an adversary from tampering with the firmware, a secure element can be implemented into the end-device, making it very difficult if not impossible to access the protected memory.

In this project, the servers are running on a virtual machine in Metropolia's local network, so they are very unlikely to become physically attacked. The gateway, however, is in a very open and easily accessible location in the public spaces of the Metropolia Myyrmäki Campus and could very easily be captured or otherwise harmed by someone with malicious intent. This flaw in security is however overlooked until the end of the testing since the open and centered location was purposely selected to optimize the reach of the gateway over the campus. Once the testing of the network is over, a more secure location will be chosen for the gateway.

2.6 Key distribution

LoRaWAN utilizes a wide variety of different authentication keys depending on the implementation. In the case of OTAA devices, there are always two root keys, namely AppKey and NwkKey, from which most of the other keys are derived. The root keys are stored in the end-device as well as in the Join-server that is responsible for enabling the connection of an end-device to the Network Server. The key scheme as whole is described by [15].

In LoRaWAN v1.0, the AppKey was the sole root key from which the session keys were derived. In addition to being the only root key, it was unchangeable. These issues posed several security threats that were pointed out by Jaehyu Kim and JooSeok Song in [16]. These threats were addressed in LoRaWAN v1.1 by introducing the dual-key scheme and an updated OTAA join-procedure, both of which were proposed by Kim and Song.

The following figure illustrates the key distribution scheme of LoRaWAN v1.1.

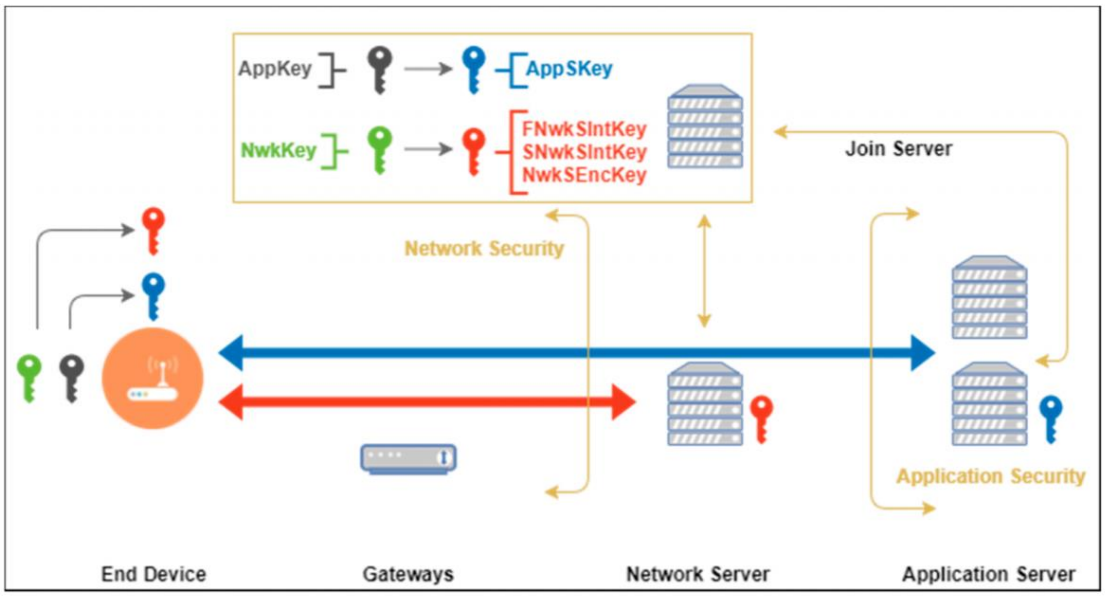


Figure 12. LoRaWAN 1.1 key distribution scheme [15, p. 3].

As seen in the previous figure, the session keys (FNwkSIntKey, SNwkSIntKey, NwkSEncKey, AppSKey) are derived from the root keys in the Join Server. These session keys are then used by the end-device and the corresponding server to ensure a secure connection.

According to [10, p. 9] there is currently no definition on the session durations of LoRaWAN, which means that it is up to the provider to define the time to live for each session. In LoRa Server the device session TTL is defined in the configuration of the Network Server component.

In order to enable remote multicasting in a secure way, LoRa Alliance introduced a set of new keys in [13] at the end of 2018. The generation of these keys follow a logic similar to the basic LoRaWAN 1.1 key distribution scheme shown in Figure 8. A new multicast root-key is derived from the existing AppKey and from that key, all the other multicast related keys are derived. This process is described in more detail in chapter 2.6.3.

2.7 LoRaWAN FUOTA Process

LoRaWAN supports Firmware Update Over-The-Air (FUOTA). The process runs at the application layer of the OSI model which means the whole process is version agnostic as the different versions of LoRaWAN have differences on the MAC sublayer of the OSI model. At the time of writing, the FUOTA update was fairly recently specified by LoRa Alliance. LoRa Server has already support for limited functionality of FUOTA. However, it is in experimental stage and only supports firmware updates to single devices, whereas the specification by LoRa Alliance allows updating multiple devices at once by multi-casting. Even though the FUOTA method is not being utilized in this project, the inner functionality of it will be quite thoroughly discussed in the following sections due to it being a functionality of high importance in development environment.

2.7.1 Update Process Summary

LoRa Alliance defines several new components for the FUOTA process, and the firmware update happens between these components:

- Firmware Update Server: The firmware update images are generated on this server and passed on to the Application Server. The images contain a header and a digital signature using a private/public key scheme.

- File Distribution Server: On the Application Server resides File Distribution Server (FDS), which is tasked to deliver the firmware update images to the end-device. If a multicast group is used, the FDS gathers the identifiers of all the end-devices that will be updated. The FDS sets up the fragmentation session for the end-devices to be updated and sends the fragmented file to the Network Server. The Network Server will then broadcast or unicast the fragmented file to the end-devices as specified in [11, p. 8].

- File Distribution Client: The counterpart of the FDS resides on the end-device. The end-device reconstructs the firmware image once it has received enough fragments.

- Firmware Update Agent: Once the File Distribution Client on the end-device has reconstructed the image, the Firmware Update Agent verifies the digital signature signed by the Firmware Update Server and confirms that the received image is compatible with the end-device. Once confirmed and verified, the end-device marks the image “ready”, which means that the bootloader will know to install it at the next reboot. Once the end-device reboots, the bootloader will decompress and install the new firmware and reboot again. It rejoins the network if in OTAA mode. Once completed, the end-device may optionally send its firmware version to the Firmware update server which can then collect the firmware versions of all the end-devices and mark them as updated. As pointed out in [11, p. 8], this last feature is yet to be defined by the LoRa Alliance.

