Sapienza
Università di Roma

Design, Analysis and Evaluation of a new Secure Rejoin Mechanism for LoRaWAN using Elliptic-Curve Cryptography

Department of Computer, Control and Management Engineering

Corso di Laurea Magistrale in Engineering in Computer Science

Candidate

Stefano Milani
ID number 1707181

Thesis Advisor

Prof. Ioannis Chatzigiannakis

Academic Year 2019/2020

Design, Analysis and Evaluation of a new Secure Rejoin Mechanism for LoRaWAN using Elliptic-Curve Cryptography
Master's thesis. Sapienza – University of Rome

This thesis has been typeset by LaTeX and the Sapthesis class.

Version: March 19, 2021

Author's email: stefano.milani96@gmail.com

"Chi dice che il mondo è meraviglioso
non ha visto quello che ti stai creando per restarci"
Michele Salvemini, in arte Caparezza

Abstract

LoRaWAN is one of the most promising LPWANs, and the Internet of Things applications using it are rapidly increasing, so it is vital to guarantee a high-security standard.
The thesis is composed by two parts: a research on LoRaWAN security based on a scientific paper that will be published shortly; the design and the implementation of an Industry 4.0 LoRaWAN use case.
In the first part of this thesis, we will focus on the activation schemes of LoRaWAN, in particular on the Over-The-Air Activation Method (OTAA).
After a brief introduction of OTAA, we will analyze the possible vulnerabilities, taking into account related works relative both to LoRaWAN and to other IoT-specific technologies. Then, we propose a new Rejoin-Request to improve the security of the IoT resource-constrained devices using the Over The Air Activation methods, then we will discuss the time and energy consumption of the proposed new protocol.
The last chapter presents and describes the architecture of ssense, an Industry 4.0 use case of LoRaWAN, developed in Arpsoft S.r.l. during these months.

Acknowledgments

I really would like to thank my advisor, Professor Ioannis Chatzigiannakis, for giving me the opportunity of working on stimulant research projects. The sincerest and heartfelt gratitude to him advice and guidance, from both professional and human perspective. During these months I had the pleasure to develop part of the work into Arpsoft S.r.l., I would like to sincerely thank Alessandro Ardito, Massimiliano Mattioli, Stefano Finocchiaro and Francesco Barbaglio for their advice, suggestions and patience, which have greatly enhanced my experience.
A special thank also to my family, my friends and my colleagues with whom I experienced several beautiful experiences during my academic years.

Contents

Chapter 1

Introduction

The Internet of Things is growing exponentially, and in years our cities, our houses will be filled with IoT devices exchanging data and helping us in every aspect of our life. Most of these devices are cheap sensors and actuators, and they are exposed to security threats.
Securing the enormous network composed of the "things" is crucial since they can exchange sensible data or they can control critical features, for example in a Smart City, the traffic lights or the street lights, or a security system in our house.
To support the huge number of IoT devices new technologies are emerging like the Low-Power Wide Area Networks (LPWANs) [40], that permits, as the name points, to connect "things" also at big distances, with reduced energy consumption and computational power. The LPWANs are promising since they offer the possibility to develop scalable networks and at the same time guarantee a long lifetime for IoT devices.
LPWANs are emerging technologies thus they can lack security, so it is vital to define some security protocols and standards for this type of network. The "classical" Internet communication makes use in the majority of the case IP-based protocols, with well-defined security standards. These standards need special attention to be used in Low-Power Wide Area Network [18], and the limitations in terms of packet size, data rate, and in some cases the elevated Time-On-Air could be a problem. In addition to that, the IoT devices cannot make use of cryptographic primitives that require high computational power, or that are not energy efficient, since they have to guarantee a long lifetime.
LoRaWAN is one of the most interesting LPWANs, developed by the LoRa Alliance as an open standard [27], its private and public networks are spreading all over the world [53]. This thesis focuses on LoRaWAN activation methods, i.e. on the key agreement protocols between the end-devices and the Network or Join Server, analyzing the possible vulnerabilities. In particular, the focus is on the **Over-The-**

Air activation method, which makes use of two root keys, the **NwkKey** and the **AppKey** to compute the session keys. Those keys are hard-coded, into the device that means that even in the case of device restart those never change, and the parties have to restart the process of activation with the same root keys.
In the actual specification, it is possible to refresh the session keys, but it is not possible to refresh the aforementioned root keys. That can be seen as a weak point of OTAA and it can be a vulnerability, taking into account that a LoRaWAN device should have a lifetime of at least 10 years.
This work proposes a new type of Rejoin-Request and Join-Accept message that allows refreshing also the root keys using the **Elliptic-Curve Cryptography** and without modifying the actual architecture, in this way, we ensure the backward-compatibility with the actual specification, taking into account that a **Key Agreement protocol** must guarantees key freshness [11], and it must encounter these cryptographic properties [22, 26]: **Computational Key Secrecy**, **Decisional Key Secrecy**, and **Key Independence**.
During the period of my thesis I have done an Internship in **Arpsoft S.r.l.**, where I designed and implemented an Industry 4.0 LoRaWAN use case **ssense**. The application's goal is to offer the end-user a web-based interface in which the user can control and visualize in several ways some data collected by LoRaWAN sensors. The actual version of *ssense* is focused on environmental sensors, in particular temperature, humidity, light, motion, and CO2 level, but the architecture of the application is scalable and can adapt easily to new type of sensors.
Moreover, *ssense* allows the user to manage a hierarchy of points of interest where the sensors are actually mounted. The hierarchy starts from groups of geographical areas, to the single room on a building or to the single sensor.
The rest of the thesis is organized as follows: **Chapter 2** summarizes the state of the art regarding LPWANs security listing some related works and comparing them to my proposal; in **Chapter 3** I describe the main technologies I used to reach my goal: *LoRaWAN*, the *Over-The-Air Activation Method*, the *Elliptic-Curve Cryptography*, and the *Elliptic-Curve Diffie-Hellman* protocol. In **Chapther 4** are listed some vulnerabilities of *LoRaWAN1.1*; in **Chapter 5** is described in detail the *proposed new Rejoin Mechanism* for *LoRaWAN1.1 OTAA*, and then in Chapter 6 the protocol is analysed with respect to security and performances point of view.
In **Chapter 7** is described the architecture and the technologies used to implement *ssense*. Finally, in **Chapter 8** are listed the conclusion and the possible future works both regarding *LoRaWAN security* and *ssense*.

Chapter 2

Related and Previous works

The enormous heterogeneity of the IoT devices is expected to magnify security threads with respect that to the current internet, and it is crucial to set high standards of security, privacy, and trust for the IoT application [47]. The security of **LPWANs** is an open issue, and several works analyze possible vulnerabilities.
The authors in [10] perform a detailed analysis on **LoRAWAN** security, ranging from the hazard of physical access to the end-device to the possibility to perform an ACK spoofing attack or an Application-Specific attack. An interesting analysis regards the eavesdropping, which can lead to compromise the encryption method and to decrypt part of the entire cipher-text, in the case of the frame counter reset and the session keys remain the same, for example when we use the Activation By Personalization activation method. That underlines the importance of having an efficient and complete key refresh mechanism in every Key Agreement Protocol.
The authors in [9] lists some potentials vulnerabilities in LoRaWAN1.1. They focus on the possibility of performing a **replay attacks** relaying on jamming techniques, to lead to a Denial Of Service. They also point as a vulnerability the **non-secured beacons** in case of *LoRaWAN class B devices* to de-synchronize the receive windows and the possibility of network analysis.
In [6], similarly to the aforementioned work, the authors find the possibility to combine a jamming attack to a replay attack. In addition, they focus on a replay attack in case of a bad configured application server, i.e. when the frame counter is disabled.
In [48] is proposed a **new architecture for LoRaWAN networks**, in order to reinforce security and to provide an end-to-end secure communication scheme. The authors point as a possible solution a Median Server, a new entity in the architecture that has the role of a registration authority for both end-devices and gateways. To fulfil this purpose a Central Authority is introduced to ensure that only authenticated devices interact with the system, and that they connect only with authenticated

