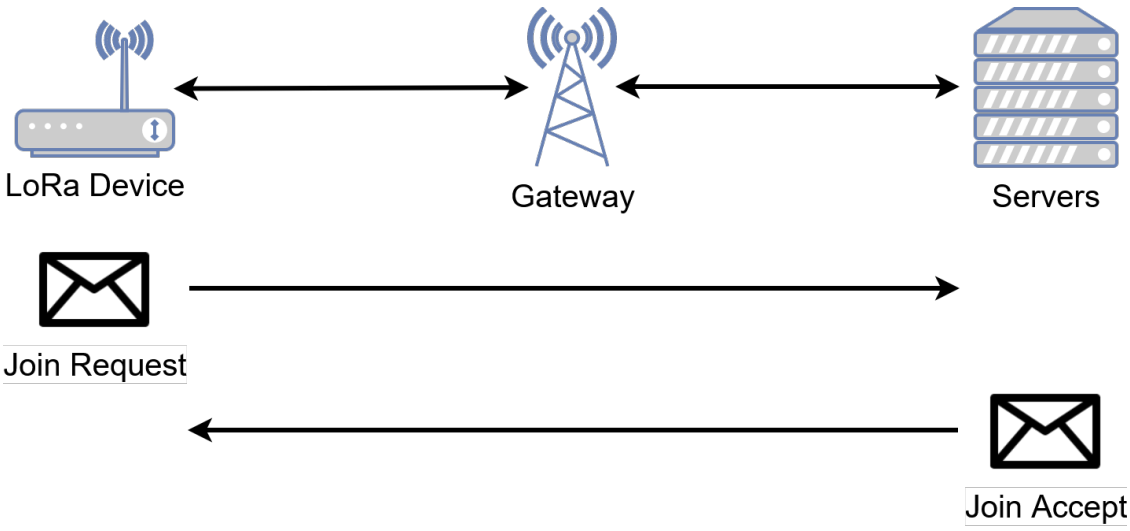

Activation of LoRaWAN end devices by using Public Key Cryptography

Master's thesis in Computer Science and Engineering

Fredrik Mårlind

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2020

Master's thesis 2020

Activation of LoRaWAN end devices by using Public Key Cryptography

Fredrik Mårlind

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2020

Activation of LoRaWAN end devices by using Public Key Cryptography
Fredrik Mårlind

Supervisor: Ismail Butun, Department of Computer Science and Engineering
Examiner: Magnus Almgren, Department of Computer Science and Engineering

Master's Thesis 2020
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2020

Activation of LoRaWAN end devices by using Public Key Cryptography
Fredrik Mårlind
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

The usage of Internet of Things (IoT) devices is growing every day, and Long Range Wide Area Network (LoRaWAN) is one specification that enables these devices to have access to the internet. The current specification uses a robust symmetric encryption scheme for secure communication. However, the root keys used for the encryption is static and delegated to the device before deployment. If the root keys are compromised, all past and future messages also become compromised. Since the root keys are static, physical access to the device is required to enforce a new key.

This thesis aims to design and evaluate a new process to assign root keys to the devices in LoRaWAN using Public Key Cryptography (PKC) and evaluate the feasibility of the proposed process. The new process allows a device to get the root keys dynamically and for them to be replaced at will. However, PKC requires longer keys to get equivalent cryptographic strength to a symmetric encryption scheme. This increased key size in turn requires more processing power to use, thus increasing battery consumption. The feasibility of the new process is evaluated based on the increase in power usage since IoT devices are usually battery powered.

The implemented process, called Public Key Over the Air Activation (PK-OTAA), is based on a PKC algorithm, specifically ECDH. PK-OTAA provides enhanced security at the cost of increasing battery consumption by 77% compared to the standard Over the Air Activation (OTAA) in LoRaWAN. Considering that PK-OTAA will only be executed a few times during the lifetime of the device, the increase in consumption is negligible. The report concludes that PK-OTAA is feasible to use when looking at power consumption. Although the security of the PK-OTAA procedure is yet to be proven, it can improve the regular OTAA procedure of LoRaWAN by allowing renewal of the root keys.

Keywords: LoRa, LoRaWAN, LPWAN, LPWA, IoT, PKC, PKI, Security, OTAA, ECDH

Acknowledgments

First and foremost, I want to thank my thesis advisor Ismail Butun. He was always available to answer any questions and provide valuable feedback on my work. I would also like to thank my examiner Magnus Almgren for his valuable input and pointers throughout the thesis.

I want to thank Lars Svensson for providing pointers with regards to electrical theory and for providing the measurement tool.

Finally, I must express my utmost gratitude to my friends and family for their support throughout my years of study and throughout the work in this thesis. I would not have accomplished this without all of you. Thank you.

Fredrik Mårlind, Gothenburg, August 2020

Contents

List of Figures

List of Tables

List of Abbreviations

ABP	Activation by Personalization
AES	Advanced Encryption Standard
AppKey	Application Key
AppSKey	Application Session Key
AS	Application Server
BDH	Bilinear Diffie-Hellman
CA	Certificate Authority
CIA	Confidentiality, Integrity and Availability
CL-PKC	Certificateless Public Key Cryptography
CL-PKE	Certificateless Public Key Encryption
CL-PKS	Certificateless Public Key Signature
CRC	Cyclic Redundancy Check
DH	Diffie-Hellman
DHP	Diffie-Hellman Problem
DLP	Discrete Logarithm Problem
DSA	Digital Signature Algorithm
ECAES	Elliptic Curve Authenticated Encryption Scheme
ECC	Elliptic Curve Cryptography
ECDH	Elliptic Curve Diffie-Hellman
ECDLP	Elliptic Curve Discrete Logarithm Problem
ECDSA	Elliptic Curve Digital Signature Algorithm
ECIES	Elliptic Curve Integrated Encryption Scheme
EDHOC	Ephemeral Diffie–Hellman Over COSE
ID-PKC	Identity-based Public Key Cryptography
IETF	Internet Engineering Task Force
IoT	Internet of Things
JS	Join Server
KDF	Key Derivation Function
KGC	Key Generation Center
LoRa	Long Range
LoRaWAN	Long Range Wide Area Network
LPWAN	Low Power Wide Area Network

MHDR	Message Header
MIC	Message Integrity Check
MITM	Man in the Middle
NS	Network Server
NwkKey	Network Key
NwkSKey	Network Session Key
OTAA	Over the Air Activation
PHDR	Physical Header
PHDR_CRC	Physical Header Cyclic Redundancy Check
PKC	Public Key Cryptography
PKI	Public Key Infrastructure
RSA	Rivest–Shamir–Adleman
TTP	Trusted Third Party

1

Introduction

Ericsson made a prediction that by 2025, there could be about 25 billion Internet of Things (IoT) devices [1]. All of these devices will be able to connect and communicate with the internet. IoT devices will play an important role in our infrastructure and our lives. It is, therefore, crucial that these devices can communicate securely and that we can trust the integrity and confidentiality of the communication. A common communication method for IoT devices is called Long Range Wide Area Network (LoRaWAN) [2]. LoRaWAN is a Low Power Wide Area Network (LPWAN), and it aims to fill a gap in the networks that are in use today. Figure 1.1 shows a rough view of what gap LoRaWAN covers.

Computer security has become an important topic for many since the Morris worm showcased the lack of computer security in 1988 [3]. The importance is shown by the recent regulation from the European Union, called General Data Protection Regulation (GDPR) that emphasizes the need for security in communication [4]. In computer security, there is a commonly used model to describe security. The model is composed of three components Confidentiality, Integrity and Availability and is known as the CIA triad [5]. Confidentiality is about the ability to hide information from everyone other than the intended recipient. Confidentiality can be achieved using encryption. Integrity is about trusting that the data is accurate. Integrity should also ensure that the data has not changed along the way to the recipient. Availability is about ensuring that the information is available at all times to the recipient.

The security of LoRaWAN is good as it uses a strong symmetric key encryption scheme, which provides good cryptographic strength [6]. LoRaWAN can use a method called Over the Air Activation (OTAA) to register a device to a network and to generate session keys to use in the encryption of messages [7]. The session keys are generated based on the root keys that are pre-configured on the device. However, there is a problem with key distribution and renewal of the root keys, making a device vulnerable if the root keys are compromised. Replacing the root keys would require physical access to the device, which would be troublesome. Implementing Public Key Cryptography (PKC) can provide a remedy if it becomes compromised. This thesis aims at researching the feasibility of implementing a PKC algorithm for the delegation of root keys in the LoRaWAN specification.

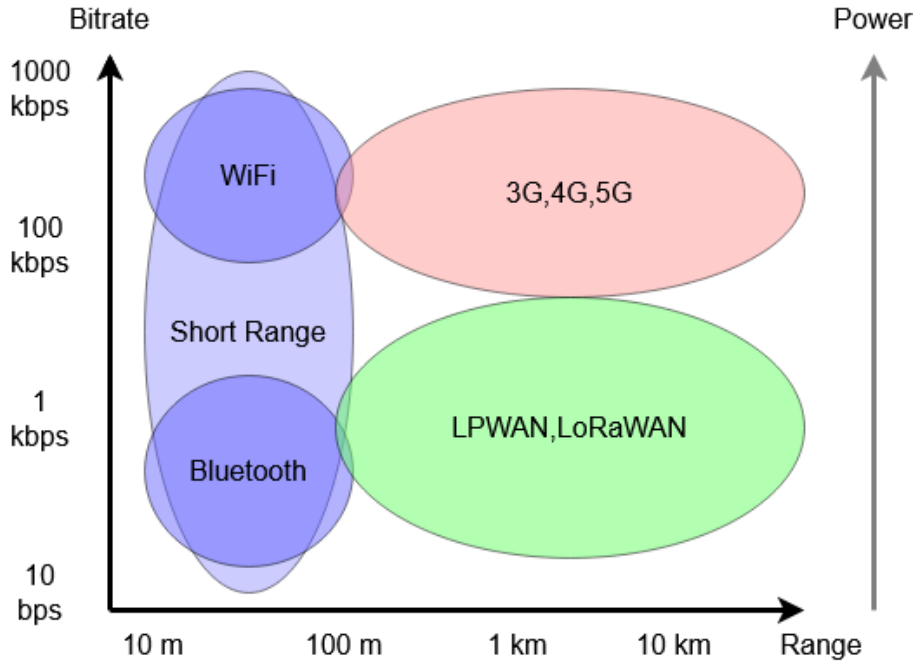

Figure 1.1: The network gap covered by LoRaWAN

1.1 Problem statement

Butun *et al.* [8] have, in their paper, presented a security risk analysis of the LoRaWAN specification. They explain that there is a problem with key distribution and renewal of root keys, which make a device vulnerable if the root keys are compromised. They suggest that an improvement to the specification is to implement a PKC for the generation of the root keys. However, Mahajan *et al.* [9] point out that using asymmetric keys takes considerably more time to encrypt or decrypt than using symmetric keys. This longer computation time resolves into less overall battery life and is one of the disadvantages of using asymmetric cryptography. An advantage of asymmetric cryptography is key distribution and management. There is then a trade-off that has to be made with regards to cryptographic strength and battery life.

1.2 Purpose

This thesis aims to investigate the feasibility of implementing and designing a PKC algorithm for generating and delegating the root keys used in the LoRaWAN specification. The ambition of the project is to add a new Over the Air Activation (OTAA) method for the LoRaWAN specification that is capable of delegating the OTAA root keys.

Furthermore this thesis aims to make the proposed method the start of a new authentication method in the LoRaWAN specification. The proposed method should conform to the LoRaWAN specification and not significantly increase battery con-

sumption.

1.3 Limitations

This thesis restricts itself to generating and delegating the OTAA keys of the LoRaWAN specification. This thesis does not change the way that LoRaWAN uses these keys for encryption as the security of the LoRaWAN encryption has previous work on it already [6, 8]. Additionally, this thesis does not aim to develop a new public key encryption scheme as that would be another undertaking in and of itself; instead, tested and proven methods and protocols is to be used. Evaluating the security of the implementation is not performed. The implementation should not make any modifications to the existing LoRaWAN system architecture. It should be possible to implement the proposed method on an existing LoRaWAN system with a software update. In this thesis, the focus is on the messages required to generate the root keys between the device and the LoRaWAN infrastructure.

1.4 Report layout

This thesis is structured as follows.

Chapter 1 introduces the background, problem, and purpose for this thesis.

Chapter 2 gives the technical background needed for this thesis. It begins with a simplified introduction to the cryptography and cryptographic protocols mentioned in this thesis. The chapter concludes by describing parts of theLoRaWAN specification.

Chapter 3 introduces related work.

Chapter 4 defines how to determine if the proposed method is feasible. It includes how the testing set-up was constructed along with the definition of feasibility used in this thesis.

Chapter 5 introduces the proposed LoRaWAN method, called PK-OTAA, and the logic behind it.

Chapter 6 shows the result of testing the LoRaWAN method.

Chapter 7 discusses the result and implementation of the LoRaWAN method.

Chapter 8 concludes this thesis.

2

Technical Background

This chapter contains technical information about different topics and technologies that are discussed or used in the project. It begins with general information about cryptography and knowledge about different cryptographic schemes and algorithms. It concludes with information about LoRaWAN.

2.1 Cryptography

This section is intended to help the reader understand some of the common terms and language used in cryptography. The primary source of information presented is from the book by Stallings [5]. It first explains some of the algebra involved. It then explains some of the mathematical problems that are the foundation for the cryptographic schemes.

2.1.1 Mathematical theory

There are a few common operations that are used in cryptographic schemes. They are called *Belongs*, *Greatest common divisor*, *Co-prime* and *Primitive root modulo n*. The operations are described below.

Belongs: For some item a and set X if a exist in the set X then $a \in X$.

Greatest common divisor: usually written $GCD(a, b) = c$, where c is the largest integer that divides both a and b.

Co-prime: Numbers a and b are considered to be *co-prime* if the only common divisor is one (1). A concrete definition is a and b is co-prime if $GCD(a, b) = 1$.

Primitive root modulo n: A number g is primitive root to modulo n if for every number a the co-prime to n is congruent to a power of $g \bmod n$. This expression $g^k \equiv a \bmod n$ where k is an integer would then only form numbers where a is *co-prime* to n.

2.1.2 Mathematical problems

The strength of different cryptographic schemes comes from the difficulty of solving certain mathematical operations. The ones discussed in this paper are the *factoring problem* and the *Discrete Logarithm Problem (DLP)*. The description of these can be found below [5].

The factoring problem concerns the difficulty of finding the prime factors of a positive number. This problem is thought to be exponential in complexity but this is not a verified fact [10]. The complexity of the best-known algorithm is of the size $10^{\sqrt{n \ln n}}$. Verification of a factored number is polynomial in time. If a number n is known and have the factors $p_1, p_2, \cdots, p_k$ then it is simple to multiply the factors together and compare it to n. However, in 1994 Shor discovered an algorithm that can factorize a number in polynomial time [11] using a quantum computer.

The Discrete Logarithm Problem (DLP) can be defined to try and find x in the equation $y = g^x \bmod p$ given g, y and p. The complexity is considered to be on par with the factoring problem. One of the fastest known algorithms to find x where p is a prime does so in the size $e^{((\ln p)^{1/3}(\ln(\ln p))^{2/3}}$ [5].

2.1.3 Terminology

There are multiple different concepts and terminologies used in cryptography. This section will describe some of the common concepts and terminologies used in this thesis.

Forward secrecy is a feature that protects against a compromised key. If a session key is compromised, only the session that used that key for encryption will be compromised. Forward secrecy also protects the session keys if the root key used to derive the session keys is compromised.

Ephemeral keys are keys that are being generated each time during a key establishment process and then discarded.

Replay Attacks is when an adversary copies a message and sends it again to the recipient. The adversary could also suppress the original message to the recipient and only allow for the adversary's copy to get through.

A **Man in the Middle (MITM)** attack is when an adversary is relaying and even altering messages that are being exchanged between two parties. Communication is usually susceptible to this kind of attack when there is no form of an authentication procedure. With two parties, A and B, and no authentication of any kind, A cannot know that they are communicating with B and the same is true the other way around.

A **Trusted Third Party (TTP)** is an entity that every user in the group trust. An example of this is passports that are issued by a government. A person with a passport can verify his/her identity to others if the others trust the government.

Key escrow, also known as a fair cryptosystem, is defined as a system where a TTP can access the secret keys. If there is a need to encrypt, decrypt, or sign a message as any user, the TTP can access the key of that user. Key escrow might be desirable for some systems or users, but is also heavily criticized [12].

2.1.4 Advanced Encryption Standard

The Advanced Encryption Standard (AES) is based on a proposal by Daemen *et al.* [13] that is known as Rijndael. AES is a symmetric key encryption scheme, and this means that it uses the same key to encrypt and decrypt a message. It uses a fixed block size of 128 bits and has three key sizes, 128, 192, and 256 bits. The key size specifies the number of rounds that the algorithm will perform on the input. It is ten rounds for 128-bit keys, 12 for 192-bit keys, and 14 for 256-bit keys. AES performs calculations on a four by four block called the *state*. With 16 bytes, $b_0, \cdots, b_{15}$, it is represented by a two-dimensional array:

$$\begin{pmatrix} b_0 & b_4 & b_8 & b_{12} \\ b_1 & b_5 & b_9 & b_{13} \\ b_2 & b_6 & b_{10} & b_{14} \\ b_3 & b_7 & b_{11} & b_{15} \end{pmatrix}$$

The AES algorithm is executed in four steps. The first step is called *Key Expansion*, and it derives 128-bit *round keys* from the symmetric key. It creates one key for every round of the algorithm plus one more. The second step is the initial execution of the *AddRoundKey* method, which performs bitwise xor operation with the state and the round key. The third step iterates four methods for 9, 11, or 13 rounds depending on the key size. The first method is *SubBytes*, which replaces bytes with others according to a lookup table. The second method is *ShiftRows*, where the rows are cyclically shifted to the left. The steps shifted are dependent on the row number, where the first row is zero. The third method is called *MixColumns*, and it performs a transform operation of all the bytes in a column. The last method is the *AddRoundKey* method. The final step in the AES algorithm is to make the last round which performs the methods *SubBytes*, *ShiftRows*, and *AddRoundKey*.

2.1.5 Public Key Cryptography

Public Key Cryptography (PKC), also known as asymmetric cryptography, is a class of cryptographic algorithms that uses pairs of keys: A public key that everyone can know and a private key that is only known by one entity. In general, the public key encrypts a message meant for the owner of the private key. PKC can also enable the private key to be used to create digital signatures that the public key can verify.

2.1.6 Public Key Infrastructure

Public Key Infrastructure (PKI) is an infrastructure for distributing and managing certificates in a PKC. The PKI binds a particular public key to a specific user and

issues a certificate that verifies the binding. A Certificate Authority (CA) is doing the binding by creating a certificate with the public key and then signing it. A user can then verify that a public key belongs to the specific user by checking the certificate. The user can then trust that the key belongs to that user. The CA is considered to be a TTP and has signed the key. The user is, therefore, able to trust the key because the user trusts the CA.