The following figure illustrates how the different FUOTA components interact with each other:

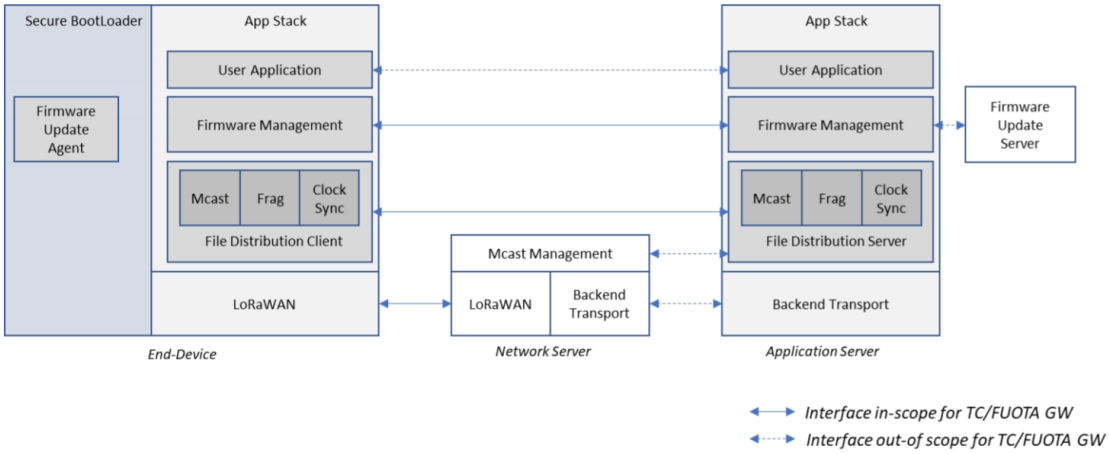


Figure 13. FUOTA architecture [11, p.7].

As seen in the figure, the File Distribution Server and the File Distribution Client comprise three subfunctions which will be further discussed in the following sections.

2.7.2 LoRaWAN Clock Synchronization

LoRa Alliance has introduced a package for end-device clock synchronization. It is useful for devices that do not have access to an accurate time source. If an end-device is running on LoRaWAN 1.1, it should use the DeviceTimeReq MAC command instead to request the correct time from the server. [12, p.7.]

A synchronized clock is essential to FUOTA process as it enables the synchronized transaction of fragmented packets between server and end-device. It is also needed for multicast group synchronization. This is discussed further in the next section.

The clock synchronization message package consists of a set of unicast uplink/downlink messages that the end-device and the server us to synchronize the clock:

CID	Command name	Transmitted by		Short Description
		End-device	server	
0x00	***PackageVersionReq***		x	Used by the AS to request the package version implemented by the end-device
0x00	***PackageVersionAns***	x		Conveys the answer to PackageVersionReq
0x01	***AppTimeReq***	x		Used by end-device to request clock correction
0x01	***AppTimeAns***		x	Conveys the clock timing correction
0x02	***DeviceAppTimePeriodicityReq***		x	Used by the application server for 2 purposes: Set the periodicity at which the end-device shall transmit AppTimeReq messages and request an immediate transmission of end-device time
0x02	***DeviceAppTimePeriodicityAns***	x		
0x03	***ForceDeviceResyncReq***		x	Used by the application server to the end-device to trigger a clock resynchronization.

Figure 14. Clock Synchronization Message package messages [12, p. 6].

The figure above describes each of the commands used to synchronize the clock of the end-device with the clock of the server. The parameters for each of the commands are out of the scope of this thesis.

2.7.3 Remote Multicast

In [13] Lora Alliance specifies an application layer messaging package which is designed to perform a set of operations on multiple end-devices synchronously. This method can be used in a wide set of applications in addition to the multicast FUOTA process.

The defined operations consist of programming a multicast distribution window into the end-devices, having the end-devices in a group switch to Class B or Class C temporarily in order to allow for higher transmission rates, and finally closing the multicast distribution window. These operations can be used on a set of end-devices once they have their clocks synchronized with the server.

A new encryption key derivation scheme is introduced for the downlink multicast message encryption. In this scheme all downlink data is encrypted using a multicast group McAppSKey, which is derived from a multicast group key called McKey. The McKey is decrypted from a McKey_encrypted key sent by the server. The decryption is done using the following formula: *McKey = aes128_encrypt(McKEKey, McKey_encrypted),* where the McKEKey is a lifetime end-device specific key derived from either from the end-device root key or the AppKey, depending on the LoRaWAN version of the end-device. The McKey and the McAppSKey are the same for all the end-devices of the same group. [13, p. 9-10.]

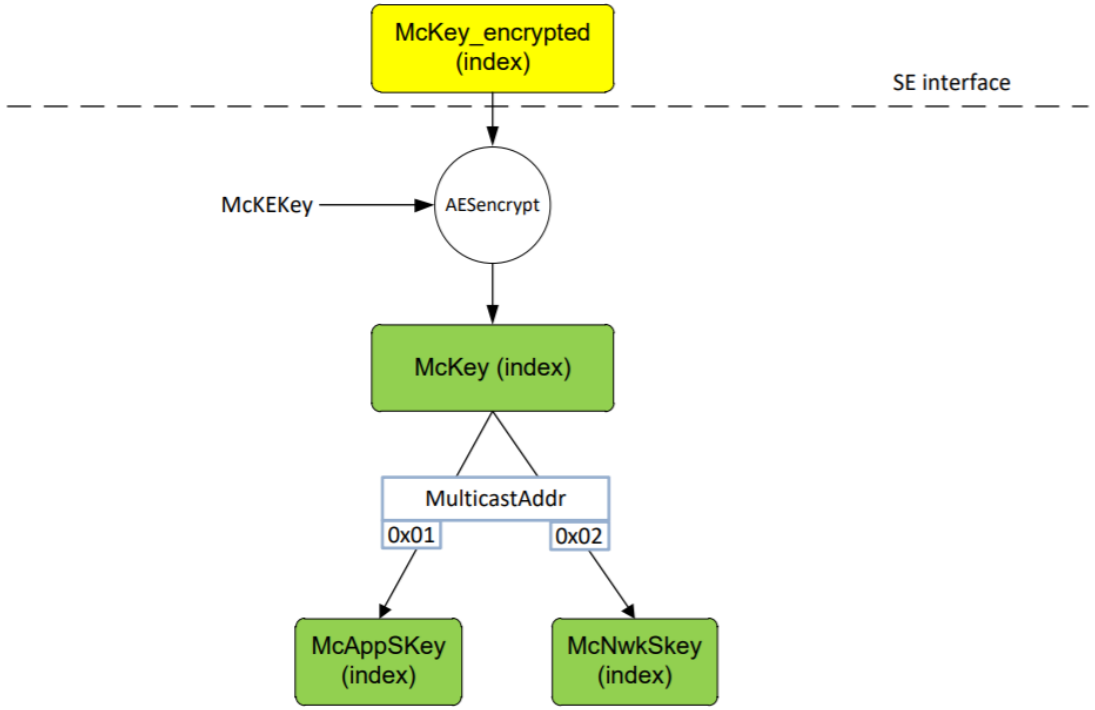


Figure 15. Multicast key derivation scheme [13, p.10].