gateways. They introduce also a VPN network to secure the communication between the gateways and the Median/Network servers.
The proposal make use of the **Elliptic-Curve Cryptography** [8, 33] to agree the new pair of root keys between the end-device and the Network Server. ECC permits us to have the same level of security of RSA using smaller parameters [24] for example an elliptic curve over a 283-bit field gives the same level of security as a 3072-bit RSA modulus or Diffie-Hellman prime.
As is shown in [31] and [51], ECC is a better alternative with respect to RSA for resource-constrained devices used in IoT context, both in terms of **computational power** and **energy efficiency**. In particular, in [31] it is shown how using ECC add a small run-time overhead, that is worth the gain in term of security.
Another topic of interest is the key agreement and the group key agreement protocols in resource-constrained context.
The authors in [43] perform an attack exploiting a vulnerabilities in the **ZigBee OTA** features, that allows them to update the firmware of smart lights with a malicious code that spread over others smart bulbs. This shows the importance of a secure Over-The-Air activation, both in the initialization phase of the end-devices, and in the phase of updating the firmware or the keying material.
In [44] is proposed a lightweight bootstrapping protocol for authentication and establishing credentials for **4G/5G** and **NB-IoT** networks. The proposal find **LO-CoAP-EAP** as a feasible and efficient solution for NB-IoT and 5G networks.
The authors in [11] propose a Group Key Establishment protocol in a Wireless Sensor Network following a distributed approach that does not require many-to-many messages and that does not rely on a global ordering of devices. The protocol is interesting for the use of the **Elliptic-Curve Diffie-Hellman** protocol to generate a shared secret between the devices.
In [45] the authors propose some alternatives to the actual key management in LoRAWAN. They explore the possibility of an approach based respectively on **IKEv2**, **DTLS** and **EDHOC**, analysing the pros and cons of each approach, but they do not formalize an alternative key management scheme for LoRaWAN using one of the aforementioned protocols.
The authors in [39] propose two group key agreement protocols for enabling a **secure multi-casting in WSNs**. Both protocols are based on Elliptic Curve Cryptography, that permits to use asymmetric encryption at low cost in term of energy consumption and computational power.
In [17] it is proposed a root key distribution scheme for LoRaWAN OTAA. The authors make use of **Rabbit**, a high-efficient synchronous stream cipher, to generate a new pair of root key. After the generation of the new keys, the end-device will

trigger a new Join-Request with the Join Server.
My proposal offers **more efficient way** to generate a new pair of **root keys**, because it can refresh all the keying material with the exchange of only two messages.
The authors in [55] propose a new key management schema for LoRaWAN based on **Hierarchical Deterministic Wallet** using the BIP32 algorithm [54]. On the contrary, the protocol described in this work adds the possibility to refresh the root keys without **any modification** of the actual architecture of LoRaWAN.

Chapter 3

Technologies

In this chapter, we will introduce the main technologies used to carry on the research work: **LoRaWAN** and the **Elliptic-Curve Cryptography**.

3.1 LoRaWAN1.1

The information contained in this section are based on the **official LoRaWAN1.1 Specification** by LoRa Alliance [5].
LoRaWAN networks are laid out in a star-topology, in which the sensors send the packets to a **gateway** that forward them to a **Network Server**. Afterwards the Network Server send the packets to the application-specific **Application Server**.
The communication between the gateways and the different types of server are based on the Internet standard TCP/IP protocol, on the contrary the communication between the sensors and the gateways are based on LoRa.
The LoRaWAN devices are distinguished in three classes:

- **Class A:** each up-link message is followed by two receiving windows, so the device listen and can receive messages only during these time windows.
- **Class B:** in addition to the Class A devices, they can open extra receiving windows at scheduled time, synchronized by beacons sent by the gateways.
- **Class C:** the devices have a nearly continuous open receiving window. They cannot receive messages only when they are sending a packet.

The power consumption is greater in case of **Class C** devices and is smaller in case of **Class A** device, so the **Class A** device can guarantee a longer battery life-time. LoRaWAN operates in unlicensed radio spectrum, that allows to anyone to use the radio frequencies without paying any fee [52]. The frequencies in which LoRaWAN operates are different with respect to the region, for example in Europe is used the 863-870MHz ISM Band [4].

3.1.1 Activation Methods

LoRaWAN has two types of activation to ensure a secure communication between the end-device and the Join/Application Server:

- **Activation By Personalization - ABP:** this activation method is the simplest, but is also the less secure. It make use of a set of predetermined keys, used as session and integrity keys. These keys are hard-coded into the end-device, so it is not possible to refresh them.
- **Over The Air Activation - OTAA:** This activation method permits to the end-device and the Join Server to agree upon the session and the integrity keys thanks to an handshake. OTAA makes use of some hard-coded keys that are used to determine the needed keys.

While ABP is used in particular in development phase, cause is faster, OTAA is the recommended activation method since it guarantee an higher level of security for the communication since it permits to refresh the session and integrity keys.

3.1.2 Activation By Personalization - ABP

The **Activation by Personalization** directly tied up the end-device to the Network/Application Server by hard-coding the session and integrity keys directly into the device. That allows to establish a secure communication in a faster way as long as a join procedure is not required. On the other hand, this activation method is not secure, since the packets are encrypted with the same key for the entire life of the device.

3.1.3 Over-The-Air Activation - OTAA

Before starting the actual handshake, both the end-device and the Join Server need some data that they will use to agree on the keys.

- **DevEUI:** a global end-device identifier of 64 bits that uniquely identify the device. The DevEUI must be stored into the device before the activation. It can be public, and it is a recommended practice to make it available on the device label.

- **JoinEUI:** a global application identifier of 64 bits that uniquely identifies the Join Server. The JoinEUI must be stored into the device before activation.
- **Device Root Keys:** the **NwkKey** and the **AppKey** are AES-128 root keys specific to the end device that are assigned to it during fabrication. These keys are the ones used to derive the session and integrity keys. They never change during the life-cycle of the end-device, so securing distribution, storage and usage of them, both in the end-device and in the Join Server, is crucial for the entire security of the protocol.

The last data needed before activation are two keys derived from the **NwkKey**:

- **JSIntKey:** used to compute the MIC (Message integrity code) for the Rejoin-Request message and the Join-Accept message.

$$JSIntKey = aes128\_encrypt(NwkKey, 0x06|DevEUI|pad_{16})$$

- **JSEncKey:** used to encrypt the Join-Accept message triggered by a Rejoin-Request.

$$JSEncKey = aes\_encrypt(NwkKey, 0x05|DevEUI|pad_{16})$$

Data computed during activation

The end-device after the activation will have these additional information:

- The **DevAddr**, 32 bits that identify the end-device within the LoRa network. It is allocated by the Network-Server.
- The **Forwarding Network Session Integrity Key - FNwkSIntKey**, used by the device to calculate the *MIC* for all up-link messages.
- The **Serving Network Session Integrity Key - SNwkSIntKey**, used by the end-device to check the *MIC* of all down-link messages.
- The **Network Session Encryption Key - NwkSEncKey**, used to encrypt and decrypt both up-link and down-link MAC commands transmitted as payload on *port* 0 or in the *FOpt field*.
- The **Application Session Key - AppSKey**, used by both the device and the Application Server to encrypt and decrypt the payload field of application-specific messages.

Join-Request Message

The join procedure is started by the end-device by sending a **Join-Request message**. The message contains the **JoinEUI**, the **DevEUI** and a **DevNonce**, that is a counter, starting at 0, incremented at each *Join-Request message*. It is used as a countermeasure for the replay attack since it can not be used twice for a given *JoinEUI*.

JoinEUI	**DevEUI**	**DevNonce**
8 Bytes	*8 Bytes*	*2 Bytes*

Figure 3.1. Join-Request message

The *MIC* for the message is computed as follows:

$$cmac = aes128\_cmac(NwkKey, MHDR|JoinEUI|DevEUI|DevNonce)$$

$$MIC = cmac[0...3]$$

The Join-Request **is NOT encrypted**.

Join-Accept Message

The Join-Accept message contains a **JoinNonce**, a network identifier **NetID**, the **DevAddr**, a **DLSettings** field providing some down-link parameters, the **RxDelay**, and an optional list **CFList** containing network parameters.

JoinNonce	**Home_NetID**	**DevAddr**	**DLSettings**	**RxDelay**	**CFList**
3 Bytes	*3 Bytes*	*4 Bytes*	*1 Byte*	*1 Byte*	*16 Bytes (optional)*

Figure 3.2. Join-Accept message

The *JoinNonce* is a device specific counter value, that never repeats itself, provided by the Join Server and used by the end-device to derive the session and integrity keys. It is incremented at each Join-Request message.

Session and Integrity keys derivation

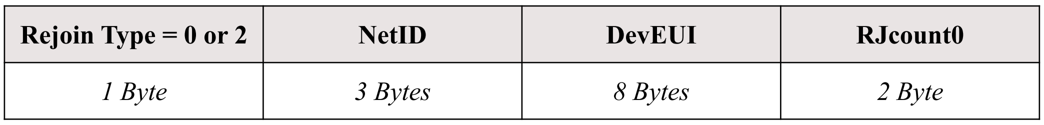
$$
\begin{aligned}
FNwkSIntKey &= aes128\_encrypt(NwkKey, 0x01|JoinNonce|JoinEUI|DevNonce|pad_{16})\\
SNwkSIntKey &= aes128\_encrypt(NwkKey, 0x03|JoinNonce|JoinEUI|DevNonce|pad_{16})\\
NwkSEncKey &= aes128\_encrypt(NwkKey, 0x04|JoinNonce|JoinEUI|DevNonce|pad_{16})\\
AppSKey &= aes128\_encrypt(AppKey,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

Rejoin-Request Message

Once activated an end-device may periodically transmit a Rejoin-Request message, which gives the server the possibility to initialize a new session context for an end-device.
There exist three types of Rejoin-Request:

- **Rejoin-Request type 0:** contains **NetId** and **DevEUI**. Used to reset a device context including all radio parameters.
- **Rejoin-Request type 1:** contains **JoinEUI** and **DevEUI**. Equivalent to type 0, but transmitted on top of normal application traffic without disconnecting the end-device.
- **Rejoin-Request type 2:** contains the **NetId** and **DevEUI**. Used to re-key a device or change its **devAddr**, the radio parameters are kept unchanged.