2.1.7 Diffie-Hellman

Diffie-Hellman (DH) was one of the first public-key protocols. DH allows two users that have no prior knowledge about each other on an insecure channel to establish a shared secret. The shared secret can then be used to establish a symmetric key for future communication. DH uses modular exponentiation ($g^a \bmod p$), and the cryptographic strength comes from the discrete logarithm problem [14]. Mathematically breaking DH is also known as the Diffie-Hellman Problem (DHP).

The mathematics in DH is quite simple, and using small numbers can be done by hand on paper. Here is an example with Alice (A) and Bob (B):

A and B agree to use modulus p and g as the base where g is primitive root modulo p. A then selects a as her secret value and computes $A = g^a \bmod p$ and sends it to B. B selects his secret value b and computes $B = g^b \bmod p$ and sends it to A. A can now compute $s \equiv B^a \bmod p$ and B compute $s \equiv A^b \bmod p$. They will compute the same value because:

$$B^a \equiv (g^b)^a \equiv (g^a)^b \equiv A^b \pmod{p}$$

An adversary listening to the exchanged messages will not be able to produce the same value. The adversary can read the messages A and B. However, the adversary does not know either a or b and is unable to produce the value s. If the discrete logarithm problem is considered hard to solve, it will be challenging to extract either a or b from the messages.

2.1.8 Rivest–Shamir–Adleman

The name RSA stands for Rivest, Shamir, and Adleman, which are the names from the creators of it [15]. RSA provides methods to encrypt, decrypt, and to create digital signatures, by using exponentials in modulo arithmetic to encrypt and decrypt messages. The cryptographic strength comes from the difficulty of the factoring problem on large numbers. RSA is a slow algorithm, and it requires long keys to be secure. Therefore, it is commonly used to establish a symmetric key between users.

Currently, there are not any known polynomial algorithms for efficiently factoring

large numbers. However, Shor's algorithm shows that a quantum computer is able to do it in polynomial time [16].

The modular exponentials in RSA works on the following premise $(m^e)^d \equiv m \mod n$, where m would be a message, (e, n) is the public key, and d is the private key. It is easy to calculate what e, d, and n to use. However, it is difficult to find the private key d when only knowing the value of the public key.

2.1.9 Digital Signature Algorithm

The Digital Signature Algorithm (DSA) uses a private key to digitally sign a message and a public key to verify it. DSA uses modular exponentiation, and the cryptographic strength comes from the discrete logarithmic problem [17].

DSA has a flaw in the selection of random parameters. The flaw can be exploited if a random value repeats on different signatures. An attacker can obtain the private key used during the signing [18].

2.1.10 Elliptic Curve Cryptography

The Elliptic Curve Cryptography (ECC) uses the Elliptic Curve Discrete Logarithm Problem (ECDLP), which compared to RSA and DSA, takes exponential time. Because of the exponential time, the parameters of ECC can be significantly smaller compared to RSA and DSA but with an equivalent level of security [19]. The smaller parameter size makes ECC, in theory, a good option for devices that have constraints in processing power, storage space, bandwidth, and battery power.

In ECC, an entity has a public and private key pair that is associated with a set of elliptic curve domain parameters. The domain parameters are specified over a finite field $\mathbb{F}_q$ and consist of the following:

$$T = (q, FR, a, b, G, n, h)$$

- q is a number specifying the prime power,
- FR is a method for representing field elements,
- a and $b \in \mathbb{F}_q$ that is part of the elliptic curve equation E $(y^2 = x^3 + ax + b)$,
- $G = (x_G, y_G)$ is a point,
- n is a prime number,
- h is an integer.

Generating a key pair in ECC using the domain parameters consists of two steps. First selecting a random or pseudo-random integer d in the interval $[1, n-1]$. Then

computing $Q = dG$ where Q becomes the public key and d the private key.

Elliptic Curve Diffie-Hellman (ECDH), Elliptic Curve Digital Signature Algorithm (ECDSA), and Elliptic Curve Authenticated Encryption Scheme (ECAES) are three conventional protocols that are based on ECC. ECDH is used to generate a shared secret between two entities, ECDSA is used to create and verify signatures, while ECAES is used to encrypt and decrypt data.

Exchanging a secret value using ECDH is done by using the private key of one entity A and a public key from another entity B. If the entities perform the same computations, they will get the same value. An example can be made with two entities, A and B, that form key pairs (d_A,Q_A) and (d_B,Q_B) that are associated with the same domain parameters. Assuming that both entities know each other's public keys, they can then compute $S = d_A Q_B = d_B Q_A = (x_P, y_P)$ where x_P is the secret value. A and B get the same value since:

$$Q = dG \rightarrow d_A Q_B = d_A d_B G = d_B d_A G = d_B Q_A$$

2.1.11 Elliptic Curve Integrated Encryption Scheme

Elliptic Curve Integrated Encryption Scheme (ECIES) is known as a hybrid scheme and is used to encrypt messages. It consist of five different functions: key agreement, KDF, encryption, message authentication, and hash [20]. The implementation details for the different functions are dependent on the specification and standard that is being followed. The key agreement function is used to exchange a shared secret between two entities. The KDF is a method to derive a set of keys from a value. The encryption function is used to encrypt messages. The message authentication function handles the message integrity. The hash function is used by the KDF and authentication functions.

2.1.12 Identity-based Public Key Cryptography

Created by Shamir [21], Identity-based Public Key Cryptography (ID-PKC) aimed to provide a cryptographic scheme based around users' identities. It uses a Key Generation Center (KGC) that acts as a TTP to generate secret keys for users based on their identity. The identity of a user could be common knowledge, such as a name and an address. A user can then send a message to another user by encrypting it using the common knowledge, and the receiving user can decrypt it using the secret key.

Shamir notes that the security of the scheme depends on four parts. The first part is the security of the underlying cryptographic functions. The second part is the security of the information stored at the KGC. Thirdly, it is the identity checks

performed by the KGC before issuing a new secret key. Finally, the fourth part is how securely the users store their secret key to prevent unauthorized usage.

As pointed out by Al-Riyami *et al.* [22], the ID-PKC has key escrow. As mentioned in Section 2.1.3, key escrow means that a TTP is able to decrypt or sign any message. The KGC is able to do this since it is in control of the identities in the system.

2.1.13 Certificateless Public Key Cryptography

Certificateless Public Key Cryptography (CL-PKC) created by Al-Riyami *et al.* [22] aims to resolve the problem of maintaining infrastructure to verify that a public key belongs to a specific user. CL-PKC is based on ID-PKC and is using a KGC, but it aims to resolve the key escrow problem of ID-PKC. It solves the key escrow problem with the KGC, generating a partial private key and transmitting it over a secure channel to a user. The user can use the partial private key to generate its private key and public key. Because the KGC does not generate the full private key, it cannot recreate it, thus it avoids the key escrow problem. A user can then transmit the generated public key to others. CL-PKC has two schemes for encryption and creating digital signatures. For encryption, it uses Certificateless Public Key Encryption (CL-PKE) to encrypt messages with the public key. The Certificateless Public Key Signature (CL-PKS) scheme is used to create a digital signature using the private key. The cryptographic strength from CL-PKE and CL-PKS comes from the Diffie-Hellman Problem (DHP).

Al-Riyami *et al.* describe a basic version of CL-PKE that uses seven algorithms, and they are as follows.

Setup: The KGC starts with running the setup. It takes a security parameter k, and a Bilinear Diffie-Hellman (BDH) parameter generator IG. It returns system parameters $params$ and a master-key s. The $params$ are publicly available, but the *master-key* is only known to the KGC.

System parameters $params =< \mathbb{G}_1, \mathbb{G}_2, e, n, P, P_0, H_1, H_2 >$:

- $\mathbb{G}_1$ is an additive group of prime order q
- $\mathbb{G}_2$ is a multiplicative group of the same order
- e is a map pairing of $\mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$
- n is the bit-length of plaintext
- P is a generator of $\mathbb{G}_1$
- $P_0 = sP$
- H_1, H_2 are hash functions

Partial Private Key Extract: The KGC runs this and it takes the *params*, $Master{-}key$, and the identity ID_A of user A. It first computes $Q_A = H_1(ID_A) \in \mathbb{G}_1^*$ and it produces the partial private key for A with $D_A = sQ_A \in \mathbb{G}_1^*$.

Set Secret Value: A runs this and it takes *params* and the identity of A. It returns the secret value x_A of A where x_A is selected at random.

Set Private Key: A runs this and it takes *params*, the partial private key D_A and the secret value x_A. It computes the full private key as $S_A = x_A D_A = x_A s Q_A \in \mathbb{G}_1^*$.

Set Public Key: A runs this and it takes *params* and the secret value x_A. It outputs the public key $P_A =< X_A, Y_A >$ where $X_A = x_A P$ and $Y_A = x_A P_0 = x_A s P$.

Encrypt: It takes *params*, a public key $P_A =< X_A, Y_A >$, identifier ID_A, and a message M. It computes $Q_A = H_1(ID_A) \in \mathbb{G}_1^*$ and chooses a random value r. The ciphertext is then $C =< U, V >=< rP, M \oplus H_2(e(Q_A, Y_A)^r) >$.

Decrypt: It takes *params*, a ciphertext $C =< U, V >$ and a private key S_A. It computes $M = V \oplus H_2(e(S_A, U))$.

2.1.14 Hybrid scheme

A public-key encryption scheme can be designed as a hybrid scheme. Such a scheme takes advantage of the strengths of asymmetric and symmetric key encryption. It uses a public key to encrypt or generate a symmetric key that is used to encrypt the actual message. This method is called key encapsulation [23]. The asymmetric part allows authentication to be established, and the symmetric part provides efficient encryption.

2.2 Electrical theory

This section introduces some basic theory in electricity to give the reader a better understanding of the theory used to measure the energy consumption of the proposed protocol. It starts with describing how to measure electrical power and how to calculate energy consumption.

2.2.1 Electrical power

Electrical power (P) is defined as work done per unit of time and is measured in the unit watt (W). It is calculated with the following formula:

$$P = I * V$$

Here I is the electric current measured in ampere (A), and V is the electric potential measured in voltage (V). A common way of measuring I or V is by using Ohm's Law, which describes a relationship between I, V, and the resistance (R).

$$I = \frac{V}{R}$$

Voltage is defined as the electric potential between two points. It is measured by checking the difference between the two points. The current can be measured by observing that, according to Ohm's Law, the current is proportional to the voltage drop over the resistor.

2.2.2 Energy consumption

Energy (E) is measured in joules (J) or also in watt-hour (Wh). Although power and energy appear similar, energy is the power consumed during a given time. Measuring energy consumption is then acquired by integrating the power over time.

$$E = \int P(t)dt$$

2.3 LoRaWAN

LoRaWAN is a specification for Low Power Wide Area Network (LPWAN) that defines a communication protocol and a network architecture [24]. This section examines some of the details in the LoRaWAN version 1.1 specification. It starts with describing the network architecture used in LoRaWAN. The second section explains how devices are activated and allowed to use the network to communicate. The third section defines the three different types of devices. The fourth section explains how LoRaWAN handles traffic congestion. The fifth section contains information about the cryptographic security of the protocol. The last section defines the different messages and their structure.

2.3.1 Network architecture

The LoRaWAN network architecture (as seen in Figure 2.1) primarily consists of five parts: devices, gateways, Network Servers (NS), Join Servers (JS), and Application Servers (AS). A device here can be anything, and its primary purpose is to transmit or receive messages using radio frequencies with the LoRa modulation. The gateways are responsible for receiving these messages and forward them to a network server or sending a message to a device. Networks servers are the managers of the gateways and have multiple responsibilities. The network servers handle the adaptive data rate, and duplicate messages as a device might be in the range of multiple gateways. The adaptive data rate allows a device that is closer to a gateway to transmit using a higher data rate and thus lowering the total transmission time. Devices that are further away can then have more time to transmit their message. However, the NS's primary purpose is to forward messages to the AS. The AS is the end receiver for a device, and what it does with the message is dependent on the purpose of the device

and application. The JS handles device activation, which is described in the next section.

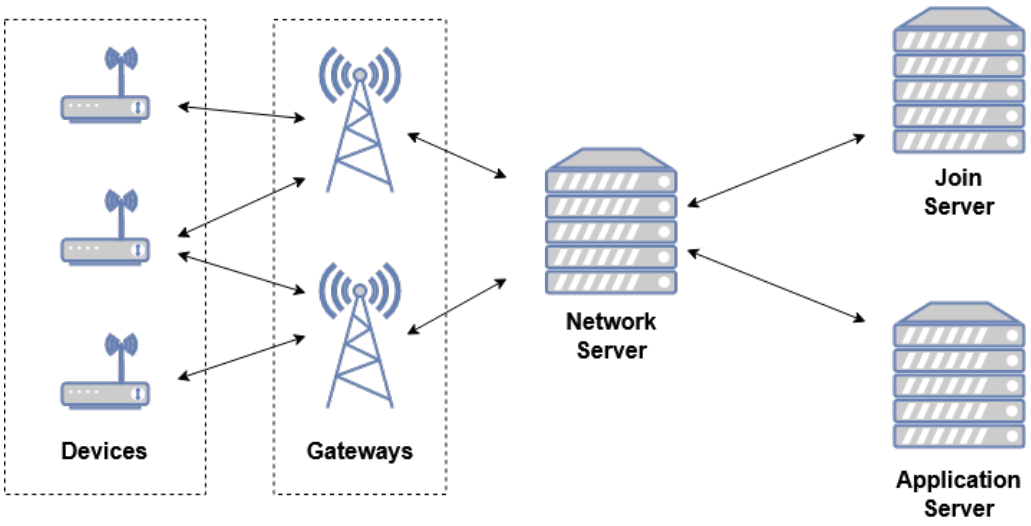

Figure 2.1: Overview of a LoRaWAN system architecture

2.3.2 Device activation

Before a message from a device is sent over the network, the device first has to join the network, and the process is called device activation. The goal for the activation is to have session keys for the network. The LoRaWAN specification specifies two ways that the activation is achieved [7]:

- Over the Air Activation (OTAA)
- Activation by Personalization (ABP)

Both of them need to be pre-configured with certain information, such as keys and network identifiers.

OTAA establishes the session keys by using two pre-configured root keys, a Network Key (NwkKey) and an Application Key (AppKey). The NwkKey is known to the NS, while the AppKey is known to the AS. The device uses this information to verify its identity to the servers. Device activation occurs with the following procedure, as shown in Figure 2.2. Further details of the structure of the messages can be found in Section 2.3.6. First, the device sends a join request to the NS containing three things: what network it wants to join (JoinEUI), its unique identity number (DevEUI), and a hash of the NwkKey, JoinEUI, and DevEUI. The NS then looks up the JS and sends a JoinReq message to it. Thirdly the JS verifies the request and sends a JoinAns back to the NS. Finally, the NS can send a Join Answer back to the device. With the activation complete, the device can then compute the session keys. ABP works by pre-configuring the session keys instead.

When using OTAA and the device has received the *Join-Accept* or the JS the *Join-Request* messages they are able to calculate the session keys. The session keys are

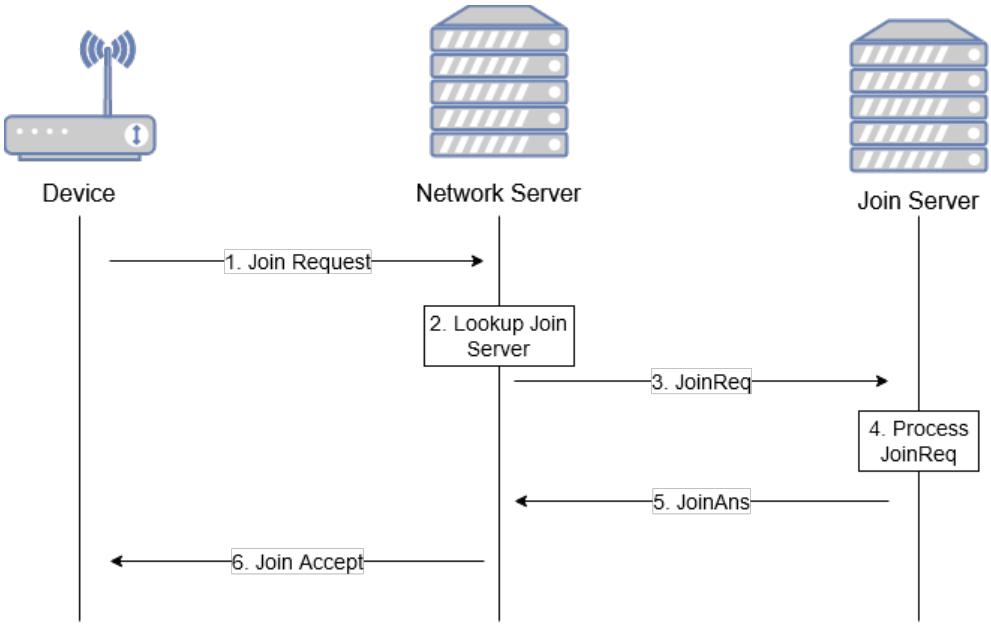

Figure 2.2: Message overview of OTAA

calculated in the following way:

$$
\begin{aligned}
AppSKey &= aes128\_encrypt(AppKey, 0x02 \mid JoinNonce \mid JoinEUI \\
&\mid DevNonce \mid pad_{16}) \\
FNwkSIntKey &= aes128\_encrypt(NwkKey, 0x01 \mid JoinNonce \mid JoinEUI \\
&\mid DevNonce \mid pad_{16}) \\
SNwkSIntKey &= aes128\_encrypt(NwkKey, 0x03 \mid JoinNonce \mid JoinEUI \\
&\mid DevNonce \mid pad_{16}) \\
NwkSEncKey &= aes128\_encrypt(NwkKey, 0x04 \mid JoinNonce \mid JoinEUI \\
&\mid DevNonce \mid pad_{16}) \\
JSIntKey &= aes128\_encrypt(NwkKey, 0x06 \mid DevEUI \mid pad_{16}) \\
JSEncKey &= aes128\_encrypt(NwkKey, 0x05 \mid DevEUI \mid pad_{16})
\end{aligned}
$$

Different keys have different purposes and uses. Table 2.1 shows a description of all the keys used in LoRaWAN.

2.3.3 Device classes

The LoRaWAN specification defines three types of device classes, A, B, and C. Class A is the most energy-efficient one as after sending a message, it allows for two short messages to be received before the device stops listening. Class B devices allow for

Key	Description
AppKey	**Application Key** is a root key used to derive other keys. Lasts the lifetime of the device.
NwkKey	**Network Key** is a root key used to derive other keys. Lasts the lifetime of the device.
AppSKey	**Application Session Key** is a key used to encrypt the payload between the device and Application Server.
FNwkSIntKey	**Forwarding Network Session Integrity Key** is used in the MIC calculation.
SNwkSIntKey	**Serving Network Session Integrity Key** is used in the MIC calculation.
NwkSEncKey	**Network Session Encryption Key** is used to encrypt and decrypt MAC commands sent to the device or Network Server.
JSIntKey	**Join Server Integrity Key** is used for the MIC calculation of Join-Accept or Rejoin-Request messages.
JSEncKey	**Join Server Encryption Key** is used to encrypt and decrypt Join-Accept messages in the Rejoin procedure.

Table 2.1: The purpose and uses of the different keys in LoRaWAN

more messages to be received on a scheduled basis. The schedule is synchronized with the gateway. Class C devices will listen to messages while it is not transmitting any.