The figure above illustrates the multicast key derivation scheme. As seen in the figure, in addition to the McAppSKey, a McNwkSKey is also derived from the McKey. The McNwkSKey is used to encrypt the messages between the end-device and the Network Server.

2.7.4 Fragmented Data Block Transport

As specified by the LoRa Alliance, the fragmented data block transportation package can be used not just in the FUOTA process, but in all data transfer where a large (from 1kB up) amount of data is transferred to an end-device. It is a process where, as threcee name suggests, a large amount of data is separated (fragmented) into data blocks and sent separately to the end-device. Data transferred this way is less likely to become corrupted due to connection errors and such.

To ensure that the data received by the end-device is valid, the payloads are encrypted and authenticated using the McAppSkey and McNwkSKey. However, this method alone

does not guarantee data validity, because these keys are the same on all the end-devices in the same group, and if one of the devices is compromised, the keys can no longer be considered secure.

To address the shortcomings of the key authentication scheme, LoRa Alliance provides guidelines to an additional step of authentication based on public/private cryptography certificate in [14, p.13].

3 LoRa Server

LoRa Server is an open-source LoRaWAN network server project. It was created by Orne Brocaar, a Dutch freelance software engineer.

Lora Server introduces a secure and an easy-to-use method of implementing a private or public LoRaWAN network. It abstracts each of the core components of the LoRaWAN as specified by the LoRa Alliance and adds to them features such as authentication over MQTT and PostgreSQL databases.

The reason LoRa Server was used in our implementation was precisely because of its clear, open-source interface along with its modifiability to our needs. It also utilizes well-known technologies, making the learning curve for using it a bit less steep.

The architecture of LoRa Server can be seen in the following figure.

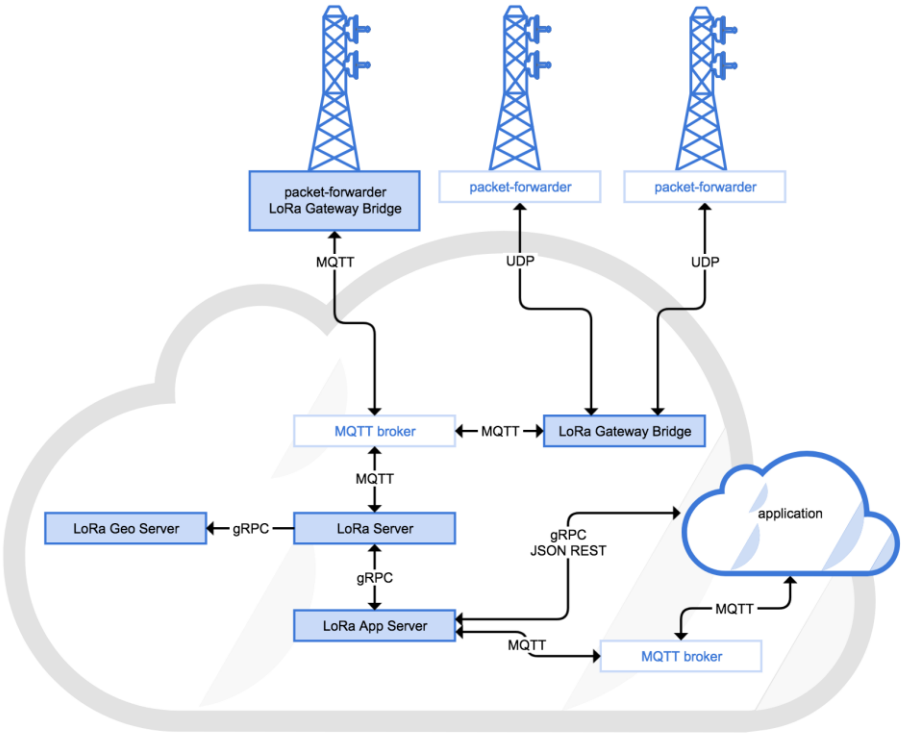


Figure 16. LoRa Server architecture [8]

As seen in Figure 6, LoRa Server uses MQTT and gRPC as the primary way of communicating between different components.

In the heart of the LoRa Server project there is the Network Server -component of the LoRaWAN, confusingly also called LoRa Server. This LoRa Server is responsible for implementing all the functions that the Network Server of LoRaWAN has. Inter-component communication between the server components is communicated over gRPC. [8.]

Another important component of the LoRa Server is the LoRa Gateway Bridge. It abstracts the packet-forwarder UDP protocol used by minimal firmware gateways into JSON packets over MQTT protocol. This LoRa Gateway Bridge can be implemented either in the backend or in the gateway itself. Both implementations are illustrated in Figure 6. Configuring it on the gateway adds a layer of security over the communication, by using MQTT with TLS instead of UDP in gateway-server connections.

LoRa Server project introduces LoRa Gateway OS as a way for quickly setting a gateway up and running as well as bundling the packet forwarder and LoRa Gateway Bridge in the same packet. Our project uses the LoRa Gateway OS for Wifx LORIX One gateway.

The Application Server in LoRa Server is abstracted to the LoRa App Server which in addition to providing the required functions of an Application Server, also provides a web user interface as well as a RESTful and gRPC API. It can handle end-device data over different protocols such as MQTT or HTTP and the data can be written to InfluxDB database directly. Our implementation uses MQTT over TLS.

LoRa App Server uses JSON web-tokens for authentication and authorization on its gRPC and JSON REST interfaces, while the communication over MQTT uses its own way of authenticating.

LoRa Server also has a component for geolocation services called LoRa Geo Server. It is used to resolve end-device locations based on time difference of arrival (TDoA) metadata provided by the gateways [9]. This component is not included in our implementation of the network but could be added to it at later development.

4 Project specifications and Methods

4.1 Meetings and Discussions

The initial project meeting was held at Metropolia with the instructor of the thesis. In this meeting the framework for the project was discussed. A decision was made that that the project would start by studying the background of the subject. It was also discussed that the project would possibly be carried out in cooperation with another student from Switzerland.

According to the project plan the hardware part, including setting up and configuring the gateways and doing the testing of the network with self-configured devices, would be carried out by the author. The network related area, including the backend and the authentication services, would be implemented by the collaborator from Switzerland.

The second meeting with the instructor was held at Metropolia on 11 February, 2019. The implementation was further discussed in detail. It was decided that the author would investigate the implementation of the whole network utilizing the TTN open-source libraries in more detail. The goal was to find out if the source code offered by TTN would be suitable for this project.

During the second meeting, it was also discussed how to test the network once it had been set up. The instructor of the thesis handed out an expansion board and a discovery kit to the author which could be used to test the network. It was discussed that the expansion board, USI STM32 Nucleo expansion board for LoRa, would be researched first. The goal was to see how wide usage the AT-command interface of the board provided, and if the board was relatively easy to connect to an evaluation board to be later easily usable by students.

After the second meeting, the customer service of The Things Industries (TTI) was contacted and asked whether the TTN could be used for the needs of this project. The response from TTI did not mention if there would be a way of implementing the LoRaWAN using TTN free of charge. The TTI customer service was contacted again with a follow-up question regarding this issue, but no reply was received.