The Rejoin-Request **is NOT encrypted**.
The **Rejoin-Request of type 0 and 2** make use of counter, **RJcount0**, incremented at every type 0 or 2 Rejoin frame transmitted. If the *RJcount0* reaches $2^{16} - 1$ the device shall stop transmitting Rejoin-Request of the given types, and restart completely the Join Request.



Rejoin Type = 0 or 2	NetID	DevEUI	RJcount0
1 Byte	*3 Bytes*	*8 Bytes*	*2 Byte*

Figure 3.3. Rejoin-Request type 0 or 2 message

The **Rejoin-Request of type 1** make use of another counter, **RJcount1**, incremented at every type 1 Rejoin-Request message transmitted. In this case, the counter shall never warp around, due to the lifecycle of the device for a given *JoinEUI* value.

Rejoin Type = 1	JoinEUI	DevEUI	RJcount1
1 Byte	*8 Bytes*	*8 Bytes*	*2 Byte*

Figure 3.4. Rejoin-Request type 1 message

Rejoin-Request Message Processing

For all the three types of Rejoin-Request message the Network Server respond with:

- A *Join Accept* message to modify the device's network identity. The *RJcount0* or the *RJcount1* replaces the *DevNonce* in the key derivation.
- A normal down-link frame optionally containing MAC commands.

In most cases following a Rejoin-Request message of type 0 or 1, the server will not respond.

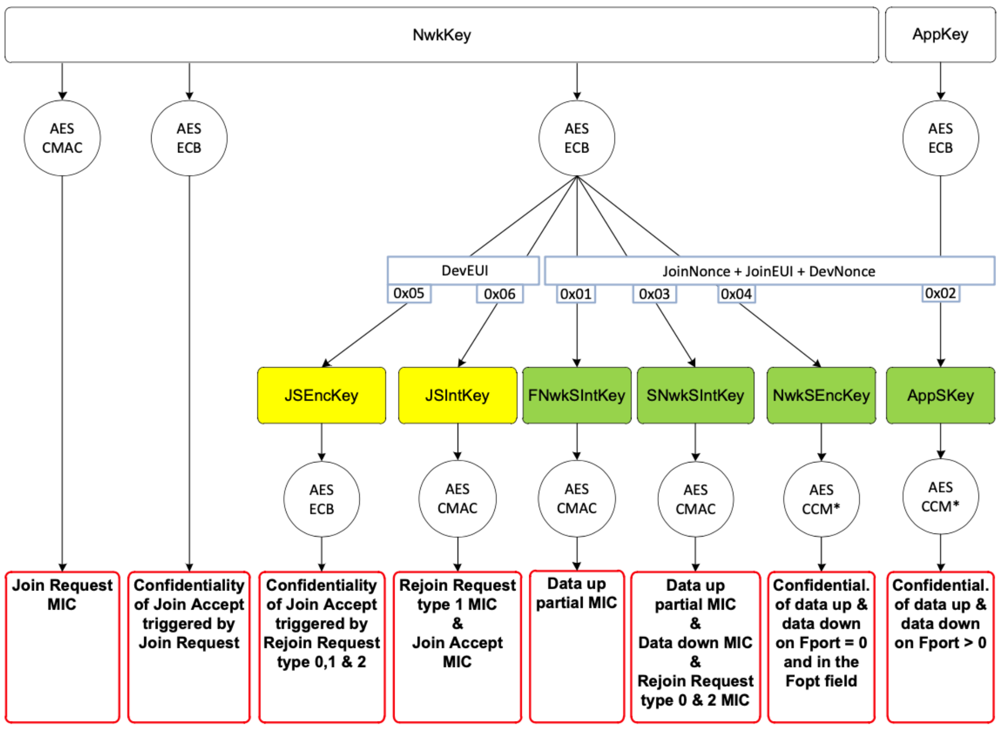

NwkKey
AppKey
AES CMAC
AES ECB
AES ECB
AES ECB
DevEUI
JoinNonce + JoinEUI + DevNonce
0x05
0x06
0x01
0x03
0x04
0x02
JSEncKey
JSIntKey
FNwkSIntKey
SNwkSIntKey
NwkSEncKey
AppSKey
AES ECB
AES CMAC
AES CMAC
AES CMAC
AES CCM*
AES CCM*
Join Request MIC
Confidentiality of Join Accept triggered by Join Request
Confidentiality of Join Accept triggered by Rejoin Request type 0,1 & 2
Rejoin Request type 1 MIC & Join Accept MIC
Data up partial MIC
Data up partial MIC & Data down MIC & Rejoin Request type 0 & 2 MIC
Confidential. of data up & data down on Fport = 0 and in the Fopt field
Confidential. of data up & data down on Fport > 0

Figure 3.5. LoRaWAN 1.1 Key Derivation scheme [5]

3.2 Elliptic-Curve Cryptography - ECC

The **Elliptic-Curve Cryptography** curves are implemented over two number fields, the prime field, and the binary field. The **binary field curve** has worse performance and energy efficiency when executed on general-purpose processors [51], but outperforms the **prime field curve** if dedicated hardware is used. Since the IoT devices are heterogeneous and in most cases are cheap boards with no particular hardware capabilities, we pointed towards a prime field curve to implement the new type of Rejoin. In this way, we can assure good performance and energy efficiency for all devices.
In particular, we found the **secp256k1** and the **secp256r1** curves as the best candidates, because smaller curves do not guarantee always better performances [51], and the key size of these two curves fit the purpose of the proposal.
The main difference between the candidates curves is that the *secp258k1* curve is generated over a prime field associated with a **Koblitz** curve, on the other hand, the *secp256r1* curve is generated over random domain parameters. Koblitz curves are generally less secure but in 256-bit curve, the impact is minimal [7]. The *secp256k1* curve is generally faster than the other curve if the implementation is optimized, in particular for the signature generation and verification, and that is one of the motives the curve is used by Bitcoin [19]. On the other hand we can see in [46] that the elapsed time difference for public key and secret generation in the two types of curves is very low.
Since the difference in terms of performance and energy efficiency for our purpose are negligible, we choose the *secp256r1* curve, since it offers a major level of security.

3.2.1 Elliptic-Curve Diffie-Hellman

To fulfill the goal of the new proposal, we need a key agreement protocol, to refresh the LoRaWAN root keys. The **Elliptic-Curve Diffie-Hellman** protocol allows to do that using the **Elliptic-Curve Cryptography**. The *ECDH* is is similar to the classical **Diffie-Hellman Key Exchange** protocol but it uses **ECC point Multiplication** instead of **modular exponentiation** [32].
The *ECDH* protocol works as follows [3]:

1. The two parties A and B have to agree on a common **Elliptic-Curve Group G** of order n, and on a **Primitive Element P** in G of order n. In our case we choose the *secp256r1* curve that defines these parameters.

2. A selects an integer $a \in [2,\ n-1]$ and it computes $Q = [a]P$. Where Q is the public key of A, and a is the private key.

3. B selects an integer $b \in [2,\ n-1]$ and computes it $R = [b]P$. Where R is the public key of B, and b the private key.
4. The two parties exchange their public keys.
5. A computes the secret $S_A = [a]R = [a][b]P$
6. B computes the secret $S_B = [b]Q = [b][a]P$
7. The two parties have a common secret $S_A = [a]R = [a][b]P = [b]Q = S_B$

Chapter 4

Vulnerabilities in LoRaWAN1.1

To analyse the possible vulnerabilities of **LoRaWAN1.1 Over-The-Air-Activation method** we have to take into account the goals of the attacker [56]. This thesis focuses on the key agreement between the end-device and the the Join/Application Server, so we can suppose that the attacker aims to collect information that can help it to guess the session and integrity keys used to encrypt the application messages. If an attacker knows those keys it can decrypt the messages and it can impersonate the end-device as long as a new Rejoin-Request is successfully completed.
The attacker has another goal, i.e. to guess the root keys, in this case the attacker can decrypt all the messages and it can impersonate the end-device until the end of its lifetime.

4.1 Replay Attack

The authors in [9] point the replay attack as a vulnerability of OTAA. Using a selective RF jamming technique an attacker can jam and capture a first Join-Request from an end-device. The device will re-transmit a second Join-Request after a timeout, since it will not receive a Join Accept message. The second Join-Request message will be jammed, and the first jammed Join-Request is transmitted by the attacker. The Network Server will respond to the first Join-Request with a Join Accept message. From now on, the Network Server, the Join Server and the end-device are de-synchronized.

4.2 Passive Man In The Middle Attack

An attacker performing a simple Passive Man In The Middle Attack, can eavesdrop all messages from/to and end-device, and read the content if not encrypted, and it can collect some useful information used during the OTAA procedure.