2.3.4 Duty cycle

In radio communication, two messages can collide and inadvertently disrupt each other. The risk of this occurring can be reduced by scheduling or reducing the congestion on the radio frequency with a duty cycle. LoRaWAN specifies a maximum duty cycle for each device, where the exact allowed duty cycle is different for each country and is defined in a document published by the LoRa Alliance [25]. The duty cycle limits how much a device is allowed to transmit on the radio. If a device uses a duty cycle of 10%, it would be allowed to transmit for one time unit for every ten time units.

2.3.5 Security

LoRaWAN uses two symmetric keys for encrypting the messages with AES-128. When a device is sending a message to the AS, the message is first encrypted with the Application Session Key (AppSKey). After the encryption, a Message Integrity Check (MIC) is calculated using the encrypted message along with the Network Session Key (NwkSKey). The message is then protected against a third party listening on radio communication. The network operator is also unable to decipher the contents of the message but can verify its integrity.

The cryptographic security in LoRaWAN is good as it uses AES-128. AES-128 is considered as being the block cipher of choice for future applications [26]. However, that does not mean that the communication protocol is secure. Butun *et al.* [27] conclude that there are multiple attack vectors to LoRaWAN and that the security is dependent on the implementation. The authors state that there are a few critical mechanisms in the implementation that needs to be considered. The first, being the storage of keys in an embedded platform. Secondly, there are multiple security-critical frame counters. Thirdly, there is a separation of trust between the NS and AS. However, the AS still has to trust the network infrastructure as there is no integrity check of the messages. Finally, the LoRaWAN specification has provisions for backward compatibility, which is known as downgrade attacks and has been exploited by attackers on other protocols [28].

2.3.6 Message formats

The uplink or downlink messages sent over LoRa has to follow a specific format specified in LoRaWAN. All messages contain a Preamble, Physical Header (PHDR), Physical Header Cyclic Redundancy Check (PHDR_CRC) and a PHYPayload. A Cyclic Redundancy Check (CRC) is added to any uplink messages. Table 2.2 displays the structure.

Preamble	PHDR	PHDR_CRC	PHYPayload	CRC

Table 2.2: Messages structure of LoRa packet where the CRC is only added for uplink packets

The PHYPayload is different depending on the type of message that is sent. Every payload contains a Message Header (MHDR), some content and a Message Integrity Check (MIC). The MIC is used to see if the content has changed during the transmission. See Table 2.3 for the structure of these messages. The reader might notice that the Join-Accept does not have the MIC. That is because the Join-Accept is encrypted, and the MIC is included in it.

MHDR	MACPayload	MIC
MHDR	Join-Request or Rejoin-Request	MIC
MHDR	Join-Accept	

Table 2.3: The three different types of PHYPayloads

The MHDR seen in Table 2.4 contains the message type, some bits reserved for future use, and the major version. Different message types is shown in Table 2.5. The table shows that there can be confirmed or unconfirmed data up and data down message types. The difference between them is that the confirmed ones require an acknowledgment to be sent by the receiving party. The proprietary message type can be used to implement a non-standard message format. However, it requires that everyone understands the format. If an entity receives an unknown proprietary message, it will silently drop it.

Bit	7..5	4..2	1..0
MHDR	Message type	Reserved	Major version

Table 2.4: Content of the MHDR

Value	**Type**
000	Join-Request
001	Join-Accept
010	Unconfirmed Data Up
011	Unconfirmed Data Down
100	Confirmed Data Up
101	Confirmed Data Down
110	Rejoin-Request
111	Proprietary

Table 2.5: The different message types and their associated bit value

Table 2.3 shows three types of messages. The first type *MACPayload* is used to transmit data and to send MAC commands. MAC commands are used to manage the network between the device and NS. Table 2.6 shows the contents of the *MACPayload*. It has a frame header *FHDR*, an optional *FPort* field, and the data payload *FRMPayload*. The *FRMPayload* is variable in length and the maximum size of it is restricted to the size limit specified by LoRaWAN. The *FHDR* can be seen in Table 2.7. It contains the *DevAddr*, *FCtrl*, *FCnt*, and *FOpts* fields. The *DevAddr* is a value that is given to the device when joining the network. The *FCtrl* field has some control parameters that the network needs and the length of the *FOpts* field. The *FOpts* field is used to transport the MAC commands.

Bytes	7..22	0..1	0..M[1]
MACPayload	FHDR	FPort	FRMPayload

[1] The message size limit

Table 2.6: Contents of the MACPayload

The MIC for the *MACPayload* message is calculated over all the fields. The MIC is calculated differently depending on it being a up-link or down-link message. Up-link is when a message is coming from the device and it is going to one of the servers. Down-link is the opposite it is when a message is coming from the servers and is heading to the device. The down-link MIC is calculated as follows with B_0 in Table 2.8:

$$
\begin{aligned}
msg &= MHDR \mid FHDR \mid FPort \mid FRMPayload \\
cmac &= aes128\_cmac(SNwkSIntKey,\ B_0 \mid msg) \\
MIC &= cmac[0..3]
\end{aligned}
$$

Bytes	4	1	2	0..15
FHDR	DevAddr	FCtrl	FCnt	FOpts

Table 2.7: Contents of the FHDR

Bytes	1	2	2	1	4	4	1	1
B_0	0x49	ConfFCnt	0x00	Dir = 0x01	DevAddr	AFCntDwn or NFCntDwn	0x00	len(msg)

Table 2.8: The content of the B_0 block in a down-link message

The up-link MIC is instead calculated like this with B_0 and B_1 in Table 2.9:

$$\begin{aligned} msg &= MHDR \mid FHDR \mid FPort \mid FRMPayload \\ cmacS &= aes128\_cmac(SNwkSIntKey,\ B_1 \mid msg) \\ cmacF &= aes128\_cmac(FNwkSIntKey,\ B_0 \mid msg) \\ MIC &= cmacS[0..1] \mid cmacF[0..1] \end{aligned}$$

Bytes	1	2	1	1	1	4	4	1	1
B_0	0x49	0x00	0x00	0x00	Dir = 0x00	DevAddr	FCntUp	0x00	len(msg)
B_1	0x49	ConfFCnt	TxDr[1]	TxCh[2]	Dir = 0x00	DevAddr	FCntUp	0x00	len(msg)

[1] The data rate used
[2] The index of the channel used

Table 2.9: The contents of the B_0 and B_1 blocks in an up-link message

The second type of *PHYPayload* is the *Join-Request* or *Rejoin-Request*. The first one is used to join the network and derive the session keys. The second one is used to create a new pair of session keys. Table 2.10 show the content of a *Join-Request*. It has a *JoinEUI*, *DevEUI* and *DevNonce*. The *JoinEUI* is used to identify what network to join. The *DevEUI* is a identifier for the device. The *DevNonce* is a counter that is increased after every join attempt. The purpose of the *DevNonce* is to stop replay attacks. The MIC of the *Join-Request* message is calculated like this:

$$\begin{aligned} cmac &= aes128\_cmac(NwkKey,\ MHDR \mid JoinEUI \mid DevEUI \mid DevNonce) \\ MIC &= cmac[0..3] \end{aligned}$$

The third type of *PHYPayload* is the *Join-Accept*. Table 2.11 shows the contents of a *Join-Accept*. The *JoinNonce* is a unique value provided by the JS that is used

Bytes	8	8	2
Join-Request	JoinEUI	DevEUI	DevNonce

Table 2.10: The contents of a Join-request

to derive the session keys. The *Home_NetID* is an identifier for the home network of the device. *DevAddr* is the address of the device. *DLSettings* contains a list of parameters for the down-link. The *RxDelay* is the delay between sending and receiving. *CFList* is an optional field with content that is different depending on the region. The final field is the *MIC*.

Bytes	3	3	4	1	1	0..16	4
Join-Accept	JoinNonce	Home_NetID	DevAddr	DLSettings	RxDelay	CFList	MIC

Table 2.11: The contents of a Join-Accept

The *Join-Accept* is encrypted with the NwkKey to protect the contents as it is used to derive the session keys. How it is encrypted, and the *MIC* is calculated is shown below. The message is encrypted using the *aes128_decrypt* function. It is using a decrypt function as an optimization for devices. The message is decrypted using the *aes128_encrypt* function. This means that a device only needs to implement the encrypt function of AES.

$$
\begin{aligned}
cmac &= aes128\_cmac(JSIntKey, JoinReqType \mid JoinEUI \\
&\quad \mid DevNonce \mid MHDR \mid JoinNonce \mid NetID \mid DevAddr \\
&\quad \mid DLSettings \mid RxDelay \mid CFList) \\
MIC &= cmac[0..3] \\
\text{Join-Accept} &= aes128\_decrypt(NwkKey, JoinNonce \mid NetID \mid DevAddr \\
&\quad \mid DLSettings \mid RxDelay \mid CFList \mid MIC)
\end{aligned}
$$

3

Related Work

There has been ample work on LoRaWAN. The literature shows that LoRaWAN version 1.0 has some security vulnerabilities. Many of these vulnerabilities have since been fixed in version 1.1 and has improved the security of LoRaWAN. This chapter aims first to summarize security-related work for LoRaWAN specifically on key management related topics and explain how the work in this thesis is different. Secondly, the chapter will examine different work that has tried to improve the LoRaWAN specification. Finally, the chapter will summarize what has been learned from the related work.

3.1 Key Management

Butun *et al.* [8] are not the only ones concerned about the key management in LoRaWAN. This is apparent from the multiple works that have been published on the subject. This section aims to examine a few of them and compare them to this thesis.

3.1.1 Enhancing LoRaWAN Security through a Lightweight and Authenticated Key Management Approach

Sanchez-Iborra *et al.* [29] introduce a new key management approach for LoRaWAN. It is supposed to be lightweight, and it follows a recent standardization by the Internet Engineering Task Force (IETF). The approach is based on the use of Ephemeral Diffie–Hellman Over COSE (EDHOC) that combines the strength of public-key cryptography with the lightweight Concise Binary Object Representation (CBOR) and Object Signing and Encryption (COSE). They argue that LoRaWAN needs better key management for the session keys as they are derived from static root keys and rarely updated.

The authors decided to use LoRaWAN version 1.0 to compare against their approach. They do, however, mention LoRaWAN version 1.1 in their related work section. Why they decided to use an older version of LoRaWAN is not mentioned, but some of the vulnerabilities mentioned in this work are mitigated on the newer

version. Their main argument concerning the root keys being static still holds for the newer version.

In this paper, the authors present a new key management protocol using EDHOC and evaluate it based on the increase in message overhead compared to using Datagram Transport Layer Security (DTLS) for key management. EDHOC is, according to the authors, the more viable option as it has the shortest message length. However, the shortest length of a message in their work was 71 bytes, while the max message length needed to support all scenarios in LoRaWAN is 59 bytes. The proposed key management approach would then not be applicable to all scenarios. The key management approach by Sanchez-Iborra *et al.* is different from this thesis as the main focus is on the root keys, not the session keys.

3.1.2 A Survey on Certificate-Less Public Key Encryption for Authentication in a Smart IoT-Based LoRaWAN

Tiwari *et al.* [30] argue that CL-PKC is a good PKC to use in LoRaWAN. The argument is that CL-PKC overcomes two of the major drawbacks of a traditional PKI, CA, and key escrow. The authors do not go into detail how CL-PKC can be implemented in LoRaWAN, only that it is a good option. This work strengthens the need for in-depth research of using PKC in LoRaWAN. Compared to this work, this thesis will, in the evaluation, use CL-PKC. However, other options will also be considered, and the end goal is to compare a solution to the current one employed by LoRaWAN.

3.1.3 Key management with a trusted third party using LoRaWAN protocol

Ralambotiana [31] researches the known threats and flaws for key management with a Trusted Third Party (TTP) in LoRaWAN (version 1.0.2). The author then assesses how a key management solution from a company that was called Gemalto[1] handles these threats. Gemalto's solution is a TTP that manages the distribution and generation of the keys used in LoRaWAN. The solution is similar to the JS used in newer versions of LoRaWAN.

The author finds five vulnerabilities, where the first one is the possibility of compromising the AppKey. The second vulnerability is regarding undefined behavior if and when a counter is reset. Thirdly is the ability to perform brute force attacks on the MIC. The fourth vulnerability is being able to compromise the cipher mode encryption. The last vulnerability is the absence of explicit control for claiming devices.

The work is a thorough review of a key management solution's security, and the author proposes solutions to the vulnerabilities. The vulnerabilities and their solutions are something that can be considered when designing a key management system for

[1] Acquired by Thales Group in 2019, `https://www.thalesgroup.com`

LoRaWAN. Compared to this work, this thesis will not focus on introducing a new TTP to LoRaWAN and instead work from within the boundaries of the current specification.

3.1.4 An Improved Secure Key Management Scheme for LoRa System

Xing *et al.* [32] propose a key management scheme that is capable of remotely updating the keys in LoRaWAN. The scheme uses the Hierarchical Deterministic (HD) Wallet for key management and Elliptic Curve Diffie-Hellman (ECDH) for key agreement. The authors also apply the bi-directional hash algorithm to reduce the computational cost and the overhead in communication.

The authors compare the performance of different cryptographic schemes against each other. ECDH is shown to be faster than Diffie-Hellman (DH), but ECDH with a 256-bit key was faster than using it with a 192-bit key. Xing *et al.* do not discuss this result. It is difficult to know if this is an oversight or what caused the performance increase when using a longer key. The authors conclude that their scheme can enhance the security of LoRaWAN, and using bi-directional hash chains reduces the performance impact that the scheme introduces.

The protocol from the authors does not adhere to the lowest data rate specified by LoRaWAN. The max length of a message payload is 59 bytes with the lowest data rate. The first message in the protocol includes the device identifier, a value for the hash chain, the ECC public key, and a signature. The first three would sum to 73 bytes with a compressed public key. It is unknown how big the signature is as the authors do not specify the exact implementation. The first message would then be at least 73 bytes and would exceed the size limit.

This work is similar to this thesis as the authors design a method to generate the root keys, and they decide to use a PKC in their solution. Their method requires six messages to be exchanged before the root keys are generated. Two of those messages would also be message acknowledgments, and having that in the method increases the risk of a re-transmission. It also introduces two new components with the HD wallet and the hash chains. The authors include some data regarding the time and energy usage of their protocol, but they do not compare this to OTAA. Their protocol does not comply with the restrictions and design choices specified by the LoRaWAN specification.

3.1.5 Signcryption Method Suitable for Low-Power IoT Devices in a Wireless Sensor Network

Ting *et al.* [33] propose a new signcryption method for IoT devices in a Wireless Sensor Network (WSN). Signcryption is a public key scheme that simultaneously executes a digital signature function and encryption function. The authors implement their method in a sensor system that is using Identity Based Cryptography (IBC).

Their method is divided into two phases: offline and online. The offline phase computes a partial ciphertext without a known message. The online phase is done when there is a message to be sent and uses the partial ciphertext from the previous phase to encrypt the message.

The main similarities of this work and this thesis are the underlying network architecture that is used. The authors have developed a PKC method that works well in their wireless networks. However, the method is not directly transferable to LoRaWAN as the underlying system uses IBC.

3.1.6 An improved key distribution and updating mechanism for low power wide area networks (LPWAN)

Leent [34] investigates an answer to the question, "how can the distribution and updating capabilities of cryptographic keys on the application layer in LoRa be improved?". From the research question of the author, the work appears to be similar to the work in this thesis. Leent also comes to the conclusion that it is appropriate to use Elliptic Curves as the cryptographic scheme because of its lower computational requirements. The author solves the problem of device authentication by using ECDSA. The author shows that it is possible to use these schemes to distribute and update the static keys used in LoRaWAN. However, the author does not consider two essential aspects. The first is the actual impact on energy consumption. The second is the added message length that is needed to use their protocol. This thesis intends to consider and address these two aspects.

3.1.7 An Enhanced Key Management Scheme for LoRaWAN

Han *et al.* [35] propose a root key update scheme based on the Rabbit cipher embedded in a two-step Key Derivation Function (KDF). The authors build upon the idea that LoRaWAN allows for re-join requests that derive new session keys to be used. Han *et al.* introduce new commands that orchestrate the update procedure of the root keys. The update procedure is based on a stream cipher, and it also assumes that a device already has a set of root keys. This approach is interesting and is different from this thesis. This thesis will focus on using public-key cryptography to distribute root keys. The authors used their scheme's performance as a metric, while this thesis will use energy usage as a metric to determine the feasibility.

3.1.8 Secure Session Key Management Scheme to Meter-Reading System Based on LoRa Technology

Xia *et al.* [36] attempt to improve smart meters by using LoRa technology. The authors propose an improvement to the LoRaWAN OTAA method. The improvement comes in the form of the introduction of a Key Distribution Server (KDS) to the LoRaWAN infrastructure. The protocol also introduces a timestamp for the session keys. The KDS is tasked with ensuring that new keys are distributed after a key has expired. The authors created a key management scheme for the session keys

used in LoRaWAN. This thesis is different from this work as it intends to look into the delegation of the root keys.

3.2 LoRaWAN Security

This section aims to cover some related work that has been focusing on improving other aspects of LoRaWAN. It will examine work that has been done to improve the security aspects and how the authors have considered the impact on battery life or performance.

3.2.1 A Novel Chaining Encryption Algorithm for LPWAN IoT Network

Bidgoly *et al.* [37] propose a chaining encryption method that is suitable for use in LoRaWAN. They argue that chaining encryption is not suitable in LPWAN as there is the possibility of losing messages. The message loss is also not discovered by either the sender or the receiver. Since chaining encryption makes each cipher dependent on the previous cipher, message loss will cause the decryption of messages to fail. Message loss can always happen, but it can be handled with some form of message acknowledgment. In LPWAN, this is a costly operation for a device and for the network. The authors' method is based on introducing a new module called Appropriate Key Finder (AKF) that finds a key that has a hash that is equal to a part of the message. Their method is capable of handling message loss, and the AKF module does not have a significant impact on performance. The authors have produced an interesting work with regards to replacing the encryption method used by LoRaWAN.

3.2.2 AES-128 based secure low power communication for LoRaWAN IoT environments

Tsai *et al.* [38] propose a new communication scheme called Secure Low Power Communication (SeLPC) to be used in LoRaWAN. The scheme is supposed to reduce the number of encryption cycles of AES, which reduces the computational power used. The authors also incorporate D-Box and AppSKey update procedures. D-Box is a dynamic box that replaces the fixed S-Box used in AES, and this replacement is then increasing the security of the scheme. The authors show that SeLPC reduces the AES encryption rounds from 10 to 5 and can minimize the computational power used for AES by up to 26.2%. The authors do not introduce an update procedure for the NwkSKey but mention that it is for future work.

3.3 Summary

This thesis is not the first work to try and improve upon the key management aspect of LoRaWAN. Some of the literature introduces a new TTP to the specification, while others implement PKC in some form. The literature evaluates their work by

examining the performance increase or decrease. The primary difference between the related literature and this thesis is the focus on comparing battery usage. The work by Xing *et al.* shares the most similarities with this thesis (Section 3.1.4). The authors decide to use ECDH for its shorter key length and computational requirements. They propose a protocol that can update the root keys used in OTAA. However, they do not adhere to the limits set by the lowest data rate in the LoRaWAN specification. Although the authors provide data regarding energy consumption, they do not compare it to other activation methods such as OTAA. This thesis adheres to the message limit by specifying it as a condition for the protocol to be considered feasible, and it will compare the protocol to OTAA.