Later, it was discovered that TTN also had their own separate email address for Q&A. Thus, the same questions regarding the implementation were asked again, this time from the email provided by TTN.

The CEO of TTI replied to the email sent and confirmed that the TTN could be used freely, now and in the future, to implement a private network. He also confirmed that the Identity server, a feature of TTN which enables user authentication, could be used in a private network. Lastly, he confirmed that the whole network could be hosted and managed privately independent of the global network provided by TTN.

A third meeting with the instructor was held on 13 May, 2019 at Metropolia Myyrmäki Campus. During this meeting the porting of the LoRaWAN to the Metropolia premises was discussed. A decision was made that the LoRaWAN would be installed in Myyrmäki Campus during the summer. Another topic discussed with the instructor was the implementation of the user authentication of the server. The authentication could be done utilizing LDAP (Lightweight Directory Access Protocol) and that Ilkka Kylmäniemi, a lecturer at Metropolia, would possibly be able to give further information regarding this matter.

Kylmäniemi was approached via email on 15 May, 2019 regarding the user authentication. He confirmed that the API to use is indeed LDAP and that Metropolia's IT Services could tell more about it.

A series of emails was exchanged with Metropolia Helpdesk starting from 10 May, 2019. The topic of discussion was the implementation of the Metropolia's user authentication. Initially the Helpdesk was asked about the usage of user credentials in MQTT authentication. The conversation was continued over the phone on 5 June, 2019 with Juhana Lähteenmäki from Helpdesk. All the matters to consider when moving the LoRaWAN to Metropolia premises were discussed over the phone. Lähteenmäki provided detailed information on who to contact on which matter.

After the discussion with Lähteenmäki, the webmaster of Metropolia, Pekka Perälampi was approached with an email requesting for general guidance on the migration of the server to Metropolia's local network. Perälampi forwarded the request over to Metropolia's IT Services, but no response was ever received from their end.

From June to July of 2019, the thesis work was carried out at Metropolia's Myyrmäki Campus. During this time the Helpdesk operating on campus was aiding with the integration of the LoRaWAN to the Metropolia's network. Janne Teräslahti from IT support helped finding a suitable place for the LoRaWAN gateway and gave general advice about the network infrastructure of Metropolia and details on who to contact on which matter if needed.

4.2 Researching and Planning

The planning of the project was started after the first meeting. The first step was to find out what the currently existing implementations for a LoRaWAN are, and which method would best suit the needs of this project, if any.

The first option researched that could have been possibly used for the implementation was The Things Network (TTN). TTN refers to the public LoRaWAN network currently funded by The Things Industries.

The reason that this was researched first was that the TTN has a lot of their data open source and if they had offered a way to create a custom-made backend that could have been hosted by Metropolia, but still be connected to the vast TTN, it would have been an ideal way of creating the Metropolia LoRaWAN.

Another seemingly viable existing way to implement the LoRaWAN was the LoRa Server project, which is another open source set of applications with which one can implement their own private LoRaWAN. The positive features of LoRa Server were its easy configurability and the fact that it uses MQTT, a lightweight messaging protocol, as its primary communication protocol.

After the second meeting with the instructor, the extent of the usage possibilities of the Things Network were researched. After some research it was concluded that the TTN V3, which was released at the beginning of 2019, could possibly be suitable for the implementation.

While waiting for the response from TTN to the mail querying the usage limitations of the TTN, the alternative option, LoRa Server, for the implementation of the LoRaWAN was researched further.

Based on the information provided on the website of LoRa Server, it also seemed like a very promising solution for the implementation. The LoRa Server provided the necessary level of security over the connections between the gateways and the backend. It also provided the possibility of setting up a custom user database, which could potentially be used to map the Metropolia accounts to the backend.

Ultimately, after some research on both implementation options, the LoRa Server was chosen as the platform and the development of the LoRaWAN was initiated.

5 Development and Implementation

Initially the network was intended to be implemented utilizing The Things Network, and the open source libraries related to it. However, due to the mechanics of it not being completely transparent and the implementation being unfamiliar and abstract, it was concluded that a more suitable approach for the implementation would be utilizing an open-source project, LoRa Server. Most of the configuration was carried out by following the guidelines provided by LoRa Server documentation.

5.1 Gateway Configuration

Once the approach was chosen, the implementation was initiated by configuring a gateway for the network. LORIX One, a low cost LoRa IP43/IP46 gateway by Wifx, was used for this purpose. LoRa Server provided a bootable LoRa Gateway OS which enabled a relatively fast configuration of the gateway.

The LoRa Gateway OS was flashed on an SD card and the LORIX One gateway module was then booted up using this SD card. Once the gateway was up and running, it was configured over an SSH connection. In order to make the gateway forward the messages that it received to the Network Server; necessary parameters had to configured.

The LoRa Gateway OS includes two fundamental message relay functions, the first being the packet forwarder UDP protocol. This is the very basic protocol which handles the forwarding of data from an end-device to a server through an UDP/IP link. Due to the unreliability of UDP, LoRa Server adds a level of abstraction on the packets before sending them to a server. This level of abstraction is called the LoRa Gateway Bridge. It abstracts all outgoing packets into JSON over MQTT, which makes them easy to monitor and secure.

The packet-forwarder was thus configured to forward all packets locally within the gateway module to the LoRa Gateway Bridge, which was then configured to send the abstract JSON packets to the correct Network Server over TCP/IP protocol. In the first stage of development, little to no security measures were taken.

5.2 Server-side Configuration

Initially the backend was deployed on Cloud IoT Core, an IoT service by Google. However, the configuration of the connection between the gateway module and the Cloud IoT Core took unnecessarily much time and work, and eventually resulted in a failure. Thus, a more familiar approach for the backend was chosen. The Network Server along with the Application Server was deployed on an Ubuntu VM running on Google Cloud Platform.

A PostgreSQL database was set up for the Network Server to persist gateway information. Then the actual LoRa Server was installed, using a Debian package. The Network Server was configured to have the correct network parameters.

After the Network Server had been verified to be functional, the Application Server, LoRa App Server, was configured following the same structure as the Network Server configuration.

5.3 Testing the Network

Along the development and initial set up of the network, there were steps of testing each individual component of the network. Finally, when everything seemed to function as it was intended, the functionality of the network was tested by connecting an end-device to the gateway and sending test data in.

B-L072NZ-LRWAN1, a LoRa Discovery kit by ST was used. This device was chosen due to its seemingly quick prototyping and the author being already familiar with STM32 microcontrollers. The test device was set up utilizing the example libraries provided by STMicroelectronics.

After getting all the device and network configurations correct, the end-device sent data to the Application Server successfully. The following figure shows the uplink message sent by the end-device, displayed on the web user interface of the LoRa App Server.