The information useful to the attacker to compromise the session and the integrity keys are:

- **NwkKey/AppKey**
- **JoinNonce**
- **JoinEUI**
- **DevNonce**

When an end-device transmit a Join-Request message, since it is not encrypted the attacker collects the **JoinEUI** and the **DevNonce**. These information are always available to the attacker even in case of a Rejoin-Request message, since also this message is unencrypted. From a Rejoin-Request of type 0 or 2, an attacker can extract the **DevEUI**, and the **RJcount0**. From a Rejoin-Request message of type 2 instead he can collect the **JoinEUI** and the **RJcount1**. The **RJcount0** or the **RJcount1** take the place of the **DevNonce** during a rejoin request. So an attacker knows the **DevNonce** and the **JoinEUI** at each time of the life-cycle of the device. As we seen in **Section 3.1.3** the **JoinNonce** is a device specific counter incremented at each Join-Request message, so it is easy for the attacker to compute the value of this information counting the number of Join-Request the end-device send to the Network Server.

4.3 Root keys

Since an attacker can collect the **JoinNonce**, the **JoinEUI** and the **DevNonce**, simply performing a MITM attack, the security of OTAA is based on the the storage of the **NwkKey** and the **AppKey**.
These root key are hard-coded into the end-device during fabrication, and they never change during the entire life-cycle of the device. Now the only way to refresh those keys is to manually change them, both in the Network/Application Server and into the device. This process can be expensive, in particular for devices placed in hard reachable locations.
To overcome this issue, in the next section we propose a new type of Rejoin-Request that permits to refresh the root keys periodically.

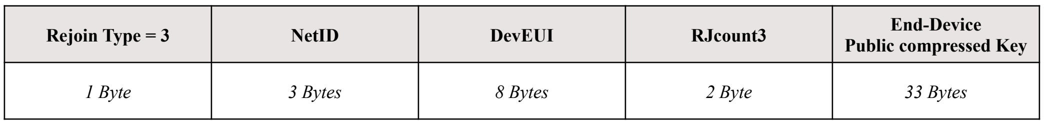
Chapter 5

New Rejoin-Request message

In this section is described in details our proposal.
To maintain the actual architecture of LoRaWAN we offer a new type of Rejoin-Request message and a respective new type of Join-Accept message. Thereby our solution is backwards compatible with the three type of Rejoin-Request previously described, moreover we maintain the same philosophy, so with the exchange of only two messages it is possible to refresh all the keying material, including the root keys.

5.1 Rejoin-Request message of type 3

We propose a new type of **Rejoin-Request** message, that permits to refresh all the key materials at once.
First of all, the end-device have to generate a new pair of private/public ECC keys using the *secpr256r1* curve.
The Rejoin-Request of type 3 contains the **RejoinType**, one byte equals to 3; the **NetID**; the **DevEUI**; the **RJcount3**, that behaves in the same way of the RJcount0 and RJcount1, and it will substitute the DevNonce for the subsequent session key generation; and previously generated **compressed public key** of the device.



Rejoin Type = 3	NetID	DevEUI	RJcount3	End-Device Public compressed Key
1 Byte	*3 Bytes*	*8 Bytes*	*2 Byte*	*33 Bytes*

Figure 5.1. Rejoin-Request type 3 message

The **RJcount3** is a counter that is incremented at each Rejoin-Request of type 3, it shall never wrap around, due to the lifecycle of the device for a given **NwkKey** and **AppKey** value. The **RJcount3** prevents replay attacks for the given type of the Rejoin-Request, since the Network Server discards all the Rejoin-Request messages with a counter value less or equal to the last Rejoin-Request type 3 valid message. The **RJcount3** restarts from 0 after a successful Rejoin Request of all type, so it has to be unique for a given combinations of values of **NwkKey**, **AppKey**, and sessions keys.
The **Message Integrity Code** of the Rejoin-Request message of type 3 is computed as follows:

$$cmac = aes128\_cmac(SNwkSIntKey,$$

$$MHDR|RejoinType|NetID|DevEUI|RJcount3|DeviceCompressedPublicKey)$$

$$MIC = cmac[0...3]$$

This message will be sent not encrypted, like the other types of Rejoin-Request.

5.2 Join-Accept message of type 1

The Network Server that receive a valid Join-Request message of type 3, will respond to the end-device with a **Join-Accept** message of type 1. Before sending the message, it have to generate a new pair of private/public keys using the *secp256r1* curve.
The **Join-Accept** message of type 1 contains: the **JoinNonce**, the **Home_NetID**, the **DevAddr**, the **DLSetting**, the **RxDelay**, and the **compressed public key** of the network Server. The **MIC** of the Join-Accept message of type 1 is computed

Join Nonce	Home_NetId	DevAddr	DLSettings	RxDelay	Server Public compressed Key
3 Byte	*3 Bytes*	*4 Bytes*	*1 Byte*	*1 Byte*	*33 Bytes*

Figure 5.2. Join-Accept type 1 message

as follows:

$$cmac = aes128\_cmac(JSIntKey,$$

$$RejoinRequestType|JoinEUI|RJcount3|MDHR|JoinNonce|NetiD|DevAddr|$$

$$DLSetting|RxDelay|ServerCompressedPublicKey)$$

$$MIC = cmac[0...3]$$

The Join-Accept type 1 message will be encrypted as follows:

$$aes128\_decrypt(JSEncKey, JoinNonce|NetID|DevAddr|DLSettings|RxDelay|$$

$$ServerCompressedPublicKey|MIC|pad_{16})$$

Where pad_{16} is a pad of 3 bytes to make the Join-Accept message 48 bytes long, multiple of 16. It is used the AES decrypt operation in ECB mode to encrypt the message, the same as the actual Join-Accept message.

5.3 NwkKey and AppKey generation

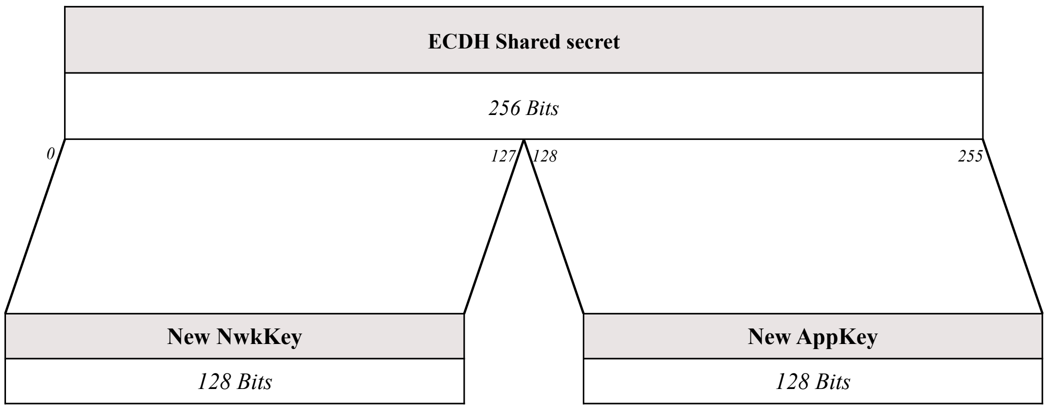


Figure 5.3. New Root Keys Derivation

Once the Network Server has received the Rejoin-Request of type 3 message and it has generated its own private/public pair of keys, it can compute a secret using the public key of the end-device and its private key using the **Elliptic-Curve Diffie-Hellman** protocol [28].
The end-device, after receiving the Join-Accept message of type 2, can also compute the same secret using its private key and the public key of the Network Server.
Both the end-device and the Network Server have a common 256-bit secret, the 128 most significant bits will be used as the **new NwkKey**, meanwhile the 128 less significant bits as the **new AppKey** (*Figure 5.3*). After the generation of the new root keys both the end-device and the Network Server can discard the private/public keys previously generated, and they can recompute all keying material: **JSIntKey**, **JSEncKey**, **FNwkSIntKey**, **SNwkSIntKey**, **NewSEncKey** and **AppSKey**.
In case of a reboot, the device needs to reissue a new Join-Request as specified in the actual specification, but the two parties will use the newly computed root keys to generate the session and integrity keys.

Chapter 6

Security and Performance Analysis

In this chapter the proposal defined in **Chapter 5** will be analyzed with respect to security and performances points of view.

6.1 Security Analysis

In this section the security of our proposal is analyzed with respect to the vulnerabilities listed in **Chapter 4**.

6.1.1 Replay Attack

In the Rejoin-Request type 3, the **RJcount3** is a counter that is incremented at each Rejoin-Request of this type. Its value substitutes the **DevNonce** in the session keys generation, and it used to prevent Replay Attacks, since the Join Server will discard a Rejoin-Request of type 3 with a value of **RJcount3** lesser than or equals to the last valid Join-Request of type 3 received. The device and the Join Server reset that counter each time a Rejoin-request of each type is successfully completed, that ensure to have unique **RJcount3**, even in the case of device reboot.
Our proposal remains vulnerable to a Replay attack combined with a selective RF jamming attack as described in **Section 4.1**.