4

Methods

This chapter introduces and explains how the feasibility of PKC for delegation of root keys in LoRaWAN is produced. The first section defines how the PKC is evaluated to be feasible. The second section explains how the hardware is set-up and how measurements are taken. The final section explains how testing is performed.

4.1 Feasibility of Method

The definition of when something being feasible is that it can be done, effected, or accomplished. This is a broad definition, and this section defines when a PKC is considered feasible. The method is considered feasible, depending on the answer to the three conditions specified in Table 4.1.

These are the conditions that state the feasibility of the method. They are stated as questions to provide a simple overview of true or false. The conditions are further described and clarified below.

The method is considered to fit into the specification when it fulfills the following. It does not add or change anything other than adding a new activation method along with support for the new PKC. This means that the method should rely on the existing functionality provided in the specification. It should, for example, not change how AES is used or how the session keys are derived. This condition is essential since it hinders any changes or additions that alter the specification unnecessarily. The condition makes it easier to adopt into existing LoRaWAN devices and systems.

The 64-byte limit also comes from the LoRaWAN specification but has been put as a separate condition. It is separated since 64 bytes is the maximum size that is allowed with the lowest data rate setting in the LoRaWAN specification. If the method is exceeding the message limit, it would still be considered to fit in the

F1	Does the method fit in the LoRaWAN version 1.1 specification?
F2	Are all messages the method sends less than or equal to 64 bytes?
F3	Is the increase in energy consumption not significant?

Table 4.1: The condition that the proposed protocol need to meet to be feasible

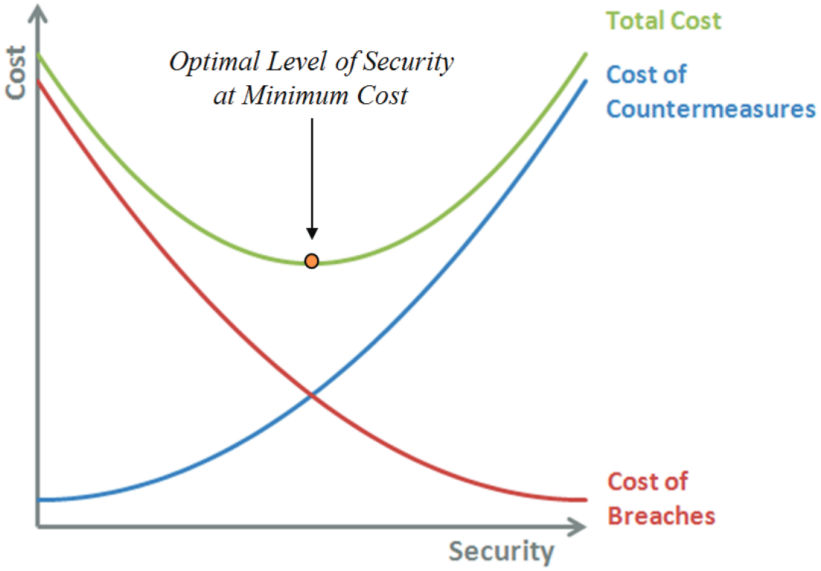

Figure 4.1: Security in relation to the cost, by Butun [39]

specification, but it would not support all devices and conditions. Therefore, the 64-byte limit has been added to ensure that the new method would still work for any device currently in use.

The final condition concerns the increase in energy consumption. The PKC will be implemented as a new OTAA method. Since it is an activation method, it means that it will run at least once as an initialization stage and perhaps a dozen more times if the root keys are changed periodically throughout the device's lifetime. The increase in energy consumption then has to be significant for the method to be unfeasible since OTAA runs once. The condition is difficult to evaluate since it is difficult to determine what a significant increase is. What can be considered to be a significant increase depends on the need for security and the cost of damages caused by the security being compromised. Figure 4.1 by Butun [39] shows how the relation between the cost of security and the cost of damages. Being able to renew the root keys will increase the security of the system. The increase in security comes at the cost of battery life in this case and the cost of damages could be in time or money. If the protocol decrease the battery life by 50% that is a significant increase, but if the cost of the damages is worse than implementing the countermeasure it might not be considered significant.

4.2 Design and Implementation of the testing set-up

Evaluating the PKC requires there to be a system where the user is in full control of the components involved. The designed and implemented PKC is going to change how the device interacts with the servers. Because of this, an entire LoRaWAN

Raspberry Pi	Model Zero	Model 3B
CPU Cores	1	4
CPU Frequency	1GHz	1.2GHz
RAM	512MB	1GB

Table 4.2: Hardware specifications of the Raspberry Pi models used

infrastructure needs to be replicated. This section starts presenting the hardware used and how it is configured. Following that is an introduction of the LoRaWAN network infrastructure used. It finishes with an explanation of how measurements of the power consumption are obtained.

4.2.1 Hardware

LoRa communication has two essential hardware parts, a device, and a gateway. Both of them consist of two other essential parts, a computing unit, and a LoRa radio chip. In this thesis, the computing units are two Raspberry Pi computers. They were selected because of their versatility and ease of use. Two different models of Raspberry Pi computers were used, the device used the Pi Zero model while the gateway used the Pi 3B model. Table 4.2 presents the specifications for the models. The Pi Zero has a 1GHz single-core processor and 512 MB of RAM [40]. The Pi 3B has a 1.2 GHz quad-core processor with 1 GB of RAM [41]. Both models use an SD card to store the operating system along with any additional files or programs.

The radio chips used are called RFM96, and they are using an SX1276 LoRa module. Figure 4.2 shows the radio chip used. The chip was chosen since it is inexpensive and easy to use with a Raspberry Pi. How the RFM96 is wired up to the Raspberry Pi can be seen in Figure 4.3. Pins with the same number on both the RFM96 and Raspberry Pi are connected. The numbers indicate what physical pin number it is on the Raspberry Pi.

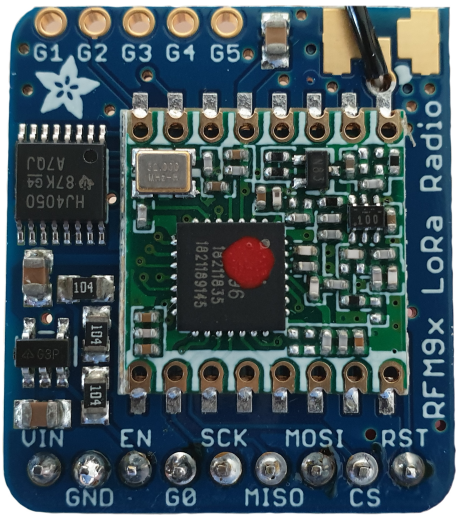

Figure 4.2: The RFM96 LoRa radio used

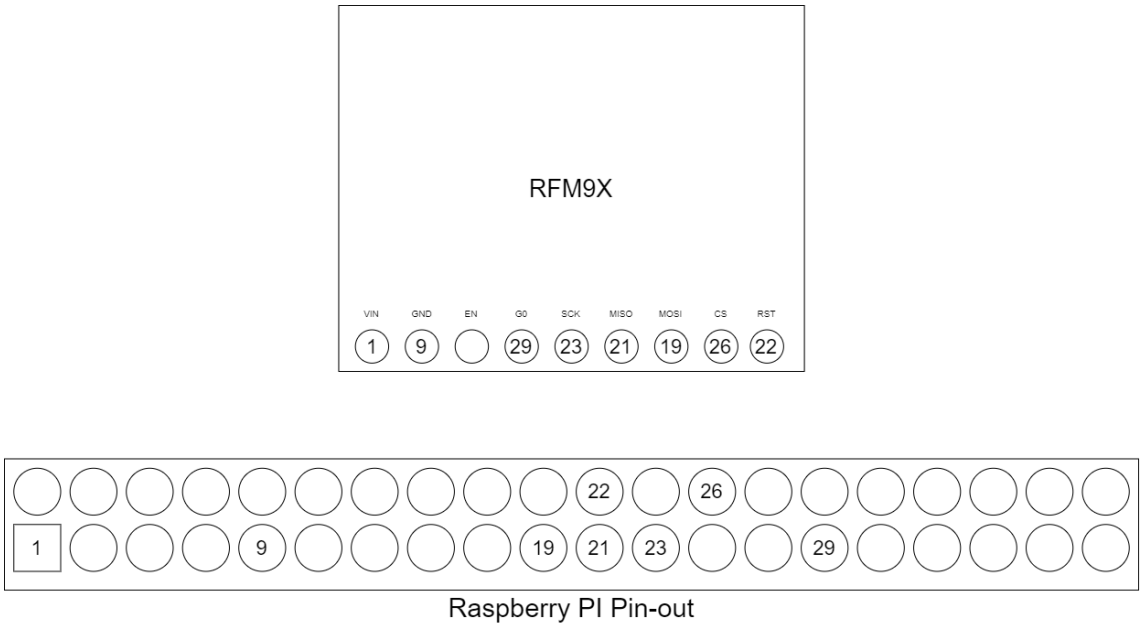

Figure 4.3: Wiring of the RFM96 and the Raspberry Pi

There is a downside to using the RFM96 chip, as it only provides one channel. This restricts the chip to only being able to use one channel at a time. The chip is therefore unable to send and receive at the same time. Constructing a gateway for LoRaWAN with only a single channel is not recommended as messages might be missed. However, for this thesis, it is not a problem since the testing is performed in an environment where there will only be one device.

The full hardware set-up can be seen in Figure 4.4. The gateway is on the left side, and the device is in the middle. On the right, there is a board with a button that is connected to the device. The button is used to start a program on the device.

4.2.2 LoRaWAN infrastructure

When choosing what LoRaWAN infrastructure to use, there are two main options: to build a custom solution or use an existing solution. There are various benefits and consequences for using either one. This project will be using a custom solution for the simplicity that it provides. Using an existing solution increases the complexity when implementing the designed protocol. The custom solution instead makes it possible to focus on only the specific parts that need to change. The complexity of an existing solution comes from its purpose of being used in an actual production environment. The custom solution can, in this case, be more straightforward as it does not need to consider multiple devices and to manage the network. The purpose of the custom solution is to provide the device with the messages it needs to complete the activation process.

A custom solution can be implemented on the gateway that would receive a message from the device and respond to it with a predefined and calculated message. Having

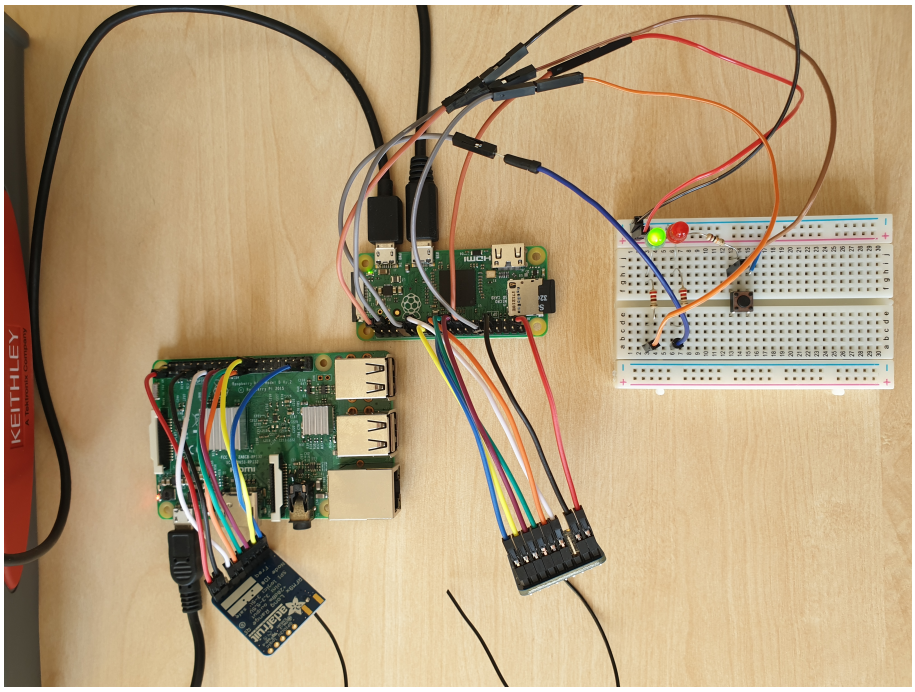

Figure 4.4: Image of the physical set-up of the gateway (left) and device (middle)

the gateway simulating the LoRaWAN also removes any additional latency from using a server. Using predefined messages makes it easier to reproduce the result, and the response time should be constant as there is no need to do any calculations. The device will think that it is using a real LoRaWAN infrastructure. Although the device will send the same message and receive the same response, it is still required to generate the message to send and receive the response. The device does not gain any performance increase from this method.

The gateway performs all the tasks of authenticating a device in the same way as a traditional LoRaWAN infrastructure. It is then essential to ensure that the message that it responds with is done according to the specification. To achieve correctness, an actual LoRaWAN network stack was set up using software from The Things Network.[1] Using the software from The Things Network, it is also possible to ensure that the device and gateway can work in a production version of LoRaWAN.

4.2.3 Measuring energy consumption

The energy consumption can be measured by taking measurements of the voltage (V) and the current (A) that are used. Since the device is normally powered through USB, the voltage should be around 5V at all times. The measurements are taken from a power supply that has a built-in data logger that makes it capable of powering the device and collect data about the usage. The data logger takes samples of the voltage and current used at regular intervals.

[1] `https://www.thethingsnetwork.org/`

Figure 4.5: The front panel of the Keithley 2280S

The data accuracy depends on the sample rate used. If spikes in voltage and current usage are between samples, then that will be concealed in the result. Therefore, it is essential to try and have a high sample rate and run the tests multiple times.

The data logger used in this thesis is the Keithley 2280S, which is also a power supply. It will be used to power the device and log the values that it supplies. It is capable of reliably doing measurements every 5 ms. However, using a higher sample rate lowers the accuracy of the measurements. For this thesis, the Keithley will take samples every 12.6 ms, as this should be more than enough to capture most of the sudden spikes that will occur. The Keithley can be seen in Figure 4.5.

Because the data logger acts as a power supply, the device will be powered by it. Figure 4.6 shows how the data logger is connected to the device. The data logger provides two ports to connect the positive and negative cables of a circuit. The Raspberry Pi can be powered through a USB port. The data logger is then connected to the Raspberry Pi through a USB cable that has been separated into four sub-cables. The USB cable can be divided into a 5V, data negative, data positive, and ground cables. From these, the 5V is connected to the positive side of the data logger and the ground to the negative. Figure 4.7 shows the different sub-cables where they have also been color-coded as red, white, green, and black [42]. It is possible to power the Raspberry Pi through the input and output pins. However, it will then not be possible to regulate the supplied voltage. This can cause damage to the device or unpredictable behavior if the voltage that is supplied fluctuates.

4.3 Testing method

This section defines how to obtain results that can be used to determine an answer to the energy consumption condition specified previously in Section 4.1. It begins with addressing methods used to mitigate the variations in the sample points obtained from the data logger. The section concludes with how the tests are executed and

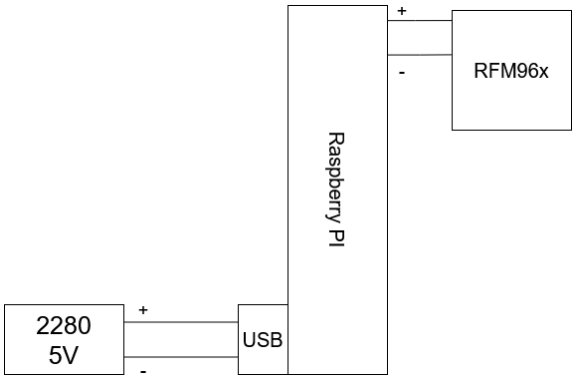

Figure 4.6: Power setup for the device using the data logger

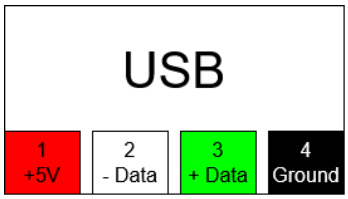

Figure 4.7: The pins in a USB cable [42]

data is obtained.

4.3.1 Handling data fluctuation

The data logger takes samples of the current and voltage at a single moment in time. Because of that, there is the possibility of missing small fluctuations in the supplied current. In order to reduce the impact of missing fluctuations, it is important to perform the same test multiple times. Making testing the code easier is crucial, therefore, the code will be divided into a set of segments that can be performed in a loop. Where to divide the code is dependent on the design and introduced steps of the new method. Normal OTAA can be divided into three segments. The segments would be sending, receiving, and deriving the session keys. However, the receiving segment is a consequence of the sending segment, and thus it is better to perceive them as one segment. When gathering data on the current and voltage for OTAA it will be divided into two segments, the sending and receiving segment and session key segment.

With a set of segments of code defined, the tests can then be executed. The segments will be executed 100 times, and the result of a segment will then be the mean of all the executions. Executing the segment multiple times should remove any fluctuation that the data logger may give because of the sample rate. A high sample rate is used as well to decrease the fluctuation.

4.3.2 Executing tests

Starting the tests will be done by the Raspberry Pi through a physical button. A button will be used because remotely connecting to the Raspberry Pi requires a separate connection to it. This connection adds extra energy consumption that could distort the result. Another issue is that the Raspberry Pi Zero has two USB ports, one for only power and another that can take power and data. If the port that can take data is used, a remote connection can be established. However, the data logger will no longer supply power to the Raspberry Pi.

The button is connected to the Raspberry Pi, along with two indicators of LED lights. When the Raspberry Pi boots, it loads a program that lights one of the LED lights indicating that it is ready and listens for a button press. If the button is pressed, the second LED is turned on, and it launches the tests. When the tests have finished, the program reboots the Raspberry Pi and turns off the LED lights.

Collecting data of voltage and current is started manually after starting the tests on the Raspberry Pi. The data logger gives access to a web interface for remote control. This interface is used to start data collection. It is also possible to connect the Raspberry Pi to the data logger through a digital input and output interface to control it. Using the digital interface on the data logger would make the data collection start at the same time as the test starts. This way of starting the collection was discarded as there is no need to start the collection at the same time. It takes a few seconds before the program starts performing its tests, and thus manually starting it will be good enough.

5

Design and Implementation

In this chapter, the process and choices behind the proposed protocol PK-OTAA are presented. The chapter begins with which PKC it is based on and why. It follows up by introducing some limitations and considerations when designing the protocol. Finally, the proposed protocol is presented and explained.

5.1 Selecting a Public Key Scheme

This section explains the reasoning for selecting a specific PKC to use in the protocol. Multiple cryptographic schemes can be used for the protocol. This thesis evaluates using Rivest–Shamir–Adleman (RSA), Elliptic Curve Cryptography (ECC), and Certificateless Public Key Cryptography (CL-PKC) for the protocol. The first two are regularly used in different security protocols today. The last one was proposed by Tiwari *et al.* [30] (refer to Section 3.1.2) to be a good option to use in LoRaWAN. The schemes are evaluated based on the conditions previously specified for the protocol in Table 4.1. The same conditions are used since the underlying scheme of the protocol has to fulfill them.