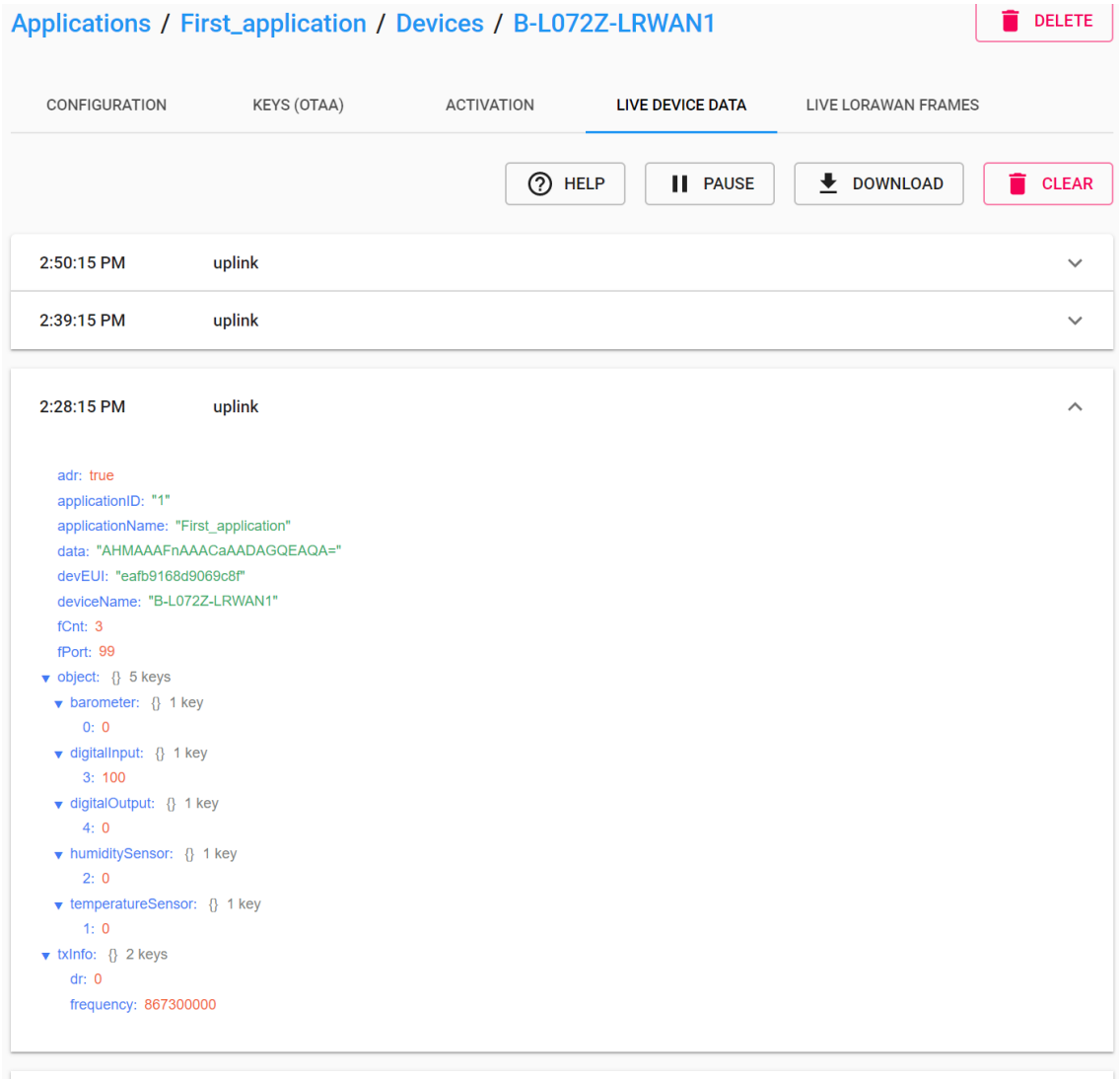


Figure 17. An uplink message sent by an end-node to the Application Server.

5.4 Network Integration

Before starting the integration of the LoRaWAN into the Metropolia network some security configurations had to be carried out. The MQTT communication between the LoRa Server and the gateway as well as the internal MQTT communication of the LoRa Server was configured to transfer data over a TLS secured port 8883 instead of the MQTT default port 1883.

The first step in the integration was to install the gateway to Metropolia's Myyrmäki Campus. The following figure shows how the gateway was installed.

Figure 18. Gateway installed at Metropolia's Myyrmäki Campus

The location of the gateway was discussed with the instructor of the thesis, a space which is one floor above the cafeteria of the Myyrmäki Campus was chosen because it is at the center of the whole campus and the space is wide and open, which will minimize the signal loss caused by obstacles.

The server was configured on a virtual machine running on a Red Hat CFME (Cloud-Forms Management Engine) managed by Metropolia. The platform only supported the creation of Windows Servers or CentOS; thus, CentOS was chosen as the operating system for the server.

The migration of the backend from a Debian-based architecture to a Red Hat Enterprise Linux (RHEL) based one was not without some complications. Whereas LoRa Server's website provided a lot of detail on the setup of the backend on Debian distributions, there was very little detail about how to do the same on RHEL distributions. In addition to the little amount of information, RHEL was also unfamiliar to the author, making the transition quite slow and complicated.

Because there were no ready-made installation packages for the images of the backend for RHEL-based distributions, the backend was first set up using Docker-Compose, which is a tool that enables running of multi-container Docker applications. The backend was successfully set up and running using Docker-compose and the process was quite trivial. However, trying to configure the MQTT broker on the Docker-compose configuration files ended up being not only non-trivial, but too complicated.

Because the Docker-compose was not familiar to the author and the project did not move forward, another approach was taken. The backend was installed component by component by downloading the source files as RPM packages and configuring them one by one. This approach ended up being fruitful and the backend was once again set up successfully and this time the MQTT broker was also configured successfully to use port 8883 and use ACLs and usernames and passwords to authenticate users.

After the backend was set up and running, the gateway was configured to connect to the IP address of the backend now running on Metropolia's Educloud. However, the gateway failed to connect to the backend after all the configurations. This issue was thoroughly investigated using different packet monitoring methods to no end. Metropolia's IT Services were also consulted regarding this issue, but according to their systems, there should nott have been anything blocking the communication.

6 Conclusion

The thesis aimed to implement a private LoRaWAN network on Metropolia's premises. The network would consist of the server-side backend functionalities as well as the physical gateway converting and relaying RF messages to the backend. In addition to the basic operation, an authentication method was intended to be implemented to the network, which would only allow Metropolia students to utilize the network.

The initial setup and testing of the LoRaWAN outside Metropolia's premises were successful. In this phase, only the basic operation was tested. The gateway seemed to successfully forward packets to the backend from which they are correctly displayed to the front-end.

However, when migrating the server to Metropolia's local network there seemed to be some complications. The development was hindered by the limitation of only being able to use a RHEL-based distribution, CentOS 7, for the server whereas the initial setup was carried out using a Debian-based distribution. The server was successfully set up. However, the connection between the server and the gateway was never established on Myyrmäki Campus, due to an unknown blockage between the two.