6.1.2 Man In The Middle Attack - MITM

An attacker performing a MITM attack can eavesdrops the Rejoin-Request message since it is not encrypted. So it can access the following information: the **NetID**, the **DevEUI**, the **RJcount3** (i.e. the DevNonce), and the **end-device public key**.
The attacker cannot obtain any information from the Join-Accept type 1 message,

since it is encrypted, but it can derive the **JoinNonce** as we already described in **Section 4.2**.
The information the attack can obtain or derive are not sufficient to guess the value of the new **NwkKey** and **AppKey** keys, or the new session keys.
The attacker cannot compute the value of the ECDH shared secret knowing only the public key of the end-device.

6.1.3 Keys freshness

Since a LoRaWAN device is supposed to work for at least ten years, it is important to not use the same keys for this long amount of time, so our proposal offers a way to refresh periodically those keys, enhancing the overall security of the entire activation method.
Moreover, a Key Agreement protocol must guarantees key freshness [11], and regarding this requirement it must encounter these cryptographic properties [22, 26]:

- **Computational key secrecy:** it must be computational infeasible for any passive adversary to discover any key.
- **Decisional key secrecy:** there must be no information leaked other that public key information.
- **Key independence:** a passive adversary that knows a subset of keys must not discovery any other information of the remaining keys. This property decompose into:
  - **Forward Secrecy:** a passive adversary that knows a subset of keys must not discover any subsequent keys.
  - **Backward secrecy:** a passive adversary that knows a subset of keys must not discover any preceding keys.

LoRaWAN1.1 current specification complies with these requirements and properties concerning the integrity and session keys (**FNwkSIntKey, SNwkSIntKey, NwkSEncKey, AppSKey**), but it does not guarantee key freshness in the case of the root keys (**NwkKey, AppKey**) and consequently for the **JSIntKey** and **JSEncKey** keys.
Our proposal guarantees key freshness also for those keys, and it encounters the aforementioned cryptographic properties. An adversary can know only the public key information, and thanks to the the Elliptic Curve Cryptography, it is computationally infeasible for him to know the private keys of both parties [25], or the shared secret computed via *ECDH*. So the **Computational** and **Decisional key**

secrecy are guaranteed.
Key independence is respected because the new root keys are completely independent from the previous and the subsequent ones, because the **Elliptic-curve Diffie-Hellman** protocol is used to compute the new root keys. In addition, the pair of private/public ECC keys is discarded, and it is generated a new pair from scratch at each Rejoin-Request, both from the server an the end-device.

6.2 Performance Analysis

The power and time consumption of each operations that our proposal introduced is tested with respect to the encryption of a single 128-bits AES block of plain-text, using several System On a Chip, with different processors and hardware support.
The tests are implemented using **RIOT Operating System** [42], in particular with the support of the **micro-ecc** library [23] that implement ECDH and ECDSA for 8-bit, 32-bit, and 64-bit processors. To implement the 128-bits AES encryption I used the **Crypto** module available in *RIOT OS* [41].
The test are executed on the FIT IoT-LAB testbed [2], in particular on six different development boards: the **IoT-LAB M3**, the **SAMR21 Xplained Pro**, the **ST B-L072Z-LRWAN1**, the **nRF51DK**, the **nRF52DK**, and the **nRF52840DK**.

SoC	CPU	Frequency	RAM	AES HW support	HWPRNG
nRF51422	Cortex-M0	$16MHz$	$20KB$	**YES**	**NO**
STM32L072CZ	Cortex-M0+	$32MHz$	$20KB$	**YES**	**NO**
ATSAMR21G18A	Cortex-M0+	$2.4GHz$	$32KB$	**YES**	**NO**
STM32F103REY	Cortex-M3	$72MHz$	$64KB$	**NO**	**NO**
nRF52832	Cortex-M4	$64MHz$	$64KB$	**YES**	**NO**
nRF52840	Cortex-M4	$64MHz$	$256KB$	**YES**	**NO**

Table 6.1. System on Chips Hardware overview

6.2.1 ARM Cortex-M0

The **ST B-L072Z-LRWAN1** board uses the **STM32L072CZ** microcontroller by *STMicroelectronics* [50] with an **ARM Cortex-M0+** 32-bits RISC core operating at $32MHz$. The microcontroller has up to $192KB$ of Flash program memory, $6KB$ of data EEPROM, $20KB$ of RAM, and it offers hardware support for 128-bits AES.
The **nRF51DK** board is based on **nRF51422** chip by *Nordic Semiconductor* [34]. The SoC has a $16MHz$ **ARM Cortex-M0** core, with up to $256KB$ of Flash memory, and up to $32KB$ of RAM. It has a 128-bit AES/ECB/CCM/AAR co-processor.

	ST B-L072Z-LRWAN1	**nRF51DK**
Compute ECC Public/Private Keys	$523432\mu s$	$1045543\mu s$
Compress ECC Public Key	$20\mu s$	$43\mu s$
Decompress ECC Public Key	$39639\mu s$	$79216\mu s$
Compute EC-DH Secret	$523353\mu s$	$1045496\mu s$
128-bits AES encryption	$116\mu s$	$216\mu s$

Table 6.2. Time consumption for ST B-L072Z-LRWAN1 and nRF51DK boards of each ECC operations and the encryption of a single 128-bits AES block (in microseconds)

The **SAMR21 Xplained Pro** board is based on the **ATSAMR21G18A** SoC by *Microchip* [29]. It has at its core an **ARM Cortex-M0+** working on $2.4GHz$ frequency, with a Flash memory of $256KB$ and a SRAM of $32KB$. Furthermore, the SoC offers a 128-bits AES crypto engine to enhance the performances of AES operations, but it does not offer any support for the Pseudo-Random Number Generator.

6.2.2 ARM Cortex-M3

The **IoT-LAB M3** board is based on the **STM32F103REY** Microcontroller Unit by *STMicroelectronics* [49]. The MCU uses an **ARM Cortex-M3**, operating at $72MHz$ frequency with a Flash memory of $512KB$ and SRAM up to $64KB$. It offers no hardware support neither for AES, nor for ECC, and nor for Psuedo-Random Number Generation.

	SAMR21 Xplained Pro	**IoT-LAB M3**
Compute ECC Public/Private Keys	$387376\mu s$	$184052\mu s$
Compress ECC Public Key	$22\mu s$	$12\mu s$
Decompress ECC Public Key	$29298\mu s$	$13881\mu s$
Compute EC-DH Secret	$387330\mu s$	$184035\mu s$
128-bits AES encryption	$97\mu s$	$48\mu s$

Table 6.3. Time consumption for IoT-LAB M3 and SAMR21 Xplained Pro boards of each ECC operations and the encryption of a single 128-bits AES block (in microseconds)

6.2.3 ARM Cortex-M4

The **nRF52DK** board has the **nRF52832** SoC by *Nordic Semiconductor* [35], with a $64MHz$ **Cortex-M4 core with Floating-Point Unit**. The chip has up to 512KB of Flash memory, and up to $64KB$ of RAM. Like the previous chip has a 128-bit AES/ECB/CCM/AAR co-processor.
The last board, the **nRF52840DK** has the **nRF52840** chip by *Nordic Semiconductor* [36]. It has an **Cortex-M4 core with Floating-Point Unit** operating at

$64MHz$ frequency, with $1MB$ of Flash memory and $256KB$ of RAM. Moreover it offers the Arm CryptoCell CC310 crytographic security module and a 128 bit AES/ECB/CCM/AAR co-processor.

	nRF52DK	**nRF52840DK**
Compute ECC Public/Private Keys	$161234\mu s$	$161279\mu s$
Compress ECC Public Key	$10\mu s$	$10\mu s$
Decompress ECC Public Key	$11778\mu s$	$11754\mu s$
Compute EC-DH Secret	$161253\mu s$	$161245\mu s$
128-bits AES encryption	$41\mu s$	$41\mu s$

Table 6.4. Time consumption for nRF52DK and nRF52840DK boards of each ECC operations and the encryption of a single 128-bits AES block (in microseconds)

No board has an Hardware Pseudo-Random Number Generator, that increase of a significant factor the time to execute some ECC-based operations. In **Tables 6.3, 6.2 and 6.4** are summarized the results of our tests regarding the time consumption.

6.2.4 Power Consumption

The power consumption of the introduced ECC-based operations are listed in **Table 6.5**, taking into account only the **IoT-LAB M3**, and the **SAMR21 Xplained Pro** boards, since the Consumption Monitor in FIT IoT-LAB testbed was available only for those boards.