RSA is the standard used today in securing web connections through *HTTPS* [43]. Using RSA would require using long keys to achieve good encryption security. With a longer key, the size of the ciphertext would then also be longer. Encrypting a message with a 2048-bit RSA key would then produce a cipher size of 256 bytes [44]. This length exceeds the one set in the condition of 64 bytes. Using RSA to create a signature has the same problem. The signature size would also be 256 bytes with a 2048-bit key. RSA would also introduce the usage of a certificate infrastructure. Because of this RSA, is not considered to be feasible for use in LoRaWAN.

In comparison, ECC uses shorter keys and a 224-bit key provides equivalent cryptographic security as a 2048-bit RSA key [19]. Table 5.1 presents the different key sizes of ECC and RSA compared to AES. Figure 5.1 shows the relation visually. The table and figure demonstrate that the RSA key size escalates rapidly to achieve the same cryptographic security. Encrypting messages with ECC can be done by using Elliptic Curve Integrated Encryption Scheme (ECIES). However, as it includes the public key used along with an integrity code, the message length would exceed 64 bytes. In ECC, a public key can be compressed to be half the size plus one byte.

When using a private key of size 224 bits (28 bytes), the public key would be two times that, but when compressed, it would be 232 bits (29 bytes). Considering that the Join Request in the current OTAA uses 23 bytes, just by adding the public key makes the message size 52 bytes. That leaves 12 bytes for the other parameters used in ECIES and the encrypted payload. There could also be a need to increase the key size to strengthen cryptographic security further. The available space would then be even less. It would then not be feasible to use ECIES to encrypt generated keys or key material used to generate them.

AES key size (bits)	**ECC key size (bits)**	**RSA key size (bits)**
80	160	1,024
112	224	2,048
128	256	3,072
192	384	7,680
256	512	15,360

Table 5.1: Shows the keys size needed to have equivalent cryptographic security compared to AES among ECC, and RSA based on [19]

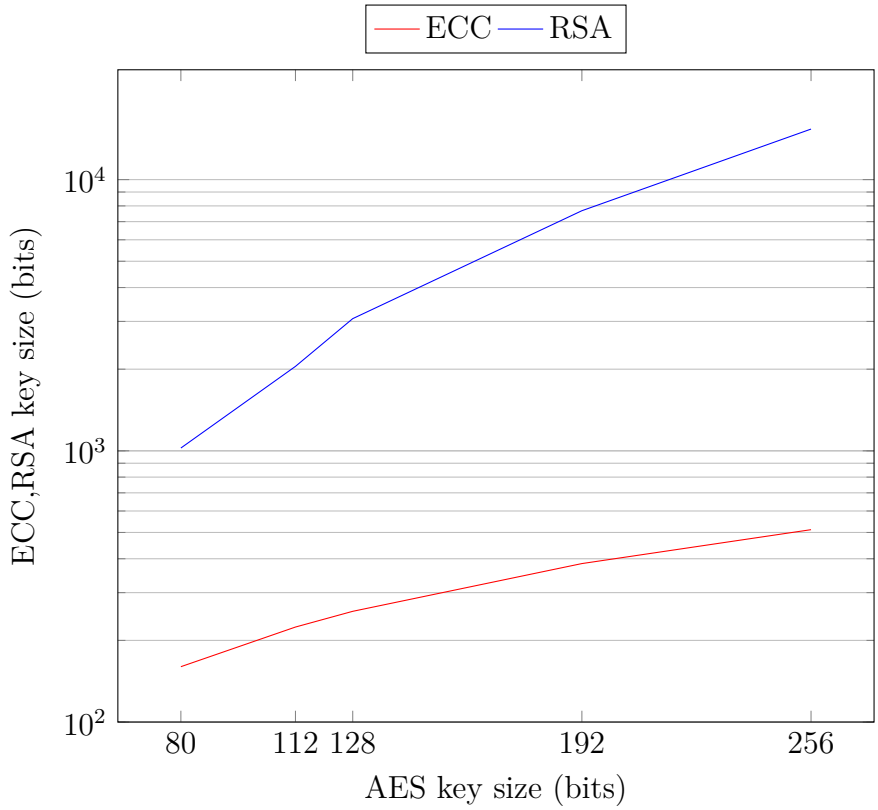

Figure 5.1: The logarithmic growth in key size to get equivalent cryptographic security compared to AES among ECC, and RSA based on [19]

CL-PKC introduces a trusted entity called the Key Generation Center (KGC). The task of the KGC is to create partial keys to transfer to different entities securely. The partial keys can then be used to create a private key. The private key can

then be used to create the public key. Because the KGC is a trusted entity and has generated the partial key, any device will be able to verify the identity of a public key. Yao *et al.* [45] implement CL-PKC using ECC which is supposed to make it more lightweight. However, encrypting a message with this method increases the size of the encrypted message to three times the key size plus the length of the message to encrypt. Using a 224-bit key, the message size would be 768 bits (96 bytes), and the message limit is broken. Thus, using CL-PKC for the protocol is not feasible.

Using these schemes for encryption in the protocol does not appear to be feasible. However, examining how ECIES functions, it is discovered that it uses Elliptic Curve Diffie-Hellman (ECDH) to establish a shared secret. The secret is then turned into a symmetric key, and AES is used to encrypt the payload with the secret. ECDH requires both parties to know each other's public key. The protocol would then only need to add the public key of the device to the Join Request. Using ECDH would then be a viable option to use in the protocol.

The *F3* condition in Table 4.1 have not yet been evaluated for the different schemes. The schemes are evaluated based on the number of CPU cycles that are increased compared to the AES128-bit encryption scheme. The increase in CPU cycles is used as the increase in cycles correlate to power usage. Data on the CPU cycles is presented in Table 5.2 and is taken from ECRYPT Benchmarking of Cryptographic Systems (eBACS) [46]. The data from eBACS does not contain measurements for CL-PKC and ECIES. Because Yao *et al.* implemented a CL-PKC version using ECDSA and ECIES, an assumption can then be made that the cycles that it uses should be more than just using ECIES, but it is excluded from the table. It is possible to estimate the cycles that ECIES uses as it uses ECDH to exchange an ephemeral key that is used to encrypt a message using AES. The assumption can then be made that ECIES use at least as much as AES and ECDH combined.

Scheme	**Cycles**	**Increase**
AES128-bit	6,688	0%
RSA2048-bit	726,420,470	10,861,501%
CL-PKC	N/A[1]	N/A
ECIES	1,646,041[2]	246,119%
ECDH[3]	1,639,353	245,119%

[1] No data available for comparison
[2] The assumed number of cycles based on ECDH and AES
[3] Based on the NIST P-256 Elliptic Curve

Table 5.2: CPU cycles used to execute the cryptographic schemes, based on data from eBACS [46]

The data from Table 5.2 show that AES is using less cycles compared to the other PKC candidates. RSA is using 700 million more cycles compared to AES. ECIES and ECDH is compared to that only using around 1.6 million more cycles. RSA appears to be the one that has the most significant increase in CPU cycles used. The

energy consumed would then also be significantly increased. ECC based schemes when compared to RSA, can then be considered to be feasible.

Based on the result above, the PKC that will be used for the protocol is ECC. The main argument for using it is the shorter key length that it requires compared to other PKCs. The shorter key length makes ECC use less computing power when using the key in calculations. This is ideal in a device that is powered by a battery and have access to limited hardware. The protocol will use ECDH to exchange a secret value that is used to generate the root keys. The key exchange uses static-ephemeral ECDH to derive the secret value. Although the scheme provides good security, it does not provide any authentication. The proposed protocol would need to consider this when receiving messages. Table 5.3 present a summary of the answers to the feasibility questions in Table 4.1 for the different evaluated schemes.

Scheme	F1	F2	F3
RSA	✗	✗	✗
ECIES	✓	✗	✓
CL-PKC	✓	✗	✓
ECDH	✓	✓	✓

Table 5.3: Feasibility summary of the various PKC schemes

5.2 Design limitations and considerations

When designing the protocol, there are three main aspects to consider. One aspect is the limited hardware of the device. The limited hardware of a device can be reduced computational power and available memory. The hardware might also be running on a processor architecture that does not support specific calculations or cannot perform them fast. The hardware can have specialized components that can perform specific calculations with high-speed and efficiency. That could restrict the type of calculation that the protocol or the PKC can use. Another limitation is the restriction in the LoRaWAN specification as different data rates are restricting the maximum size of packets. The protocol will need to support the lowest data rate, restricting the amount of data in a message, and being able to split the data into multiple messages. Finally, there are changes of trust that is introduced with a PKC. The root key could previously be used to authenticate a device since the device and the server shared root keys. The device and server might not share their public keys over an authenticated and secure channel, and because of that, neither of them can determine if the identity is legitimate. Table 5.4 shows a set of requirements that needs to be considered when designing the protocol.

5.3 Proposed protocol

This section introduces and explains the proposed protocol. Figure 5.2 shows the message flow of the protocol, which is almost identical to the original OTAA protocol. The significant changes happen in the first message and how the message is

RQ1	Minimize introducing too many new algorithms
RQ2	Reuse algorithms from OTAA
RQ3	Support the lowest data rate with a maximum content length of 59 bytes
RQ4	The device should be able to trust the server

Table 5.4: The requirements to consider when designing the protocol

handled by the NS, JS, and AS. For simplicity and clarity, the names of the messages in the LoRaWAN specification has been altered.

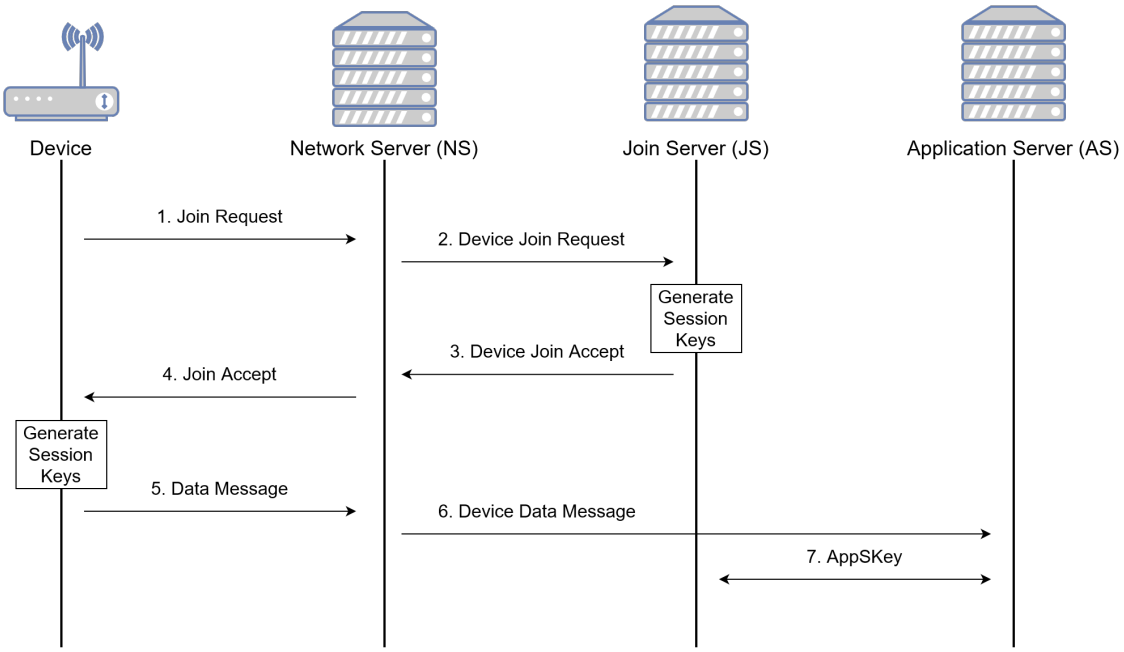

Figure 5.2: Message flow of the proposed protocol

The device is not pre-configured with the root keys (NwkKey and AppKey), but is instead configured with a public key to the JS. When the device starts, it generates a new key pair, a private and a public key. Using the private key it calculates a secret with the public key of the JS. The secret is then used to derive the root keys. The generated public key is then added to the join request and then sent to the NS. The NS receives the Join Request and proceeds to send a message to the JS containing the Join Request. The JS will then be able to calculate the same secret and derive the same root keys, register the device to the network, and generate session keys. After the JS has the session keys, the proposed protocol will then proceed to use the regular OTAA protocol from that point forward. The content of the changed message can be seen in Table 5.5. The table shows the original content in OTAA, and using the new method.

The proposed protocol uses ECDH where the JS is using a static public key, and the device is using an ephemeral public key. The protocol thus follows the recommendations from NIST [47] regarding the usage of one static key and one ephemeral key. The secret value that the protocol appends to the message is done as follows:

Message Type	Content	MIC
Join Request	JoinEUI, DevEUI, DevNonce	aes_cmac(NwkKey, MHDR | JoinEUI | DevEUI | DevNonce)
Join Request ECDH	JoinEUI, DevEUI, DevNonce, P_d	aes_cmac(NwkKey, MHDR | JoinEUI | DevEUI | DevNonce | P_d)

Table 5.5: The content of the messages in the proposed protocol

- Device: Generate ephemeral ECC key pairs
- Device: Calculate shared secret $s = S_d * P_{JS}$
- Device: Derive root keys from s
- Device: Sends Join Request ECDH
- Join Server: Receive Join Request ECDH
- Join Server: Calculate shared secret $s = S_{JS} * P_d$
- Join Server: Derive root keys and verifies MIC
- Join Server: Derive session keys
- Join Server: Send Join Answer to Device
- Device: Receive Join Answer
- Device: Derive session keys

How the keys interact with each other can be seen in Figure 5.3. The dotted lines show that a key is derived from the originating source material. The secret is the source material to derive the root keys. The root keys are then the source material to derive additional keys.

NIST recommends that a Key Derivation Function (KDF) is to be used on the shared secret to derive the key [47]. The protocol uses a hash algorithm on the shared secret. The algorithm produces a 256-bit (32 bytes) string that is split into two to create the root keys. The hash algorithm used is *BLAKE2s*, which is supposedly a fast algorithm for 8 to 32-bit processors [48], which a lot of the lower-end hardware is using.

With key agreement protocols, there is a risk of an attack called Man in the Middle (MITM). ECDH is also vulnerable to this attack. However, because the proposed protocol uses one static public key, the attack is thwarted. Figure 5.4 shows a flow of messages in how it is mitigated. The device creates the ephemeral keys (S_D, P_D)

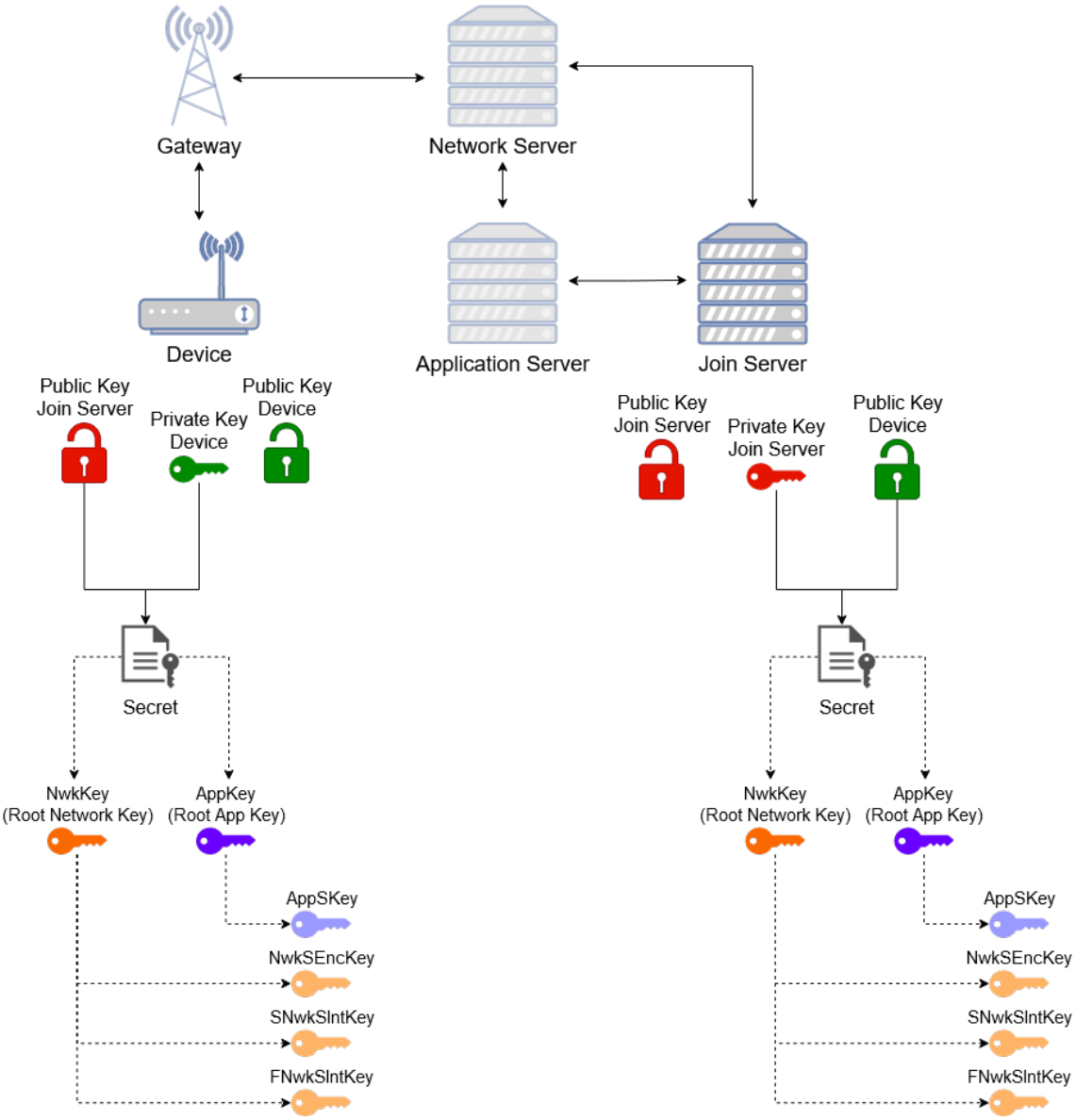

Figure 5.3: The keys used in the modified protocol

and broadcasts the public key P_D. A Malicious Device (MD) intercepts the message and can create root keys. However, it is unable to create the same root keys as the device. It can generate every possible private key and derive the root keys, but this would be the same as just trying every possible root key. This approach would take a considerable amount of time, and it would not be feasible.