The secondary goal of implementing the user authentication was set up so that the MQTT broker would allow only specific topics to be published/subscribed to by only the users found from the server's PostgreSQL database.

Instead of implementing a login system that could be used with Metropolia login credentials or creating a user on the server for each student, the system was set up so that only a set of user accounts would be generated by a script located on the backend. Each of these user accounts would be shared by different groups of students defined by the lecturer. This approach was an attempt to minimize the amount of manual configuration needed, when new users start using the network.

References

1 N. Sornin. 2017. LoRaWAN 1.1 Specification. <https://lora-alliance.org/sites/default/files/2018-04/lorawantm_specification_-v1.1.pdf>. 11.10.2017.

2 A. Yegin. 2017. LoRaWAN Backend Interfaces 1.0 Specification. <https://lora-alliance.org/sites/default/files/2018-04/lorawantm-backend-interfaces-v1.0.pdf>. 11.10.2017.

3 The Things Network. 2019. LoRaWAN Overview. Web document. <https://www.thethingsnetwork.org/docs/lorawan/>. Read 2.5.2019.

4 The Things Network. 2019. Gateways. Web document. <https://www.thethingsnetwork.org/docs/gateways/>. Read 2.5.2019.

5 Vit Prajzler. 2015. LoRa, LoRaWAN and LORIOT.io. Web document. <https://www.loriot.io/lorawan.html>. 1.8.2015. Read 2.5.2019.

6 Ilson You, Soonhyun Kwon, Gaurav Choudhary, Vishal Sharma, Jung Taek Seo. 2018. An Enhanced LoRaWAN Security Protocol for Privacy Preservation in IoT with a Case Study on a Smart Factory-Enabled Parking System. <https://www.mdpi.com/1424-8220/18/6/1888/htm>. 8.6.2018.

7 Techplayon. 2018. LoRa- (Long Range) Network and Protocol Architecture with Its Frame Structure. Web document. <http://www.techplayon.com/lora-long-range-network-architecture-protocol-architecture-and-frame-formats/>. 10.8.2018. Read 2.5.2019.

8 LoRa Server. 2019. System architecture. Web document. <https://www.loraserver.io/overview/architecture/>. Read 3.5.2019.

9 LoRa Server. 2019. LoRa Geo Server. Web document. <https://www.loraserver.io/lora-geo-server/overview/>. Read 3.5.2019.

10 Ismail Butun, Nuno Pereira, Mikael Gidlund. 2018. Security Risk Analysis of LoRaWAN and Future Directions. <https://www.researchgate.net/publication/329858421_Security_Risk_Analysis_of_LoRaWAN_and_Future_Directions>.

11 J. Catalano. 2019. FUOTA Process Summary Technical Recommendation. <https://lora-alliance.org/sites/default/files/2019-04/tr002-fuota_process_summary-v1.0.0.pdf>. 30.1.2019.

12 J. Catalano. 2019. LoRaWAN Application Layer Clock Synchronization Specification. <https://lora-alliance.org/sites/default/files/2018-09/application_layer_clock_synchronization_v1.0.0.pdf>. 10.9.2018.

13 J. Catalano. 2019. LoRaWAN Remote Multicast Setup Specification. <https://lora-alliance.org/sites/default/files/2018-09/remote_multicast_setup_v1.0.0.pdf>. 10.9.2018.

14 J. Catalano. 2019. LoRaWAN Fragmented Data Block Transport Specification. <https://lora-alliance.org/sites/default/files/2018-09/fragmented_data_block_transport_v1.0.0.pdf>. 10.9.2018.

15 Jialuo Han, Jidong Wang. 2018. An Enhanced Key Management Scheme for LoRaWAN. <https://www.researchgate.net/publication/328741904_An_Enhanced_Key_Management_Scheme_for_LoRaWAN>. 21.8.2018.

16 Jaehyu Kim, JooSeok Song. A Dual Key-Based Activation Scheme for Secure LoRaWAN. 2017. <https://www.researchgate.net/publication/320902435_A_dual_key-based_activation_scheme_for_secure_LoRaWAN>. 28.4.2017.

17 LoRa Alliance. 2017. LoRaWAN SECURITY. White paper. <https://lora-alliance.org/sites/default/files/2019-05/lorawan_security_whitepaper.pdf>. Read 21.9.2019.

18 Margaret Rouse. 2017. Advanced Encryption Standard (AES). Web document. <https://searchsecurity.techtarget.com/definition/Advanced-Encryption-Standard>. Read 21.9.2019.

19 Joan Daemen, Vincent Rijmen. 2002. The Design of Rijndael. New York: Springer-Verlag Berlin Heidelberg.

20 Dobre Blazhevski et al. 2013. Modes of Operation of the AES Algorithm. Conference publication. <https://pdfs.semanticscholar.org/8822/66e916ec18ea7022bfa149954a29593f7490.pdf>. Read 21.39.2019.

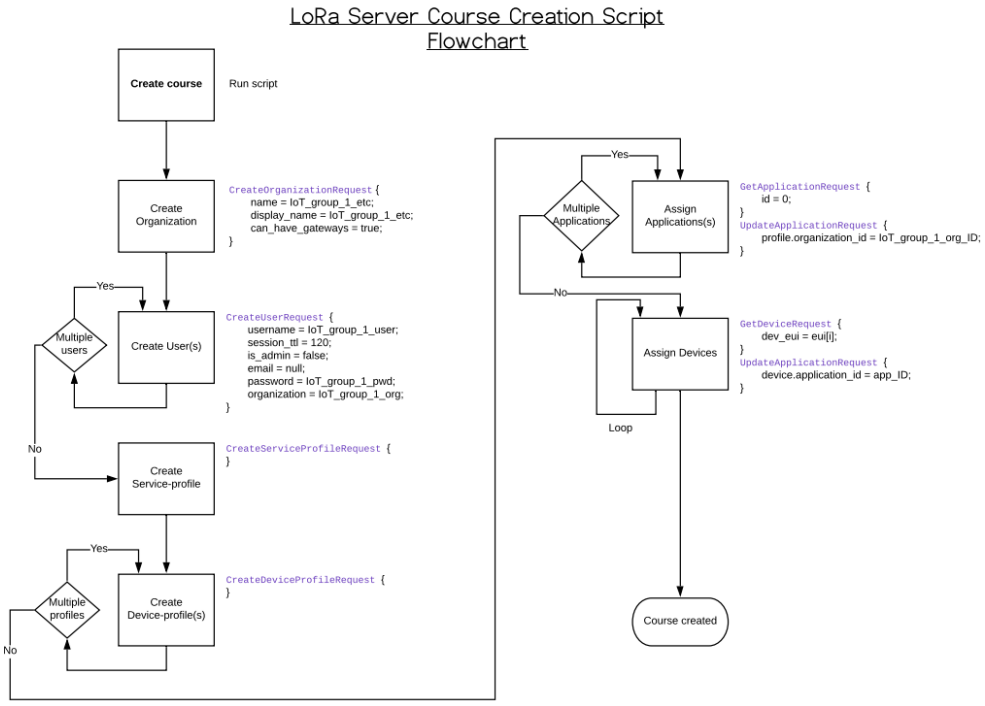
21 Helger Lipmaa et al. 2000. Comments to NIST concerning AES Modes of Operations: CTR-Mode Encryption. Web document. <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.79.1353&rep=rep1&type=pdf>. Read 22.9.2019.