	SAMR21 Xplained Pro	**IoT-LAB M3**
Idle State	$0.5200W$	$0.056W$
Compute ECC Public/Private Keys	$0.542W$	$0.130W$
Compress ECC Public Key	$0.5210W$	$0.058W$
Decompress ECC Public Key	$0.5237W$	$0.129W$
Compute EC-DH Secret	$0.5420W$	$0.129W$
128-bits AES encryption	$0.5230W$	$0.068W$

Table 6.5. Power consumption of IoT-LAB M3 and SAMR21 Xplained Pro boards of each ECC operations and the encryption of a single 128-bits AES block (in Watt)

Taking into account that the proposed Rejoin mechanism need to be executed at low frequencies, the overhead in terms of time and power consumption is affordable with respect to the enhancement of the overall security of LoRaWAN1.1 Over-The-Air Activation method. Moreover using System on a Chip with an Hardware Pseudo-Random Number Generator that overhead can be reduced.

Chapter 7

An Industry 4.0 use case: ssense

During the period of my master thesis in parallel with the work of research on LoRaWAN security, I have done an internship in **Arpsoft S.r.l.** where I had the opportunity to design and implement **ssense**, an Industry 4.0 use case for LoRaWAN.
In this chapter I will describe the architecture and the choices we made to create the service.

7.1 Overview

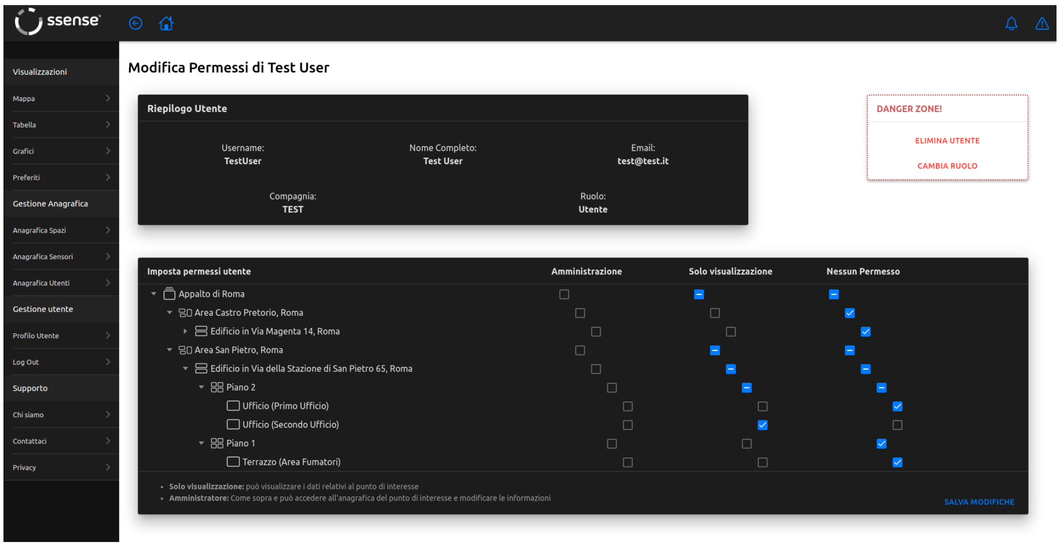
We can locate **ssense** in the smart facilities context, since it aims to create a dashboard for factories, companies, hospitals, etc. to control environmental factors like temperature, humidity, Co2 level or lightning, taking into account the maximum scalability to integrate different and new type of sensors.
Clients have full control of their buildings since *ssense* offers a hierarchical organization of the points of interest: starting from groups of geographical areas, arriving at a single room in a building; and fine management of the users.
ssense defines two main types of users:

- the **administrator**, that has complete access to all the data, it can modify the information of all the points of interest and the he can manage the users, means that he can handles the request of subscription, he can set the roles of the user, and in case of *simple users* he can set the permission level for each point of interest.

- the **simple users** according to the permissions set by an administrator can visualize the data of some or all points of interest, and in some cases he can modify the relative information. He cannot access to the user management options.

The *administrator* can set and modify the access permissions for the *simple user*. *ssense* provides three level of access to each point of interest of the hierarchy.

- **no permission**: the user has no access permission of that point of interest. He cannot visualize its data or know about its existence
- **only read**: the user can visualize the data of the given point
- **admin**: he has access to the data of the given point of interest, but he can also modify the information regarding it



Modifica Permessi di Test User

Riepilogo Utente

Username: **TestUser**
Nome Completo: **Test User**
Email: **test@test.it**
Compagnia: **TEST**
Ruolo: **Utente**

DANGER ZONE!
ELIMINA UTENTE
CAMBIA RUOLO

Imposta permessi utente	Amministrazione	Solo visualizzazione	Nessun Permesso
Appalto di Roma	☐	⊟	⊟
Area Castro Pretorio, Roma	☐	☐	☑
Edificio in Via Magenta 14, Roma	☐	☐	☑
Area San Pietro, Roma	☐	⊟	⊟
Edificio in Via della Stazione di San Pietro 65, Roma	☐	⊟	⊟
Piano 2	☐	⊟	⊟
Ufficio (Primo Ufficio)	☐	☐	☑
Ufficio (Secondo Ufficio)	☐	☑	☐
Piano 1	☐	☐	☑
Terrazzo (Area Fumatori)	☐	☐	☑

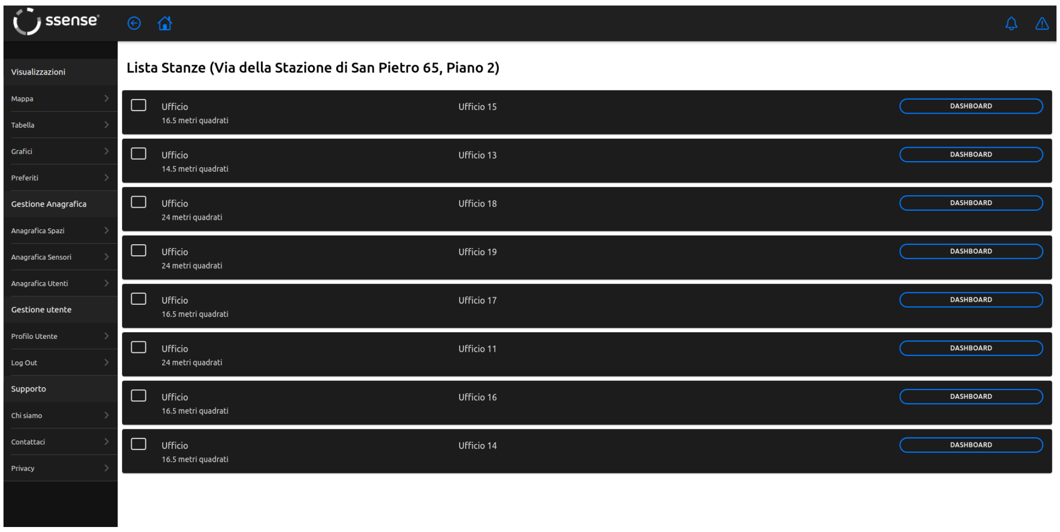
- **Solo visualizzazione:** può visualizzare i dati relativi al punto di interesse
- **Amministratore:** Come sopra e può accedere all'anagrafica del punto di interesse e modificare le informazioni

SALVA MODIFICHE

Figure 7.1. User permissions setting

To visualize the data, it is crucial to offers a simple way to navigate into the hierarchy, to reach this goal we gave to the user two possibilities:

- a list view (**Figure 7.2**) in which you can go inside a certain level and navigate the various elements



Lista Stanze (Via della Stazione di San Pietro 65, Piano 2)

Ufficio 16.5 metri quadrati	Ufficio 15	DASHBOARD
Ufficio 14.5 metri quadrati	Ufficio 13	DASHBOARD
Ufficio 24 metri quadrati	Ufficio 18	DASHBOARD
Ufficio 24 metri quadrati	Ufficio 19	DASHBOARD
Ufficio 16.5 metri quadrati	Ufficio 17	DASHBOARD
Ufficio 24 metri quadrati	Ufficio 11	DASHBOARD
Ufficio 16.5 metri quadrati	Ufficio 16	DASHBOARD
Ufficio 16.5 metri quadrati	Ufficio 14	DASHBOARD

Figure 7.2. Example of list view in ssense

- a map (**Figure 7.3**) that depending on the zoom level you can see a specific level of the hierarchy up to the level of the buildings.