Figure 5.4 shows four scenarios. In the first and second scenarios, the MD intercepts the message. It then changes the content of the message and sends it to the JS. The JS then responds with a message c'. The device can decrypt this message, but it will fail to confirm the MIC, and the attack fails. The MD could decrypt c' and create a new message c'', but the same problem would occur. In the third scenario, the MD would forward the message from the device. The JS would respond with the message c, but the MD would be unable to read this message. The MD could

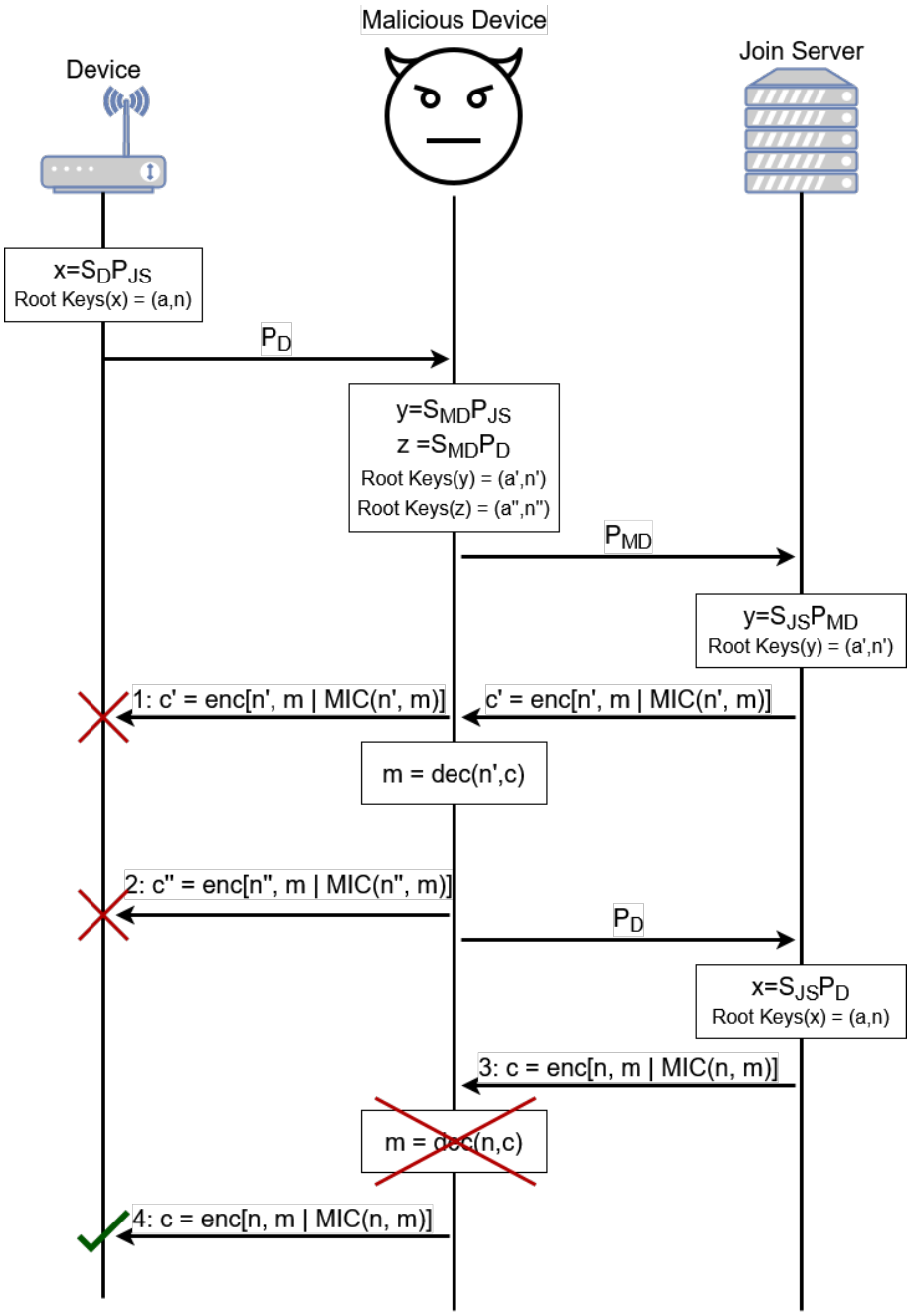

Figure 5.4: An MITM attack being thwarted by the protocol

modify the message, but the device will again fail to confirm the MIC. The last scenario is when the device receives the message c and can confirm that it is correct. The MITM attack is mitigated because the public key of the JS has been given over a secure channel.

The security of ECC is not only dependent on the size of the keys used. Another important aspect is the curve that is used. It is possible to select the parameters and define a custom curve, but there is a chance that the curve is weak. A weak curve can be susceptible to different attacks [19]. Because of this, it is recommended to use

Bytes	1	8	8	2	33	4
Join-Request ECDH	MHDR	JoinEUI	DevEUI	DevNonce	P_D	MIC

Table 5.6: The contents of the modified Join-request

a standard curve. NIST has a few curves that they recommend the US government to use [49]. In the implementation of the proposed protocol the *BrainpoolP256r1* curve by Merkle *et al.* [50] is used. Because it is a 256-bit curve, it is using 256-bit (32 bytes) long keys, and a public key would be 64 bytes long. However, after compressing the public key, it is 33 bytes long. The message size of the join request is shown in Table 5.6, where the total message length is 56 bytes.

Table 5.4 defines four requirements that the protocol needs to fulfill. The first requirement is met as the protocol introduces two new algorithms, ECDH and a hash function. While the second requirement is also met, it could be improved further by incorporating AES. The third requirement is also met as the message length is below 59 bytes. The last requirement is met because the device is configured with the public key of the server.

Listing 1 displays the main Python code that the device is running when joining a network. The code from line 6 to line 18 is the code that has been added to the normal OTAA code. The only exception is line 20, which would not have the public key added. The functions that are being called in this code are using the standard LoRaWAN implementations with the exception of line 20. The function being called is changed so that it includes the public key in the join request.

5.4 Challenges faced during the development phase

During the development and implementation of the protocol, multiple challenges were encountered. This section goes through them by analyzing the problem and solution used. Seven significant challenges were encountered. Two were related to the hardware used, while two were related to the communication. Two of the challenges were related to implementation. The final challenge was related to the authentication between a server and a device. The challenges are presented below based on their categorization. Each challenge is presented with a description of the problem and the solution or mitigation used.

5.4.1 Hardware related challenges

The first challenge that was encountered was the selection of what LoRa module to use. There are multiple vendors to choose from, and a module is using a specific frequency to transmit on. Selecting the right vendor was done by choosing the one that had good integration with the devices that would be used. The frequency that was selected was based on the regulations and laws in Sweden.

One challenge was that the data logger would take around ten samples per second.

```
def compress(publicKey):
    return publicKey.x.to_bytes(32,byteorder='big') +
        (publicKey.y % 2).to_bytes(1,byteorder='big')

def join():
    curve = registry.get_curve('brainpoolP256r1')
    privateKey = secrets.randbelow(curve.field.n)
    publicKey = privateKey * curve.g

    sharedKey = privateKey * server_public_key
    hash = hashlib.blake2s()
    hash.update(compress(sharedKey))
    hash_digest = hash.digest()
    appkey = bytearray(hash_digest[0:16])
    nwkkey = bytearray(hash_digest[16:32])
    lora.set_keys(appkey,nwkkey)

    lora.join_network_ec(join_eui,dev_eui,dev_nounce,compress(publicKey))
    (packet,length) = lora.listen()

    cipher = AES.new(bytes(nwkkey),AES.MODE_CTR)
    plaintext = cipher.encrypt(bytes(packet[1:33]))[:-4]

    join_nounce = plaintext[0:3]
    devaddr = plaintext[6:10]
    dev_nounce_b = dev_nounce.to_bytes(2,byteorder='big')
    lora.set_dev_addr(devaddr[::-1])

    (ask,fnk,snk,nek) = lora.setup_otaa_keys(join_nounce,dev_nounce_b)
```

Listing 1: The main Python code that is run on the device when it is attempting to join the LoRaWAN network

This sample rate would not always be able to detect sudden spikes in the current usage. The solution would be to do tests multiple times and to calculate the mean. The final solution was to reconfigure the data logger to make one sample every 12.6 milliseconds.

5.4.2 Communication related challenges

In the beginning of this thesis work, the idea was to have the device communicate with a self-hosted developer LoRaWAN infrastructure. However, when trying to perform the OTAA join procedure multiple times, Join Accept messages were not being sent back to the device. The infrastructure used a message scheduler to avoid traffic congestion. This is an excellent feature to have and is required when using it in the real world. A decision was made to have the gateway instead perform the

Join Procedure with the device. Another solution would be to modify the developer infrastructure and remove the scheduler.

Performing everything on the gateway presented another challenge. Some of the Join Accept messages sent to the device were not being received. The problem was because of the simplicity of the gateway solution. It was able to respond almost instantaneously to a Join Request. The speed caused problems as the device did not have the time to switch the LoRa radio chip into listening mode. The solution was to introduce a static delay on the gateway, making it wait for a second extra before responding.

5.4.3 Implementation related challenges

The main challenge is to design the protocol and keep everything within the message limit. Because the Join Request already uses 23 bytes, there are only 41 bytes that can be used. One solution is to split the content into multiple messages. However, that introduces another complexity of receiving some form of acknowledgment. The acknowledgment is needed as a message can be lost during the transmission over the air. Trying to conform to the size limit is a balance between security and battery life. Sending more messages would also cost more energy, and handling lost messages and acknowledgments would add further energy usage.

When writing the code, it was necessary to structure it to execute the same part multiple times efficiently. A challenge is also to ensure that the same work is being done each time. The code is based on the ABP example that was made by Adafruit, the maker of the radio module used[1]. That code was then modified also to support OTAA.

5.4.4 Authentication related challenges

One challenge is how to handle device authentication. Previously a device had the root key statically assigned through a trusted and secure channel. The same assignment would be done on the JS. The JS could then trust the identity of the device trying to register on the network. With the current implementation, this is not possible. The device can trust that it is communicating with the JS but not the other way around. The device is supplied with a DevEUI, which is an eight-byte number (64 bits) and acts as an identification number for a device. However, that number is broadcast over an insecure channel and could easily be spoofed. The problem is that a malicious user can register any number of devices and cause a Denial of Service attack.

PK-OTAA, in its current state, does not solve this issue; however, possible solutions could be added to the protocol to amend this. The server could issue a separate public key for each device owner. The main issue with this approach is that this owner specific public key from the server would need to be a secret only known to

[1] `https://github.com/adafruit/Adafruit_CircuitPython_RFM9x`

the server and the owner. It would not be a true public key because everyone is not allowed to know it. Everyone that knows the owner specific public key would then be able to register devices as that owner. The solution would then add further complexity and could potentially cause more issues.

6

Results

This chapter shows the data obtained from the data logger. The purpose is to provide information to answer the questions in Table 4.1. The goal was to create a new method to establish the root keys that a device will use to derive the session keys. The method had to conform to the LoRaWAN specification, not exceed a message length of 64 bytes, and not significantly increase the power consumption. The chapter begins by looking at the collected data of the power consumption. It ends with summarizing the basis for answering the questions.

6.1 Power consumption

This section examines the data on the power consumption when the device is running the OTAA method and the proposed method. The section starts by showing the consumption of OTAA. The section concludes with the consumption of PK-OTAA and comparing it with OTAA.

Figure 6.1 shows the power (—) and energy (—) consumption when the regular OTAA protocol is sending messages and listening for a response. The spike in power is caused by the radio transmitting a message. The drop in power right before sending is caused by the radio going into standby mode in order for it to load a message into its buffers. When the radio is in standby mode, it shuts off the hardware for transmitting and receiving. Because of that, the radio uses less power. This is useful as it is easier to visually see when the code enters a different stage of execution.

Figure 6.2 shows the power usage for OTAA and PK-OTAA. Red dashed vertical lines are added to help distinguish different parts of the program execution. The lines divide the data into four parts for OTAA and five parts for PK-OTAA. The first part is only executed for PK-OTAA and is the part where the device is generating the ECC key pairs and deriving the root keys. The second part executes code to prepare the radio for sending a message. The third part is when the radio is transmitting. The fourth part is when the radio is listening for a response. The last part is for when a message is being received.

Looking at the differences in power draw over time when sending a Join Request, it is

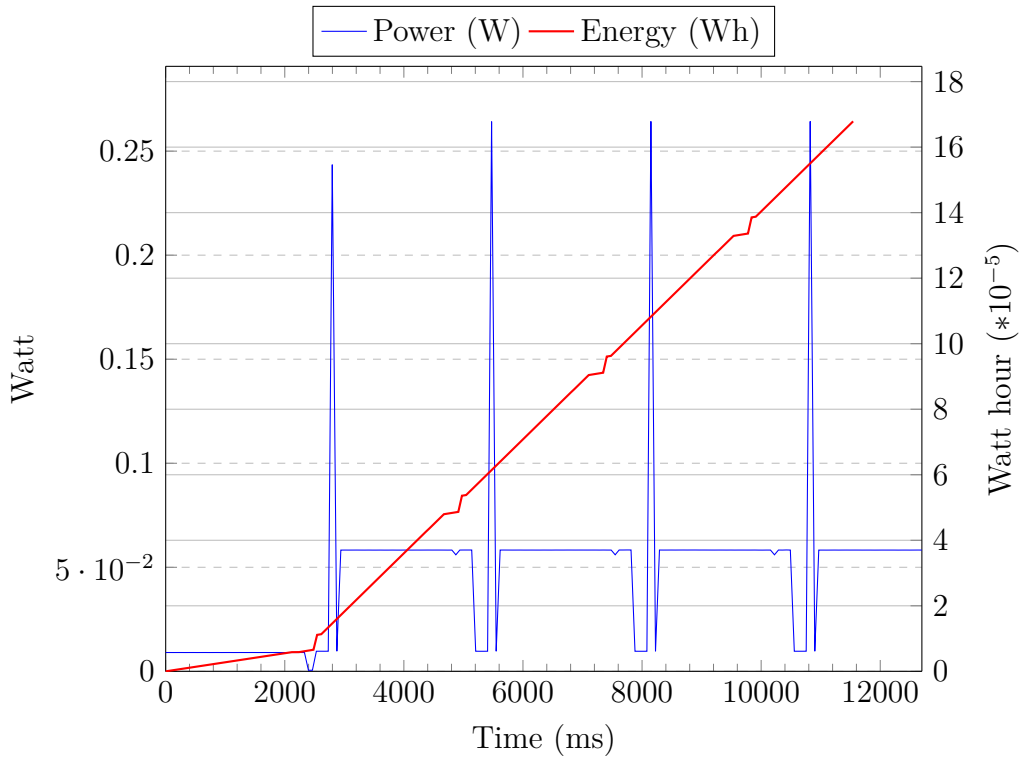

Figure 6.1: Power and energy consumption of the radio when sending four OTAA Join Requests

clear what is causing the increase in power usage. The added key generation process almost doubles the time taken to send a message. As can be seen in Table 6.1, OTAA takes 2,675 ms, and PK-OTAA takes 4,694 ms to finish. This is an increase of about 1.75 times. The battery drain of running the OTAA protocol is 0.1202 mAh, while the PK-OTAA would be 0.2132 mAh where the root key derivation drains 0.0852 mAh. The increase in mAh is about 1.77 times when using PK-OTAA.

If two AA batteries powered the device, the total battery capacity would be 2,000 mAh. OTAA would then be able to send around 17,000 Join Requests before the battery is drained. PK-OTAA would be able to send around 9,380 Join Requests before draining the battery. These estimates are based on the assumption that nothing else is draining the battery.

Method	Join Procedure Time	Join Procedure Battery
OTAA	2675 ms	0.1202 mAh
PK-OTAA	4694 ms	0.2132 mAh

Table 6.1: Overview of the time and battery usage of the two methods

There are two other things to consider in order to understand the cost of the join procedure in terms of battery life. The two things are the task of the device and the cost of sending a packet. The task of the device will be dependent on what purpose the device has. The cost of sending a packet can be measured and estimated.

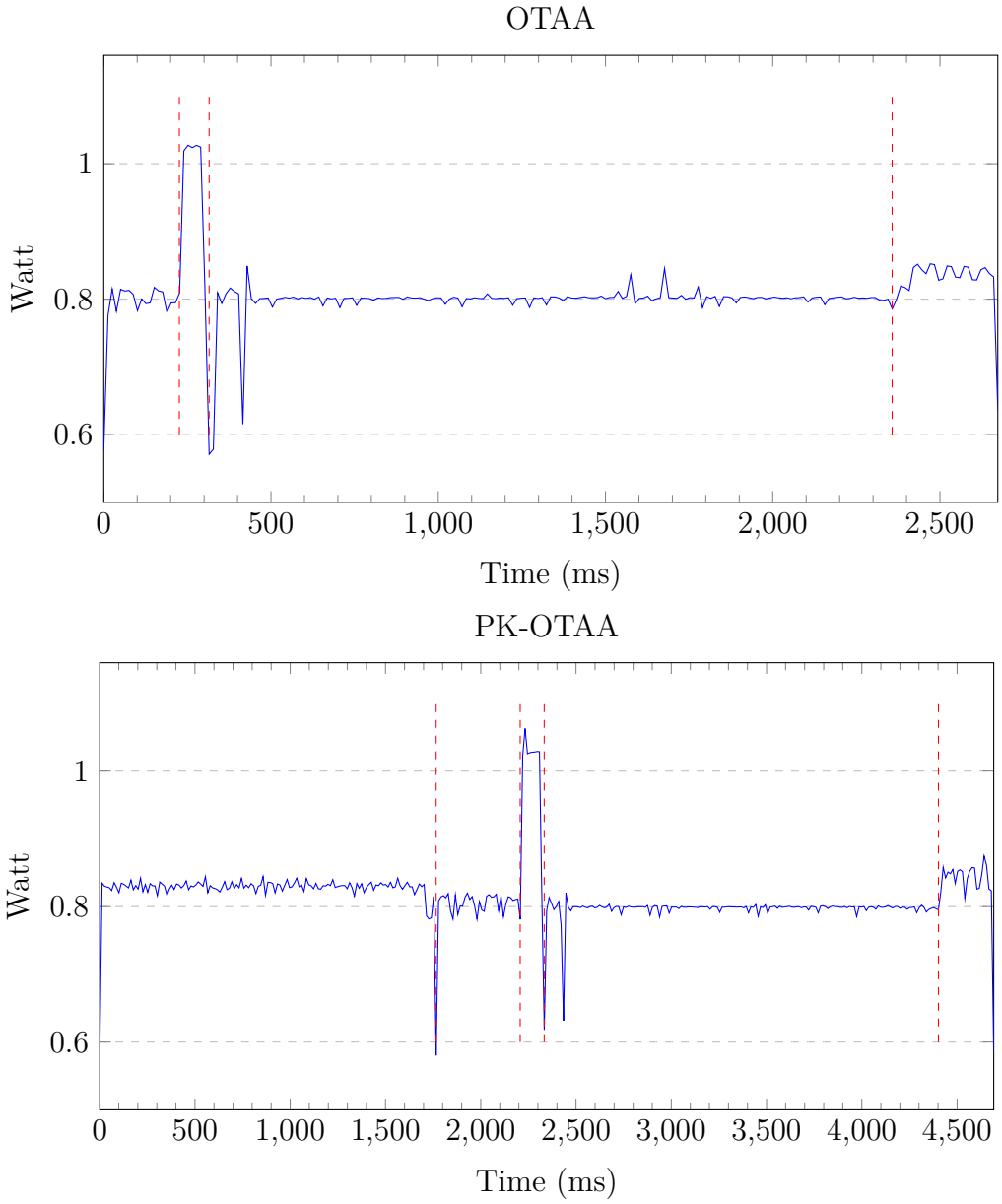

Figure 6.2: Power measurements of one message for OTAA and PK-OTAA

Figure 6.3 presents power measurements when sending three data packets. Sending a packet in OTAA or PK-OTAA is the same as they only handle the procedure for acquiring the session keys. The session keys are used to encrypt and integrity check packets, but those functions are the same for the two methods.