22 Wikipedia. 2019. CBC-MAC. Web document. <https://en.wikipedia.org/wiki/CBC-MAC>. Read 22.9.2019.

23 Tetsu Iwata, Kaoru Kurosawa. 2003. OMAC: One-Key CBC MAC. Conference paper. <https://eprint.iacr.org/2002/180.pdf>. Read 22.9.2019.

24 JH. Song et al. 2006. The AES-CMAC Algorithm. Web document. <https://tools.ietf.org/pdf/rfc4493.pdf>. Read 22.9.2019.

LoRa Server course creation script flowchart



LoRa Server course creation scripts

```
#!/bin/bash

# Author:           Abel Onditi
# Date:             21.7.2019
# Last modified:    30.7.2019
# Description:      Create all the necessary components of LoRa App Server for an IoT Course

# Define a timestamp function
timestamp() {
    date +"[%Y-%m-%d %H:%M:%S]"
}

# Create course
    source ./createOrganization.sh \
&&  source ./createUsers.sh ${organization_id_generated} 4 \
&&  source ./createServiceProfile.sh ${organization_id_generated} \
&&  source ./createDeviceProfiles.sh ${organization_id_generated} \
&&  source ./createApplications.sh ${organization_id_generated} ${serviceProfileID} \
&&  source ./createDevices.sh ${appID} ${deviceProfileID} 10
```

```
#!/bin/bash

# Author:           Abel Onditi
# Date:             21.7.2019
# Last modified:    26.7.2019
# Description:      Create a new organization to LoRa App Server

    organization_name="IoTGroup" \
&&  organization_id=0 \
&&  response=$(curl -X POST \
        --header 'Content-Type: application/json' \
        --header 'Accept: application/json' \
        --header "Grpc-Metadata-Authorization: Bearer $( cat jwt_key )" \
        -d '{
        "organization": {
            "canHaveGateways": true,
            "displayName": "'"${organization_name}"'"${organization_id}"'",
            "id": "'"${organization_id}"'",
            "name": "'"${organization_name}"'"${organization_id}"'"
        }
        }' 'http://10.114.32.4:8080/api/organizations' \
        2>/dev/null) \
&&  organization_id_generated=$( jq -r .id <<< ${response} ) \
&&  printf "$(timestamp) Organization <${organization_name}${organization_id}> created. ID [${organization_id_generated}].\n\r" | tee -a report.txt
```

```
#!/bin/bash

# Author:           Abel Onditi
# Date:             21.7.2019
# Last modified:    26.7.2019
# Description:      Create a new user to LoRa App Server
# Parameters:       {1}: Organization ID
#                   {2}: Number of users

    if [ -z "${2}" ]; then
        users_n=1
    else
        users_n=${2}
    fi \
&&  for (( i=0; i<${users_n}; i++ ))
    do

        user_id=$i \
        && username="IoTGroupUser${user_id}" \
        && password=$( hexdump -n 8 -e '"%08X"' /dev/urandom  ) \
        && response=$(curl -X POST \
            --header 'Content-Type: application/json' \
            --header 'Accept: application/json' \
            --header "Grpc-Metadata-Authorization: Bearer $( cat jwt_key )" \
            -d '{
            "organizations": [
                {
                "isAdmin": false,
                "organizationID": "'"${1}"'"
                }
            ],
            "password": "'"${password}"'",
            "user": {
                "email": "'"${username}"'@metropolia.fi",
                "id": "'"${user_id}"'",
                "isActive": true,
                "isAdmin": false,
                "note": "'"${username}"'",
                "sessionTTL": 0,
                "username": "'"${username}"'"
            }
            }' 'http://10.114.32.4:8080/api/users' \
            2>/dev/null) \
        &&  user_id_generated=$( jq -r .id <<< ${response} ) \
        &&  printf "$(timestamp) User <${username}> created. Password: [$password]. ID [${user_id_generated}].\n\r" | tee -a report.txt
    done
```

```
#!/bin/bash

# Author:           Abel Onditi
# Date:             26.7.2019
# Last modified:    30.7.2019
# Description:      Create a new service profile for the IoTGroup organizationr
# Parameters:       {1}: Organization ID

    serviceProfileName="IoTGroupSP0" \
&&  organization_id=${1} \
&&  response=$(curl -X POST \
        --header 'Content-Type: application/json' \
        --header 'Accept: application/json' \
        --header "Grpc-Metadata-Authorization: Bearer $( cat jwt_key )" \
        -d '{
        "serviceProfile": {
            "id": "1",
            "name": "'"${serviceProfileName}"'",
            "organizationID": "'"${organization_id}"'",
            "networkServerID": "1",
            "ulRate": 0,
            "ulBucketSize": 0,
            "ulRatePolicy": "DROP",
            "dlRate": 0,
            "dlBucketSize": 0,
            "dlRatePolicy": "DROP",
            "addGWMetaData": false,
            "devStatusReqFreq": 0,
            "reportDevStatusBattery": false,
            "reportDevStatusMargin": false,
            "drMin": 0,
            "drMax": 0,
            "channelMask": null,
            "prAllowed": false,
            "hrAllowed": false,
            "raAllowed": false,
            "nwkGeoLoc": false,
            "targetPER": 0,
            "minGWDiversity": 0
        }}' 'http://10.114.32.4:8080/api/service-profiles' \
        2>/dev/null) \
&&  serviceProfileID=$( jq -r .id <<< ${response} ) \
&&  printf "$(timestamp) ServiceProfile <${serviceProfileName}> created. Service Profile ID: [${serviceProfileID}].\n\r" | tee -a report.txt
```

```
#!/bin/bash

# Author:           Abel Onditi
# Date:             30.7.2019
# Last modified:    30.7.2019
# Description:      Create new device profiles for the IoTGroup organization
# Parameters:       {1}: Organization ID

    profileName="B-L072Z-LRWAN1" \
&&  organization_id=${1} \
&&  response=$(curl -X POST \
                --header 'Content-Type: application/json' \
                --header "Grpc-Metadata-Authorization: Bearer $( cat jwt_key )" \
                -d '{
                    "deviceProfile": {
                        "id": "1",
                        "name": "'"${profileName}"'",
                        "organizationID": "'"${organization_id}"'",
                        "networkServerID": "1",
                        "supportsClassB": false,
                        "classBTimeout": 0,
                        "pingSlotPeriod": 0,
                        "pingSlotDR": 0,
                        "pingSlotFreq": 0,
                        "supportsClassC": false,
                        "classCTimeout": 0,
                        "macVersion": "1.1.0",
                        "regParamsRevision": "A",
                        "rxDelay1": 0,
                        "rxDROffset1": 0,
                        "rxDataRate2": 0,
                        "rxFreq2": 0,
                        "factoryPresetFreqs": [],
                        "maxEIRP": 0,
                        "maxDutyCycle": 0,
                        "supportsJoin": true,
                        "rfRegion": "EU868",
                        "supports32BitFCnt": false,
                        "payloadCodec": "",
                        "payloadEncoderScript": "",
                        "payloadDecoderScript": ""
                    }}' 'http://10.114.32.4:8080/api/device-profiles' \
                2>/dev/null) \
&&  deviceProfileID=$( jq -r .id <<< ${response} ) \
&&  printf "$(timestamp) Device-profile <${profileName}> created. Device-profile ID: [${deviceProfileID}].\n\r" | tee -a report.txt
```

```
#!/bin/bash

# Author:           Abel Onditi
# Date:             30.7.2019
# Last modified:    30.7.2019
# Description:      Create a new application(s) for the IoTGroup organization
# Parameters:       {1}: Organization ID
#                   {2}: Service-profile ID

    appName="IoTGroupApp0" \
&&  organization_id=${1} \
&&  serviceProfile_id=${2} \
&&  response=$(curl -X POST \
        --header 'Content-Type: application/json' \
        --header 'Accept: application/json' \
        --header "Grpc-Metadata-Authorization: Bearer $( cat jwt_key )" \
        -d '{
            "application": {
                "description": "Default IoT Group Application",
                "id": "1",
                "name": "'"${appName}"'",
                "organizationID": "'"${organization_id}"'",
                "payloadCodec": "",
                "payloadDecoderScript": "",
                "payloadEncoderScript": "",
                "serviceProfileID": "'"${serviceProfile_id}"'"
            }
            }' 'http://10.114.32.4:8080/api/applications' \
        2>/dev/null) \
&&  appID=$( jq -r .id <<< ${response} ) \
&&  printf "$(timestamp) Application <${appName}> created. App ID [${appID}]. \n\r" | tee -a report.txt
```

```
#!/bin/bash

# Author:           Abel Onditi
# Date:             30.7.2019
# Last modified:    30.7.2019
# Description:      Create a new device(s) for the IoTGroup organization
# Parameters:       {1}: Application ID
#                   {2}: Device-profile ID
#                   {3}: Number of devices

for (( i=0; i<${3}; i++ ))
do
    devName="IoTGroupDevice${i}" \
&&  devEUI="$( hexdump -n 8 -e '"%08X"' /dev/urandom )" \
&&  app_id=${1} \
&&  deviceProfile_id=${2} \
&&  response=$(curl -X POST \
        --header 'Content-Type: application/json' \
        --header 'Accept: application/json' \
        --header "Grpc-Metadata-Authorization: Bearer $( cat jwt_key )" \
        -d '{
            "device": {
                "applicationID": "'"${app_id}"'",
                "description": "Default IoT Group device",
                "devEUI": "'"${devEUI}"'",
                "deviceProfileID": "'"${deviceProfile_id}"'",
                "name": "'"${devName}"'",
                "referenceAltitude": 0,
                "skipFCntCheck": true
            }
            }' 'http://10.114.32.4:8080/api/devices' \
        2>/dev/null) \
\
&&  appKey="$( hexdump -n 16 -e '"%08X"' /dev/urandom )" \
&&  nwkKey="$( hexdump -n 16 -e '"%08X"' /dev/urandom )" \
&&  response=$(curl -X POST \
        --header 'Content-Type: application/json' \
        --header 'Accept: application/json' \
        --header "Grpc-Metadata-Authorization: Bearer $( cat jwt_key )" \
        -d '{
            "deviceKeys": {
                "appKey": "'"${appKey}"'",
                "devEUI": "'"${devEUI}"'",
                "genAppKey": "'"${appKey}"'",
                "nwkKey": "'"${nwkKey}"'"
            }
            }' "http://10.114.32.4:8080/api/devices/${devEUI}/keys" \
        2>/dev/null) \
        &&  printf "$(timestamp) Device <${devName}> created. DevEUI/JoinEUI: [${devEUI}] ::: AppKey: [${appKey}] ::: NwkKey: [${nwkKey}].\n\r" | tee -a report.txt
done
```