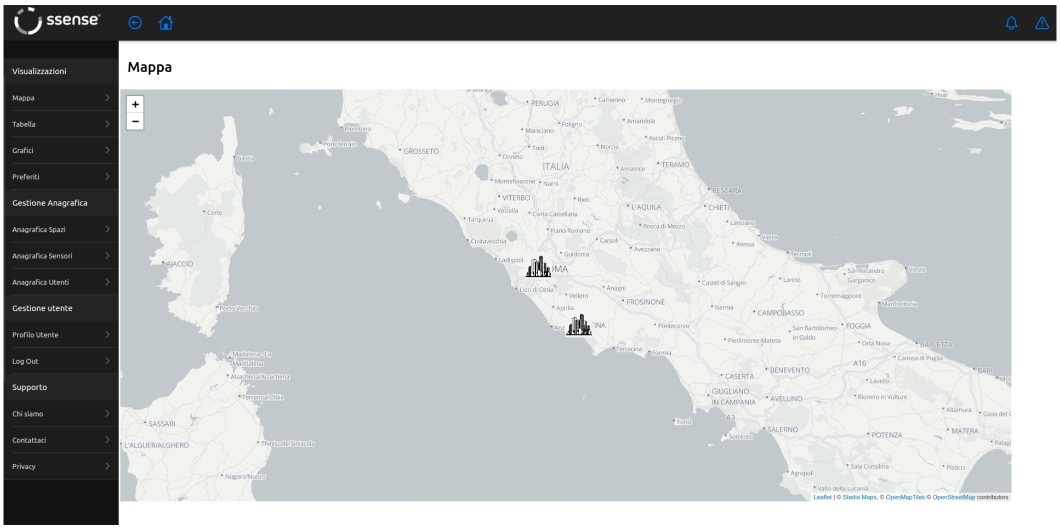


Figure 7.3. Example of map in ssense

At the level of the buildings, for both the list and map view, the user can explore the floors using the list view or using a graphical representation of the building, in which is possible to access the single floor and see the blueprint, containing only the rooms for which the user has at least the read access. *ssense* gives the possibility to create and modify the blueprint of each floor, that are implemented using SVG elements.

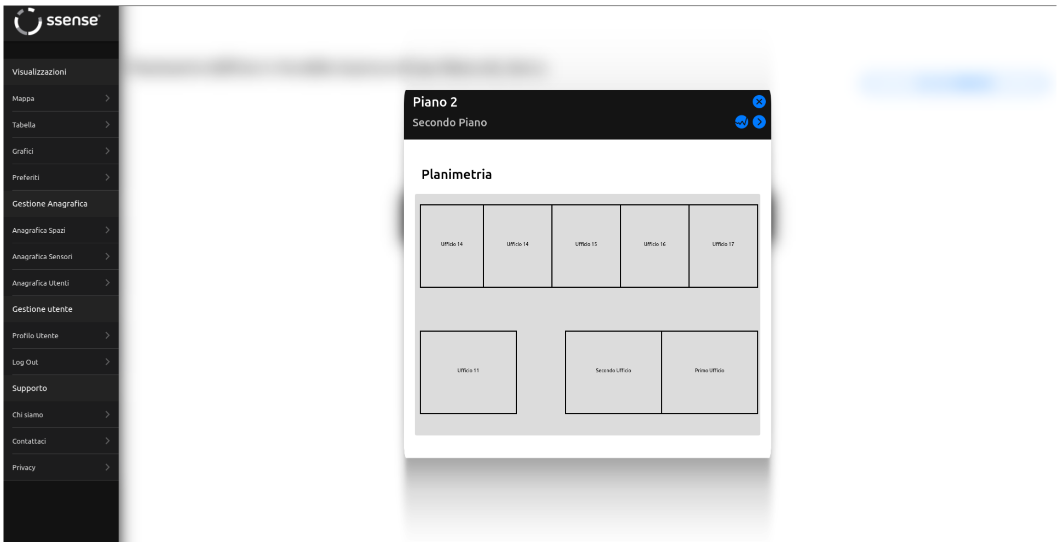


Figure 7.4. Example of floor blueprint in ssense

The end-user can visualize the data collected by the sensors in two ways, via dashboards and via a dynamic table. The **dashboards** (**Figure 7.5**) organize the data in charts and they are available for each buildings, floors and rooms. They gives the possibility to navigate the data through each moment in time, and the charts are dynamic and responsive.

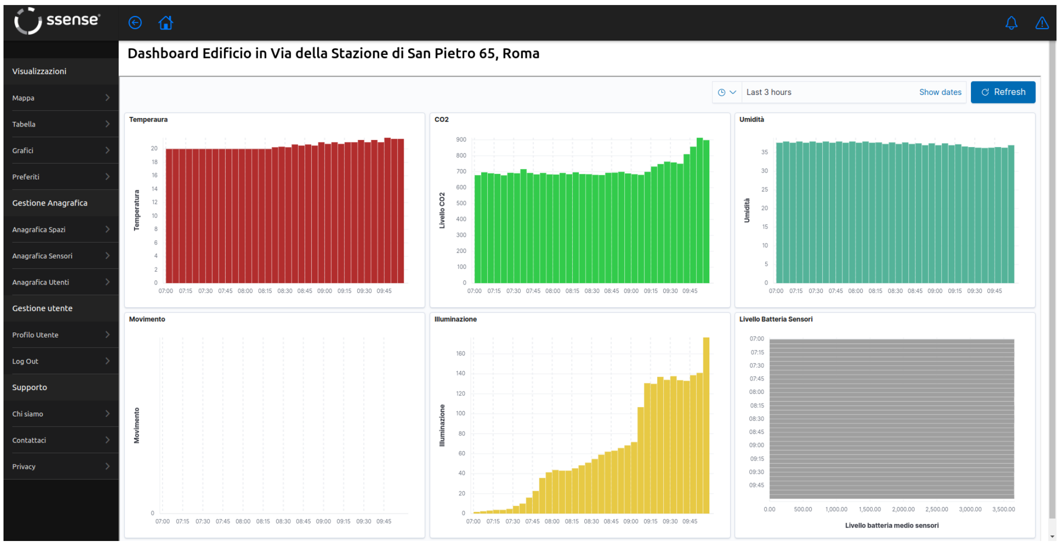
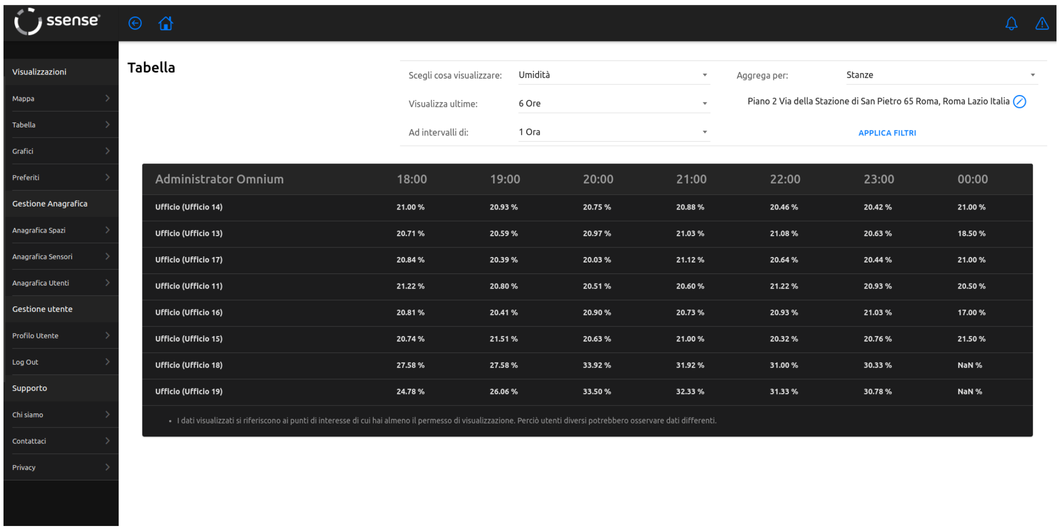


Figure 7.5. Example of dashboard in ssense

On the other hand, using the table (**Figure 7.6**) the user can aggregate the data on each level of the hierarchy and it offers several time range from one hour to 24 hours, in different time intervals.



Tabella

Scegli cosa visualizzare: Umidità

Visualizza ultime: 6 Ore

Ad intervalli di: 1 Ora

Aggrega per: Stanze

Piano 2 Via della Stazione di San Pietro 65 Roma, Roma Lazio Italia

Administrator Omnium	18:00	19:00	20:00	21:00	22:00	23:00	00:00
Ufficio (Ufficio 14)	21.00 %	20.93 %	20.75 %	20.88 %	20.46 %	20.42 %	21.00 %
Ufficio (Ufficio 13)	20.71 %	20.59 %	20.97 %	21.03 %	21.08 %	20.63 %	18.50 %
Ufficio (Ufficio 17)	20.84 %	20.39 %	20.03 %	21.12 %	20.64 %	20.44 %	21.00 %
Ufficio (Ufficio 11)	21.22 %	20.80 %	20.51 %	20.60 %	21.22 %	20.93 %	20.50 %
Ufficio (Ufficio 16)	20.81 %	20.41 %	20.90 %	20.73 %	20.93 %	21.03 %	17.00 %
Ufficio (Ufficio 15)	20.74 %	21.51 %	20.63 %	21.00 %	20.32 %	20.76 %	21.50 %
Ufficio (Ufficio 18)	27.58 %	27.58 %	33.92 %	31.92 %	31.00 %	30.33 %	NaN %
Ufficio (Ufficio 19)	24.78 %	26.06 %	33.50 %	32.33 %	31.33 %	30.78 %	NaN %

- I dati visualizzati si riferiscono ai punti di interesse di cui hai almeno il permesso di visualizzazione. Perciò utenti diversi potrebbero osservare dati differenti.