Table 6.2 presents the cost of sending one data packet. The message content for the packet was two bytes, and the total packet length was 14 bytes. Completing one OTAA join procedure would cost the same as sending approximately eight data packets. Using PK-OTAA, it would instead be approximately 13 packets. Considering the case of having two AA batteries for power and only sending data packets, it would be possible to send 125,680 packets.

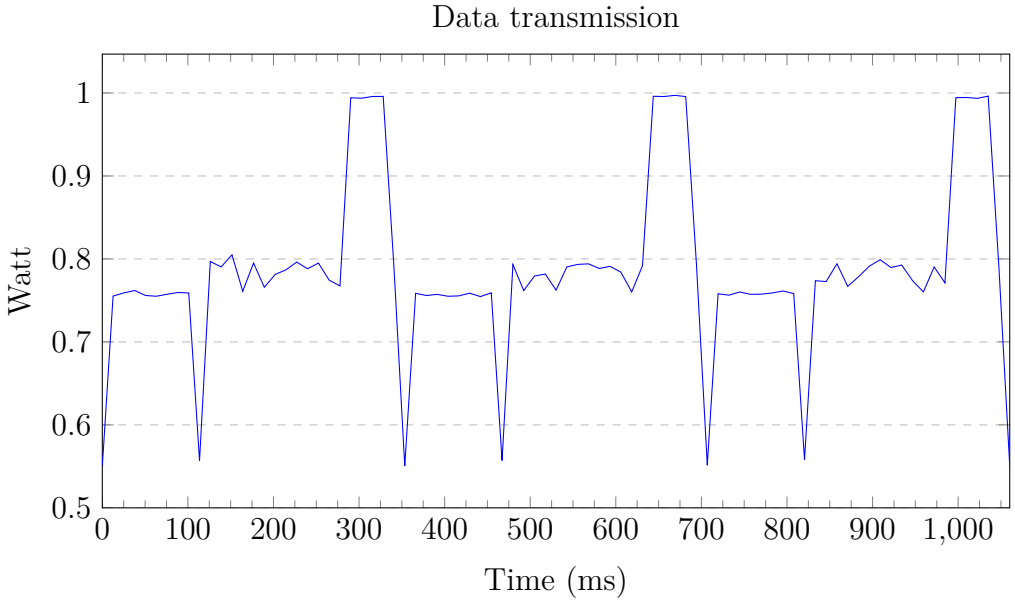

Figure 6.3: Power measurements of three consecutive data messages

Data Send Time	Data Send Battery
353.4 ms	0.016 mAh

Table 6.2: Time and battery measurements for sending one data packet

Table 6.3 presents an overview of the execution times for the different segments. PK-OTAA have the included key generation segment. The segments, listen and received, along with their sum are included to show that there is no change in handling the response. The time for preparing and sending a packet are the important ones. The code for those segments are essentially unchanged. The main difference between the two methods in the segments are the packet length, 23 versus 56 bytes (refer to Table 5.6), which is around 2.4 times bigger. The table shows that the prepare segment takes 95% longer to finish and the send segment 42%. Table 6.4 presents the usage in mAh instead of time. Here the change is almost identical to the changes in time. Figure 6.4 visualizes the data from Table 6.4 as a bar graph.

Segment	Time OTAA (ms)	Time PK-OTAA (ms)	Change
Key Generation	N/A	1766	N/A
Prepare Packet	226	441	95%
Send Packet	89	126	42%
Listen for Response	2042	2069	1.3%
Process Response	318	292	-8.2%
Listen & Process	2360	2361	0.04%

Table 6.3: The execution time in milliseconds for each segment

Table 6.5 shows another interesting observation. It shows how many milliseconds that is used per byte in a message for the preparing and sending segments. The data

Segment	Battery OTAA	Battery PK-OTAA	Change
Key Generation	N/A	0.0812 mAh	N/A
Prepare Packet	0.0099 mAh	0.0196 mAh	98%
Send Packet	0.0047 mAh	0.0069 mAh	46%
Listen for Response	0.0905 mAh	0.0915 mAh	1.1%
Process Response	0.0151 mAh	0.0140 mAh	-7.1%
Listen & Process	0.1056 mAh	0.1055 mAh	-0.07%

Table 6.4: The battery usage in mAh for each segment

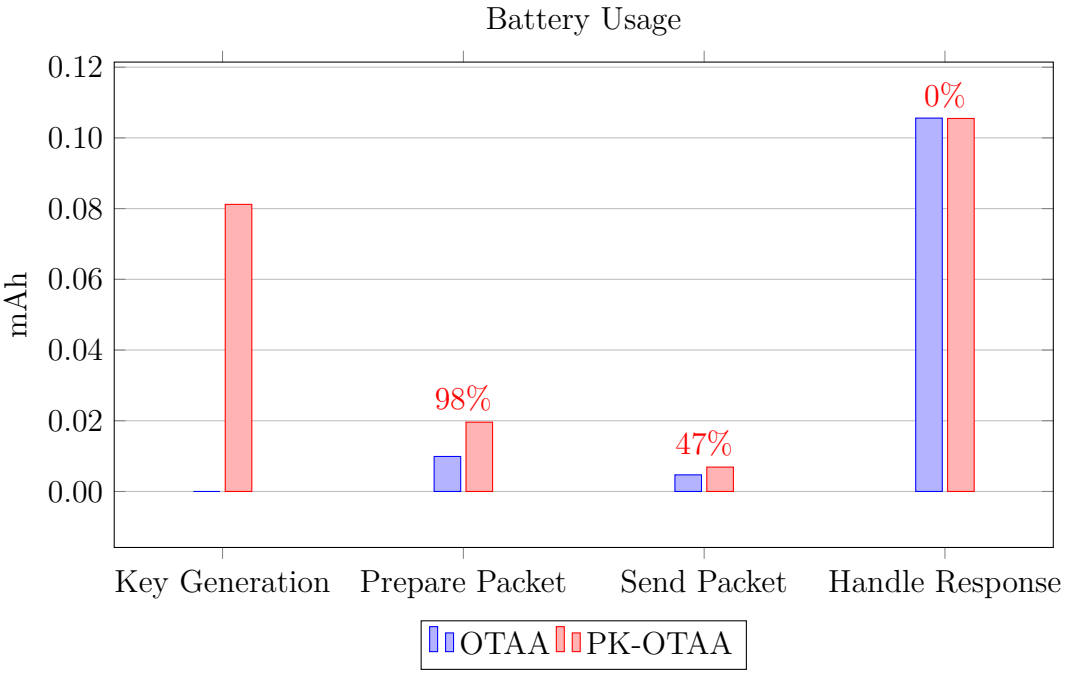

Figure 6.4: The battery usage of each segment along with the increase

shows a decrease in the time used per byte when executing the segments. Table 6.6 presents similar data when examining the battery usage. This observation will be further discussed in Section 7.1.

Segment	OTAA (ms/byte)	PK-OTAA (ms/byte)	Change
Prepare Packet	9.8	7.9	-20%
Send Packet	3.9	2.3	-42%

Table 6.5: The execution time in milliseconds per byte of a message

Segment	OTAA (mAh/byte)	PK-OTAA (mAh/byte)	Change
Prepare Packet	0.000431	0.00035	-19%
Send Packet	0.000206	0.000123	-40%

Table 6.6: The battery usage (mAh) per byte of a message

6.2 Responses to the feasibility conditions

This section provides the evidence to support the answer to the questions stated in Section 4.1. It introduces each question and provides a motivated answer to it.

F1: Does the method fit in the LoRaWAN specification? The implemented protocol introduces two new things for a device or server to implement. It introduces ECC calculations. It also requires the *BLAKE2s* hash algorithm. The rest of the protocol uses the standard OTAA method. It is possible to replace the hash algorithm with AES. The session keys are derived using that method.

F2: Are all messages the method sends less than or equal to 64 bytes? A standard OTAA Join Request requires 23 bytes in total with 41 bytes to spare (see Section 2.3.6). PK-OTAA only adds the ECC public key to that message. The maximum size of the public key then has to be 41 bytes. To get comparable security to what is currently used by LoRaWAN, the key size should be 256-bit (32 bytes). That size makes it comparable to using AES with a 128-bit key. The public key in ECC is twice the size of the private key but can be compressed. The size of the public key, in this case, is then 33 bytes. This size leaves eight bytes to be used for something else, or it could be used in the future for increasing the key size. The Join Accept message remains unchanged, and the protocol only modifies the Join Request message. All messages are thus within the limit.

F3: Is the increase in energy consumption not significant? This answer depends on how many times the root keys are going to be updated. Every time the join procedure is executed the energy for at least 13 possible data packets would be consumed. It is possible to send approximately 125,680 packets using two AA batteries. This would then be 0.01% of the total battery life of the device. Running the procedure every day for a year would consume around 3.7% of the total battery life.

7

Discussion

This chapter discusses and examines different parts of this thesis, where the first section starts with an in-depth analysis of the results. The second section has a few comments about the implementation. The third section contains comments on the chosen LoRaWAN infrastructure setup. In the fourth section, the setup for taking energy measurements is discussed. The fifth section examines this thesis from a ethics and sustainability perspective. The final section discusses the possible future work that can be performed.

7.1 Result evaluation

This section evaluates and discusses the result of the data gathered. It starts by giving some more insight into the result presented. It also discusses what could affect the result positively or negatively.

The original hypothesis was that the proposed method should be feasible. The primary condition that was dictating the feasibility would be power consumption. It was known that it would increase as the proposed method will add more computation to the join procedure, but it was unknown by how much. The result was that the new method was using a lot more than the original OTAA method, about 77% more. However, considering that the activation method is not executed often compared to the number of data packets that a device will send, this increase is negligible. With the result presented, an example device is capable of sending around 125,680 data packets. PK-OTAA costs almost the same as 13 data packets. The actual cost of PK-OTAA is then 0.01% of the total battery life. The increase of PK-OTAA compared to OTAA is significant, but it is insignificant when comparing it to the lifetime of the device.

Although the results show that the proposed method consumes 0.2132 mAh of battery, it is important to note that this value is produced using non-optimized hardware and code. The actual usage could be lower when using hardware that is made and optimized for the specific algorithms that the protocol is using. There are two improvements to be made with the code. The code is developed using Python, and there can be an improvement in performance if the code is implemented in C or some other language made for embedded systems. The second improvement is to

optimize the code as the code's actual performance was not considered when developing it. With the improvements and optimizations made to the code, the results could be different. Since the improvements would be made to both methods, it is also possible that the result will not have a drastic change.

The most significant energy consumption for the PK-OTAA protocol is the added code segment generating the root keys. The results show that the segment takes 1,766 ms to complete and consumes 0.0812 mAh of battery. PK-OTAA uses 4,694 milliseconds and 0.2132 mAh to run the procedure once, where the key generation segment uses 38% of the execution time and battery usage of the entire procedure. The listen segment is using 44% of the time and 43% of the battery. When improving the battery consumption, the focus should be on the key generation segment because the impact will be considerable. Although optimizing the listening segment would also have a considerable impact on the segment it is the same for OTAA and PK-OTAA.

One optimization for the key generation segment is to pre-compute multiple key pairs for the device. As mentioned in the previous paragraph, the key generation uses around 38% of the time and battery. However, there could be an issue with the storage that this technique needs. When using 256-bit keys, the private key would be 32 bytes, and the uncompressed public key is 64 bytes. If the device is loaded with ten key pairs, it will use 960 bytes of storage space. Depending on the hardware used, this could be a significant amount of memory as data and code share the same memory. An example could be the ATmega32U4[1] microcontroller, which has 32kB of storage. Depending on how big the program is using almost one kB of data can work. However, if the hardware has even less available memory, storing the pre-computed keys in memory might not work. There is also the possibility of introducing vulnerabilities to the protocol, since computing the keys might be executed on a different computer. If that computer does not dispose of the keys correctly or decide to keep them, it could cause the keys to be leaked. There is also the argument to be made that the same can be done for OTAA. There could be multiple root keys registered with the JS, and with the addition of a key update MAC command, the same goal of renewing root keys can be achieved. The main drawback of that method is that the number of times that the root keys can be updated is dependent on the number of root keys registered.

The result section also presents an interesting observation in Table 6.5 and Table 6.6. The tables show that the time and battery per byte is decreased when preparing a packet and sending it. The decrease is perhaps not surprising considering that the code consists of one part that is dependent on how long the message is and one part that is not. The part that is dependent on the message length will increase in cost as the length grows. The static part will stay the same regardless of the message length. The increase or decrease in the cost per byte is dependent on the time complexity of the code. Having a linear time complexity would provide a decrease while a exponential would eventually be increasing the cost per byte. The total

[1] `https://www.microchip.com/wwwproducts/en/ATmega32U4`

cost can be expressed with the following linear function $y = kx + m$ where y is the cost, x is the number of bytes, and m is the cost of the static part. Cost per byte would then be $\frac{y}{x} = k\frac{m}{x}$, and having a linear function would mean that the static part is having less of an effect on the cost per byte. If it was an exponential function $y = ax^2 + bx + m$ the cost per byte would be $\frac{y}{x} = ax + b + \frac{m}{x}$. Using the exponential function the cost per byte would be dependent on the number of bytes. The prepare and send segments of the code was never analyzed and given a complexity function it is therefore difficult to know if the cost per byte would increase when having longer messages. It could be analyzed and examined further to find optimization opportunities. However, that was not the goal of this thesis and it could be included in a future work instead.

During the testing, the radio chip was put to sleep after each message being sent or received. The radio chip was asleep for around ten milliseconds each time. Adding the extra sleep time was done to capture the spikes that can be be seen in the data. The spikes made it easier to distinguish the different segments from each other. These delays add extra time and power usage, but because they were used for both methods, it will not change the outcome of the result shown.

It can be argued that the Listen and Process segments of the code can be excluded as they are the same for both protocols. The code is identical for the two methods, and the results also show that this is the case in terms of execution time and battery usage. However, the segments are needed to provide data and insight into the entire procedure. The Listen and Process segments take up around 50% of the time and battery used by the procedure.

Examining the data from the preparation and sending segments, an interesting observation is found. There is a clear difference between the increase in time and battery compared to the increase in the message length. The join request length in PK-OTAA is 53 bytes, which is an increase of around 2.4 times compared to OTAA. Meanwhile, the increase in time is 95% and 42% for preparation and sending, respectively. The increase in battery usage is 98% and 46%, respectively. Examining the difference between the time and battery usage per byte shows the cost of increasing the message length.

The implemented protocol is meant to show that it is feasible to use PKC in the activation method of LoRaWAN devices. Therefore some of the needed verification checks have been excluded. These checks include the device checking the validity of the MIC on the join accept message from the JS. The JS does not check the MIC or that the public key of the device is valid. Since the MIC checks are excluded from both OTAA and PK-OTAA, their absence will not affect the results shown. It will not have a significant impact on the result as calculations to verify the MIC is essentially the same for both methods. The checks that the JS is supposed to perform does not matter either. This is because the server's performance does not affect the device during testing.

The answers to the feasibility conditions presented in Section 6.2 shows that PK-

OTAA is considered feasible. The conditions are: does the method fit in the LoRaWAN specification, do all messages fit in 64 bytes, and is the increase in energy consumption not significant. The protocol, as it is currently designed, satisfies the stated conditions.

7.2 Implementation

This section covers some of the issues that the proposed protocol is currently facing and what can be done about them. There are three issues, authentication, the LoRaWAN message size limit, and the difficulty in introducing multiple messages.

The protocol does not have any secure authentication of devices. The device can verify if the identity of the JS is legitimate. However, the JS can not do the same to the device. The lack of authentication allows a malicious user to create any number of devices. This problem could potentially cause the JS to be flooded with Join Requests and block other devices from using it. One solution can be to introduce a digital signature that is added to the Join Request. The device owner would then sign the identity of the device, and the JS can verify the ownership of a device. However, adding a digital signature will use more bytes, and it will probably not fit within the size limit. Another factor to consider is that verifying the ownership will not stop a malicious user from trying to flood the communication channel with Join Requests. The unverified messages can be dropped to prevent hardware resources from being exhausted. Although the physical hardware can be capable of handling malicious traffic, disruptions can happen over the radio communication channel. One important factor to also consider here is that even when using OTAA, a user can send multiple Join Requests and cause a disruption in communication.

The authentication issue could also be solved by registering the device on the JS. The device could get some unique key or value to use in verifying its identity. This value would then serve a similar purpose of authentication as the root keys in OTAA. One of the underlying ideas in this thesis is to avoid device registration. Avoiding it could make the distribution of devices simpler by not requiring the registration step for each device. The device is then only configured with the public key of the JS, which is just a static key.

The major limitation when using LoRaWAN is the message size limit. The maximum limit is a bit over 200 bytes, but that is only used in conditions with few obstructions and high signal strength. A simple solution to bypass the message size limit is to divide the content into multiple messages. It will then be possible to send a large amount of data. However, LoRaWAN is supposed to have a low bit-rate, and it is not meant to be used to send a large amount of data. Multiple messages would also potentially flood the radio communication channel, causing a disruption. The disruption can cause messages to be lost. The common way to deal with message loss is by the recipient sending an acknowledgment that the message is received. Sending acknowledgments would then further increase the messages and can cause further traffic congestion. More messages would also cause the device to use more

power to send and listen for the acknowledgments, further diminishing the battery life. LoRaWAN can support this type of solution as two of the message types are *Confirmed Data Up/Down*. Although the support for it is there, it might cause more issues than it solves, and the loss in battery life might not be worth it.

7.3 LoRaWAN infrastructure

This thesis intended at first to use a full LoRaWAN infrastructure to test the two activation methods. However, results from the initial testing with the infrastructure showed that the testing could be easier with simpler infrastructure. The simple infrastructure uses fewer software components, and the entire join procedure is easier to monitor and control. This simple infrastructure was implemented on the gateway. The task of the gateway is then to receive a message, handle it, and send a response back. Handling a message was also made simple as the gateway only needs to check if the message is a valid message of any kind. The gateway will always send the same message as a response, which is a join accept message. PK-OTAA and OTAA only need to receive a join accept message in order to complete the rest of the join procedure, and the gateway can easily fulfill that need.

7.4 Energy measurements

The data for the battery usage was collected using a power supply that has data logging capabilities. The power supply can sample the voltage and current at a precise interval. The specific interval used was 12.6 ms, as it should capture most power spikes and still provide good accuracy for the measurements. The method to measure the current and voltage used in this thesis is quite simple. The method used only measures the power usage of every component in the device. A more advanced method would perhaps check the power usage of only the CPU or other more specific components. Conducting these types of measurements could provide data that better distinguish between the cost of certain algorithms or code executions.

7.5 Ethics and Sustainability

This thesis has investigated methods to improve the security of the LoRaWAN devices. These devices could be used as sensors in people's homes or as trackers of different things. Therefore, it is important that these devices are secure and that the possibility of malicious users getting control of the devices is as low as possible. This thesis has attempted to ensure that a third party can not intercept the activation process by making sure that a MITM attack is not feasible. However, this does not stop a malicious user from trying other attack vectors. It could get access to the JS and thus be able to read all data coming from the devices. All developers that implement LoRaWAN should ensure that their systems are secured and that the only vulnerability is being able to obtain physical access to the devices, but that should also only compromise that single device.