LoRa Server course deletion scripts

```
#!/bin/bash

# Author:           Abel Onditi
# Date:             25.7.2019
# Last modified:    26.7.2019
# Description:      Delete all components created by the "createCourse.sh" -script

# Define a timestamp function
timestamp() {
    date +"[%Y-%m-%d %H:%M:%S]"
}

# Delete course
    source ./deleteOrganizations.sh \
&&  source ./deleteUsers.sh \
```

```
#!/bin/bash

# Author:         Abel Onditi
# Date:           21.7.2019
# Last modified:  26.7.2019
# Description:    Delete all but the default organization from the LoRa App Server

   response=$(curl -X GET \
                --header 'Accept: application/json' \
                --header "Grpc-Metadata-Authorization: Bearer $( cat jwt_key )" \
                'http://10.114.32.4:8080/api/organizations?limit=100&search=IoTGroup' \
                2>/dev/null) \
\
\
&& totalCount=$( jq -r '.totalCount' <<< "${response}" ) \
\
\
&& declare -a organizationIdArr=() \
&& for row in "$( jq -r '.result[]' <<< "${response}" )"
   do
      organizationIdArr+=( $(jq -r .id <<< "${row}") )
   done \
\
\
&& for id in "${organizationIdArr[@]}"
   do
      response=$(curl -X DELETE \
            --header 'Accept: application/json' --header "Grpc-Metadata-Authorization: Bearer $( cat jwt_key )" \
            "http://10.114.32.4:8080/api/organizations/$id" \
            2>/dev/null) \
      && printf "$(timestamp) Organization deleted. ID [${id}].\n\r" | tee -a report.txt
   done
\
\
```

```
#!/bin/bash

# Author:           Abel Onditi
# Date:             26.7.2019
# Last modified:    26.7.2019
# Description:      Delete all but the admin user from the LoRa App Server

   response=$(curl -X GET \
                  --header 'Accept: application/json' \
                  --header "Grpc-Metadata-Authorization: Bearer $( cat jwt_key )" \
                  'http://10.114.32.4:8080/api/users?limit=100&search=IoTGroup' \
                  2>/dev/null) \
\
\
&& totalCount=$( jq -r '.totalCount' <<< "${response}" ) \
\
\
&& declare -a userIdArr=() \
&& for row in "$( jq -r '.result[]' <<< "${response}" )"
   do
      userIdArr+=( $(jq -r .id <<< "${row}") )
   done \
\
\
&& for id in "${userIdArr[@]}"
   do
      response=$(curl -X DELETE \
            --header 'Accept: application/json' --header "Grpc-Metadata-Authorization: Bearer $( cat jwt_key )" \
            "http://10.114.32.4:8080/api/users/$id" \
            2>/dev/null) \
      && printf "$(timestamp) User deleted. ID [${id}].\n\r" | tee -a report.txt
   done
\
\
```