Researches within communication protocols should ensure that the Confidentiality, Integrity and Availability triad is upheld. With confidentiality, they should ensure that only the communicating parties can read the messages from the other, and no third party should be able to do that. However, that does not stop one of the parties in the communication to share this information knowingly or unknowingly to a third party. The message should be protected while traveling from the sender to the recipient. Concerning integrity, no third party should be able to alter the communication without either of the communicating parties knowing. Availability is harder to ensure as disruptions in communication are often not something that can be controlled by any single entity.

The work produced by this thesis can not be considered to enhance the sustainability of IoT devices. The produced protocol can make it easier and faster to deploy devices into the world, which would then cause more e-waste. Although this will probably be inevitable, as shown by Ericsson, there will be billions of these IoT devices. We, as researchers and developers, should ensure that the energy usage of the devices is kept as low as possible. This change can increase the lifetime of the devices, and the effect of that would be fewer devices that need replacing, hopefully reducing e-waste. There is also the argument that the IoT devices could be used to improve other aspects of sustainability. Although they produce some e-waste, their use could reduce energy consumption or, through digitalization, make certain tasks more efficient. One example could be to place a sensor on trash cans to have them send a notification when they are full. This thesis is a unique work that explores the increase in energy consumption when evaluating the feasibility of the protocol. To lessen the e-waste caused by our devices, we should always consider the solution's potential energy cost.

7.6 Future work

This section explains some of the work that can be done to improve the PK-OTAA method or the LoRaWAN specification. The section presents six potential future work. The first work is regarding optimizing PK-OTAA. The second work is about dynamically splitting large messages into multiple parts. Further examining the security of PK-OTAA is the third work. The fourth work is evaluating the possibility of using other key agreement protocols. The fifth work is a more in-depth comparison between PK-OTAA and other related work. The last work is regarding the idea to pre-compute the key pairs used in PK-OTAA.

7.6.1 Testing and Optimization of PK-OTAA

This thesis shows with PK-OTAA that it is feasible to use PKC in the activation of LoRaWAN devices. There is, however, more potential with the protocol, and it can be further improved. The code will need to be optimized, and it will need to be tested on purpose-built hardware. The protocol would also need to be tested in a non-lab environment. This would help show the usefulness of the protocol in a real-world scenario.

7.6.2 Dynamic message length

The message limit imposed by the LoRaWAN specification will cause issues in the future if the need for larger keys is desired. It will also cause issues if the PK-OTAA protocol is extended and needs to contain more information in the message. A possible work is, therefore, to design the protocol with the ability to handle splitting a message into multiple parts. This work would also need to include power measurements to evaluate the expected increase in battery consumption. The feature could also be dynamic as the data rate the device uses could be higher and thus allow for larger messages and the need to adapt accordingly.

7.6.3 Evaluating the security of PK-OTAA

During the design phase of PK-OTAA, the focus was not on the security of the protocol. The focus was to evaluate the feasibility of using PKC in device activation. The protocol uses ECDH to exchange a secret, and if ECDH is secure, then so is the protocol. A static and ephemeral key is used in the protocol, and this should thwart any attempts to execute an MITM attack. However, there could be other attack vectors to the protocol. There are multiple areas to inspect further and improve with the protocol. The key generation process will need to be examined as the keys may not be generated randomly. The public key of the device is sent in clear text over an insecure channel that could be exploited. A second area is the usage of the correct elliptic curve and implementation of ECDH on the device. A possibility exists that an adversary can create a lot of public keys, create multiple secrets with the JS, and deduce the private key of the JS.

An area that can be improved with regards to both security and efficiency is the root key derivation or Key Derivation Function (KDF) used in the protocol. It is currently a simple hash of the secret created with ECDH, where the hash is split in two becoming the two root keys. The security of this method can be assessed and be improved. The hash function used is called *BLAKE2s*, and the authors have claimed that it is efficient for 8 to 32-bit processors [48]. This function could be evaluated and compared to other solutions. It is also possible that different hardware will be capable of different efficient calculations, and PK-OTAA can support multiple KDF methods.

7.6.4 Key agreement protocols

PK-OTAA uses ECDH to agree on a key, but it is possible to exchange the key with other similar key agreement protocols. The protocol is shown to be feasible while using ECDH, but it is not shown to be optimal. Future work could, therefore, be to examine multiple key agreement protocols and their applicability to LoRaWAN.

7.6.5 Comparison to similar work

Xing *et al.* [32] created a similar protocol to PK-OTAA. The authors also decided to use ECDH for better performance and shorter keys. This thesis acknowledges

this work but does not make any direct comparisons to it. It is not easy to make any comparisons as the work from Xing *et al.* does not present any data regarding the potential energy consumption compared to OTAA. The main similarity to this thesis is the decision to use ECDH to agree on a value and use that value in the root key generation. The protocol from Xing *et al.* uses six messages, while PK-OTAA needs one to establish root keys with the JS. Their protocol also breaks the message size limit, while PK-OTAA can be used with any data rate. Thus, their protocol is not considered to meet the first two feasibility conditions specified by this thesis, i.e., adhering to the specification and the message size limit. The last feasibility condition regarding energy consumption could be examined further and compared to PK-OTAA.

7.6.6 Pregenerating key pairs

Section 7.1 discussed the possibility to pre-generate the elliptic curve key pair used in PK-OTAA. Investigating the feasibility of pre-generation could be a promising future work with a high impact. Although there are some issues and potential complications with such an implementation, it is worth investigating the potential further, since pre-generating the keys would drastically cut down on the battery usage of PK-OTAA.

8

Conclusion

The purpose of this thesis was to investigate the feasibility of using PKC to generate and delegate root keys to devices in LoRaWAN. This thesis has introduced and thoroughly evaluated a new method of activating LoRaWAN devices using PKC. The method is called Public Key Over The Air Activation (PK-OTAA) and is using ECDH to exchange a secret that is used to derive the root keys. PK-OTAA is considered feasible according to the criteria specified in this thesis. This thesis compares PK-OTAA to the original OTAA specified in the LoRaWAN specification. The primary evaluation comes from comparing the battery usage of the two methods. Considering the battery usage makes this thesis unique among other similar works as the impact on the battery life of devices is often not considered.

The results presented in this thesis show that PK-OTAA takes 75% longer to execute and consumes 77% more battery life compared to OTAA. Although the increase compared to OTAA appears to be significant, it is insignificant when comparing it to the possible data messages. This thesis presents an example of a device that is capable of sending 125,680 data packets. Completing a join procedure with OTAA would be the same as sending eight data packets in terms of battery consumption, while for PK-OTAA, it is 13. Doing one PK-OTAA join procedure would then consume 0.01% of the total number of data packets. The PK-OTAA join procedure would not be used very often during the entire lifetime of the device. Deciding how often to change or update the root keys is up to each user, but changing the root keys often is a must for improved security, considering that they can be compromised due to user fault or error. However, most probably, the procedure would be executed once every year. If a device has a lifetime of ten years, the increase in battery consumption would be quite negligible compared to the total lifetime of a device. PK-OTAA improves the overall security of LoRaWAN by allowing the renewal of the root keys for the devices. It also achieves this with a negligible increase in battery consumption. This thesis shows that PK-OTAA is feasible, but there is much more work that can be done to improve the method.

Bibliography

[1] Telefonaktiebolaget LM Ericsson, "IoT connections outlook," last accessed on 2020-06-10. [Online]. Available: https://www.ericsson.com/en/mobility-report/reports/november-2019/iot-connections-outlook

[2] LoRa Alliance, "LoRa Alliance," last accessed on 2020-06-10. [Online]. Available: https://lora-alliance.org/

[3] E. H. Spafford, "The internet worm program: an analysis," *ACM SIGCOMM Computer Communication Review*, vol. 19, no. 1, pp. 17–57, 1989.

[4] "EU general data protection regulation." [Online]. Available: https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32016R0679

[5] W. Stallings, *Cryptography and network security : principles and practice.* Pearson, 2017.

[6] LoRa Alliance, "LoRaWAN Security: A white paper prepared for the LoRa Alliance - FUll end-to-end encryption for IoT application providers," last accessed on 2019-12-13. [Online]. Available: https://lora-alliance.org/sites/default/files/2019-05/lorawan_security_whitepaper.pdf

[7] LoRa Alliance Technical Committee, *LoRaWAN 1.1 Specification*, 2017. [Online]. Available: https://lora-alliance.org/sites/default/files/2018-04/lorawantm_specification_-v1.1.pdf

[8] I. Butun, N. Pereira, and M. Gidlund, "Security Risk Analysis of LoRaWAN and Future Directions," *Future Internet*, vol. 11, no. 1, 2018. [Online]. Available: https://www.mdpi.com/1999-5903/11/1/3

[9] D. P. Mahajan and A. Sachdeva, "A Study of Encryption Algorithms AES, DES and RSA for Security," *Global Journal of Computer Science and Technology*, 2013. [Online]. Available: https://computerresearch.org/index.php/computer/article/view/272

[10] S. G. Krantz, *The Proof is in the Pudding. [electronic resource] : The Changing Nature of Mathematical Proof.* Springer New York, 2011.

[11] P. W. Shor, "Algorithms for quantum computation: discrete logarithms and factoring," in *Proceedings 35th Annual Symposium on Foundations of Computer Science*, 1994, pp. 124–134.

[12] H. Abelson, R. Anderson, S. M. Bellovin, J. Benaloh, M. Blaze, W. Diffie, J. Gilmore, P. G. Neumann, R. L. Rivest, J. I. Schiller, B. Schneier, J. Benalob, M. Blaze, W. Diffie, J. Gilmore, P. G. Neumann, R. L. Rivest, J. I. Schiller, and B. Schneier, "The Risks of Key Recovery, Key Escrow, and Trusted Third-Party Encryption," *World Wide Web Journal*, vol. 2, no. 3, pp. 241–257, 1997. [Online]. Available: http://dl.acm.org/citation.cfm?id=275079.275104

[13] J. Daemen and V. Rijmen, *The design of Rijndael: AES — the Advanced Encryption Standard*. Springer-Verlag Berlin Heidelberg, 2002, ISBN: 978-3-662-04722-4. [Online]. Available: https://doi.org/10.1007/978-3-662-04722-4

[14] W. Diffie and M. Hellman, "New directions in cryptography," *IEEE Transactions on Information Theory*, vol. 22, no. 6, pp. 644–654, November 1976.

[15] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Commun. ACM*, vol. 26, no. 1, p. 96–99, Jan. 1983. [Online]. Available: https://doi.org/10.1145/357980.358017

[16] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM Review*, vol. 41, no. 2, pp. 303–332, 1999. [Online]. Available: https://doi.org/10.1137/S0036144598347011

[17] C. F. Kerry and P. D. Gallagher, "Digital Signature Standard (DSS) CATEGORY: COMPUTER SECURITY SUBCATEGORY: CRYPTOGRAPHY," 2013. [Online]. Available: http://dx.doi.org/10.6028/NIST.FIPS.186-4

[18] N. Lawson, "DSA requirements for random k value | rdist," November 2010, last accessed on 2020-02-10. [Online]. Available: https://rdist.root.org/2010/11/19/dsa-requirements-for-random-k-value/

[19] J. López and R. Dahab, "An overview of elliptic curve cryptography," Tech. Rep., 2000. [Online]. Available: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.37.2771&rep=rep1&type=pdf

[20] V. Gayoso Martínez, L. Hernandez Encinas, and C. Sánchez Ávila, "A survey of the elliptic curve integrated encryption scheme," *Journal of Computer Science and Engineering*, vol. 2, pp. 7–13, 01 2010.

[21] A. Shamir, "Identity-Based Cryptosystems and Signature Schemes," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 196 LNCS. Springer Verlag, 1985, pp. 47–53.

[22] S. S. Al-Riyami and K. G. Paterson, "Certificateless public key cryptography,"

Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 2894, pp. 452–473, nov 2003.

[23] V. Shoup, "A Proposal for an ISO Standard for Public Key Encryption (version 2.1) Summary of Changes from version 2.0 (," Tech. Rep., 2001, last accessed on 2020-03-12. [Online]. Available: https://www.shoup.net/papers/iso-2_1.pdf

[24] LoRa Alliance Technical Marketing Workgroup, "A technical overview of LoRa ® and LoRaWAN ™ What is it?" LoRa Alliance, Tech. Rep., 2015. [Online]. Available: https://lora-alliance.org/sites/default/files/2018-04/what-is-lorawan.pdf

[25] LoRa Alliance, *LoRaWAN 1.1 Regional Parameters*, 2017, last accessed on 2020-03-12. [Online]. Available: https://lora-alliance.org/sites/default/files/2018-05/lorawan-regional-parameters-v1.1ra.pdf

[26] N. P. Smart, M. Abdalla, E. Bjørstad, C. Cid, B. Gierlichs, A. Hülsing, A. Luykx, K. G. Paterson, B. Preneel, A.-R. Sadeghi, T. Spies, M. Stam, M. Ward, B. Warinschi, and G. Watson, "Algorithms, Key Size and Protocols Report (2018)," ECRYPT, Tech. Rep., 2018. [Online]. Available: https://www.ecrypt.eu.org/csa/documents/D5.4-FinalAlgKeySizeProt.pdf

[27] I. Butun, N. Pereira, and M. Gidlund, "Analysis of LoRaWAN v1.1 security." Association for Computing Machinery (ACM), 2018, pp. 1–6. [Online]. Available: https://dl.acm.org/doi/abs/10.1145/3213299.3213304

[28] E. S. Alashwali and K. Rasmussen, "What's in a downgrade? A taxonomy of downgrade attacks in the TLS protocol and application protocols using TLS," in *Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering, LNICST*, vol. 255. Springer Verlag, 2018, pp. 468–487. [Online]. Available: https://arxiv.org/pdf/1809.05681.pdf

[29] R. Sanchez-Iborra, J. Sánchez-Gómez, S. Pérez, P. J. Fernández, J. Santa, J. L. Hernández-Ramos, and A. F. Skarmeta, "Enhancing LoRaWAN security through a lightweight and authenticated key management approach," *Sensors (Switzerland)*, vol. 18, no. 6, jun 2018.

[30] S. Tiwari, H. B. Patel, and B. Shrimali, "A Survey on Certificate-Less Public Key Encryption for Authentication in a Smart IoT-Based LoRaWAN," Tech. Rep., 2018. [Online]. Available: www.iosrjen.org

[31] M. Ralambotiana, "Key management with a trusted third party using LoRaWAN protocol : A study case for E2E security," Tech. Rep., 2018. [Online]. Available: https://pdfs.semanticscholar.org/9cbf/ee330d03d2d0df32b4fb1a1412fd1dffed0d.pdf

[32] J. Xing, L. Hou, K. Zhang, and K. Zheng, "An Improved Secure Key

Management Scheme for LoRa System," in *2019 IEEE 19th International Conference on Communication Technology (ICCT)*. IEEE, oct 2019, pp. 296–301. [Online]. Available: https://ieeexplore.ieee.org/document/8947215/

[33] P. Y. Ting, J. L. Tsai, and T. S. Wu, "Signcryption method suitable for low-power IoT devices in a wireless sensor network," *IEEE Systems Journal*, vol. 12, no. 3, pp. 2385–2394, sep 2018.

[34] M. Leent, "An improved key distribution and updating mechanism for low power wide area networks (LPWAN) From the viewpoint of the nationwide LoRa network," Tech. Rep., 2017. [Online]. Available: https://openaccess.leidenuniv.nl/bitstream/handle/1887/64558/Leent_M_van_2017_CS.pdf

[35] J. Han and J. Wang, "An Enhanced Key Management Scheme for LoRaWAN," *Cryptography*, vol. 2, no. 4, p. 34, nov 2018. [Online]. Available: http://www.mdpi.com/2410-387X/2/4/34

[36] Z. Xia, H. Zhou, K. Gu, B. Yin, Y. Zeng, and M. Xu, "Secure session key management scheme for meter-reading system based on LoRa technology," *IEEE Access*, vol. 6, pp. 75 015–75 024, 2018.

[37] A. J. Bidgoly and H. J. Bidgoly, "A Novel Chaining Encryption Algorithm for LPWAN IoT Network," *IEEE Sensors Journal*, vol. 19, no. 16, pp. 7027–7034, aug 2019.

[38] K. L. Tsai, Y. L. Huang, F. Y. Leu, I. You, Y. L. Huang, and C. H. Tsai, "AES-128 based secure low power communication for LoRaWAN IoT environments," *IEEE Access*, vol. 6, pp. 45 325–45 334, jul 2018.

[39] I. Butun, "Prevention and detection of intrusions in wireless sensor networks," *University of South Florida, Ph.D. Dissertation*, 2013.

[40] "Buy a Raspberry Pi Zero – Raspberry Pi," last accessed on 2020-05-18. [Online]. Available: https://www.raspberrypi.org/products/raspberry-pi-zero/

[41] "Buy a Raspberry Pi 3 Model B – Raspberry Pi," last accessed on 2020-05-18. [Online]. Available: https://www.raspberrypi.org/products/raspberry-pi-3-model-b/

[42] *USB 3 . 1 Legacy Cable and Connector Specification*, Hewlett-Packard Inc, Intel Corporation, Microsoft Corporation, Renesas Corporation, STMicroelectronics, Texas Instruments, 9 2017, rev. 1.0.

[43] "Elliptic Curve Cryptography (ECC Certificates) | DigiCert.com," last accessed on 2020-04-16. [Online]. Available: https://www.digicert.com/ecc.htm

[44] J. Jonsson and B. Kaliski, "Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1." [Online]. Available:

https://tools.ietf.org/html/rfc3447

[45] X. Yao, X. Han, and X. Du, "A light-weight certificate-less public key cryptography scheme based on ECC," in *Proceedings - International Conference on Computer Communications and Networks, ICCCN*. Institute of Electrical and Electronics Engineers Inc., sep 2014.

[46] "eBACS: ECRYPT Benchmarking of Cryptographic Systems." [Online]. Available: https://bench.cr.yp.to/

[47] E. Barker, L. Chen, A. Roginsky, A. Vassilev, and R. Davis, "Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography," US Department of Commerce, Tech. Rep. Revision 3, 2018. [Online]. Available: https://csrc.nist.gov/publications/detail/sp/800-56a/rev-3/final

[48] J. Aumasson, S. Neves, Z. Wilcox-O'Hearn, and C. Winnerlein, "BLAKE2X," pp. 2–5, 2016. [Online]. Available: https://blake2.net/blake2x.pdf

[49] P. D. Gallagher and C. Romine, "FIPS PUB 186-4 Digital Signature Standard (DSS)," *Encyclopedia of Cryptography and Security*, no. July, pp. 347–347, 2013. [Online]. Available: http://dx.doi.org/10.6028/NIST.FIPS.186-4

[50] J. Merkle and M. Lochter, "RFC 7027 - Elliptic Curve Cryptography (ECC) Brainpool Curves for Transport Layer Security (TLS)." [Online]. Available: https://tools.ietf.org/html/rfc7027