Figure 7.6. Example of table in ssense

7.2 End-to-End Architecture

To reach our goals we needed to orchestrate heterogeneous services and technologies, starting from the Wireless Sensor Network to the user interface.

7.2.1 Wireless Sensors Network

The data are collected by LoRaWAN sensors, in particular **ERS Elsys** [13] and **ERS CO2 Elsys** [14] sensors. They are sensor for indoor measurements, they are equipped with a temperature sensor, a humidity sensor, a passive infrared (PIR) sensors, a light sensor, and the ERS CO2 has also a CO2 sensor.
The temperature sensor operates from $0°C$ to $50°C$, with a resolution of $0.1°C$ and an accuracy of $\pm0.2°C$. The humidity sensor operates from $0\%rF$ to $85\%rF$, it has accuracy at $25°C$ of $\pm2\%rF$, and a resolution of $0.1\%rF$. The light sensor has a range from 2 LUX to 2000 LUX, it has an accuracy of ±10 LUX and a resolution of 1 LUX.
The CO2 sensor in the *ERS CO2* operates from $0ppm$ to $2000ppm$ with an accuracy of $\pm50ppm$.
The sensors use the Over-The-Air Activation method (**Section 3.1.3**) to establish a secure communication with the **Kerlink Wirnet iFemtoCell-evolution** [20] LoRaWAN indoor gateway.
The gateway has a 4G module with 3G/2G fall-back and an Ethernet connection to forward the data to the Internet. It has KerOS, as operating system with embedded GNU/Linux based on Yocto 2.4 and LTS kernel 4.14, native support for Python2, C/C++ and Shell, and it includes SQlite and lighttpd. A web interface is available

to configure the gateway.
The gateway gives us the possibility to create private LoRaWAN network, cause it is also a LoRaWAN Join/Network server, so it receives, decrypts, and decodes, the LoRaWAN packets from the sensors. Moreover, the gateway offers **Node-RED** [38] a flow-based programming tool targeted for event-driven application. We use *Node-RED* to decode the LoRaWAN packet received by the sensor, and to format the JSON object and finally we send the newly created JSON to **Elastic Search**.

7.2.2 Cloud Architecture

To manage the large amount of data **ssense** uses several **Cloud Services**, in particular **Elastic Cloud** and **Google Cloud**.

Elastic Cloud

To store and visualize data we used the **Elastic Stack** [12], a service that combine **ElasticSearch** and **Kibana**, to offer data storage and data visualization.
ElasticSearch is a distributed, JSON-based search and analytics engine, that we use to store all the data needed by *ssense* from the users information to the data collected by the sensors. **Kibana** offers a simple and flexible way to display the data in charts, grouped in dashboards that we export and embed in our web views.
The **Elastic Stack** is offered in two ways:

- **Self-Managed** that allows you to install the services in your machine, via a standalone application or using Docker containers.
- **Elastic Cloud** that permits to host the services in the cloud. It offers the possibility to deploy them on the three main cloud provider AWS, Google Cloud and Microsoft Azure.

We choose to deploy the services in the cloud, in particular on **Microsoft Azure** [30] to gives the client maximum scalability and availability.
Elastic Search has a native support for managing and authenticating users, and offers several authentication services as a Token-based and an API KEY-based authentication. In both cases is possible to define roles with different permissions, that can limit the access to the indexes or to the Kibana features. We define two roles for the *ssense* user: the **ssense-admin**, and the **ssense-user** to distinguish the two main type of users mentioned in **Section 7.1**.

Google Cloud

To authenticate the users of *ssense* we use the **Token-based authentication**, that is managed by a REST server in **Node.js**, developed using **Express Framework**

[37]. The back-end manages the login, the creation, the modification and the deletion of the users.
In addition, the server handles the retrieving of the data with respect to the permission level of the simple user, in such a way the information about the points of interest and of the sensors measures are filtered before reaching the client browser. The server, as the gateways, uses an API KEY to authenticate with Elastic Search, for security reason the key has only the permissions to access to the indexes and the services that it needs to fulfil its goal.
The server is deployed using **Google Cloud Functions** [15] a **Function As A Service** that allows to run code in a scalable way without server management.

7.2.3 Framework7 - Progressive Web App

The user interface is implemented using **Framework 7** [21], a free and open-source cross-platform framework to develop mobile, desktop or web application based on JavaScript.
We choose to implement *ssense* as a **Web Progressive App** [16]. The PWAs are web applications that gave the end-user the feeling of a platform-specific applications. This is possible thanks to the improved capabilities of the web applications nowdays, better performances offered by the PWAs with respect to the standard web application thanks to a caching system that permits the user to navigate the app even with a not reliable internet connection. Moreover, the Progressive Web App can be installed and run in stand-alone windows, and they are launchable like any other native applications.

ssense®

Chapter 8

Conclusion and Future Works

Keeping in mind that a *group key agreement protocol* must guarantee the freshness of the keys, we saw that the current specification of LoRAWAN does not guarantee that in case of the **NwkKey**, the **AppKey**, the **JSIntKey**, and the **JSEncKey**. The first two keys are hard-coded into the end-device, and the only way to change them is to hard-code a new pair of them.
In this work is proposed a new type of Rejoin-Request message that can permit to refresh those keys during the operational phase of the end-device [18], using the **Elliptic Curve Cryptography**. The ECC permits us to take advantages of the asymmetric encryption with a small overhead in terms of computational power and energy consumption, fitting particularly well in resource-constrained LoRaWAN devices.
In addition, the new method needs to add the possibility to refresh the root keys without modifying the core architecture of LoRaWAN1.1, that is why we proposed a new type of Rejoin-Request message, that can be seen as an add-on of the actual specification. In this way, the backward compatibility is maintained, and in this way we ease the implementation of our proposal. The length of both the new type of messages introduced, even if they are bigger than the other messages used by OTAA, they do not exceed the maximum payload that range from 51 to 222 bytes depending on the spreading factor, the frequencies and the bandwidth [1].
ssense is a real-world example of the potentialities of LoRaWAN and of the LPWANs in general, since it allows to create a very dense wireless sensors network with affordable economic cost and low maintenance, since LoRaWAN gateways offers a wide area of coverage and they can connect to a great number of sensors, and the LoRaWAN sensors ensures a long lifetime with standard AA batteries.
Potential future works include proceeding with the research on **LoRaWAN security**, a first step can be the implementation of the proposed protocol to test it in a real-world context, and to try to mitigate other vulnerabilities in LPWANs.

Regarding **ssense**, a first beta version is ready to be deployed in a company, to test better the architecture and the implementation, moreover new features and support for new type of sensors will be added in the future version of the application.

Bibliography

[1] Adelantado, F., Vilajosana, X., Tuset-Peiro, P., Martinez, B., Melia-Segui, J., and Watteyne, T. Understanding the limits of lorawan. *IEEE Communications magazine*, **55** (2017), 34.

[2] Adjih, C., et al. Fit iot-lab: A large scale open experimental iot testbed. In *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*, pp. 459–464. IEEE (2015).

[3] Ahirwal, R. R. and Ahke, M. Elliptic curve diffie-hellman key exchange algorithm for securing hypertext information on wide area network. *International Journal of Computer Science and Information Technologies*, **4** (2013), 363.

[4] Alliance, L. Lorawan 1.1 regional parameters. *technical specification*, (2017).

[5] Alliance, L. Lorawan 1.1 specification. *technical specification*, (2017).

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

[7] Bjoernsen, K. Koblitz curves and its practical uses in bitcoin security. *order (ε (GF (2k)*, **2** (2009), 7.

[8] Blake-Wilson, S., Bolyard, N., Gupta, V., Hawk, C., and Moeller, B. Elliptic curve cryptography (ecc) cipher suites for transport layer security (tls). Tech. rep., RFC 4492 (2006).

[9] Butun, I., Pereira, N., and Gidlund, M. Analysis of lorawan v1. 1 security. In *Proceedings of the 4th ACM MobiHoc Workshop on Experiences with the Design and Implementation of Smart Objects*, pp. 1–6 (2018).

[10] Chantzis, F., Deirme, E., Stais, I., Calderon, P., and Woods, B. *Practical IoT hacking the definitive guide to attacking the internet of things.* No Starch Press, Inc (2020).

[11] Chatzigiannakis, I., Konstantinou, E., Liagkou, V., and Spirakis, P. Design, analysis and performance evaluation of group key establishment in wireless sensor networks. *Electronic Notes in Theoretical Computer Science*, **171** (2007), 17.

[12] Elastic. Elastic stack. `https://www.elastic.co/elastic-stack`.

[13] Elsys. Ers. `https://elsys.se/public/datasheets/ERS_datasheet.pdf`.

[14] Elsys. Ers co2. `https://elsys.se/public/datasheets/ERS_CO2_datasheet.pdf`.

[15] Google. Google cloud functions. `https://cloud.google.com/functions`.

[16] Google. Progressive web app. `https://web.dev/progressive-web-apps/`.

[17] Han, J. and Wang, J. An enhanced key management scheme for lorawan. *Cryptography*, **2** (2018), 34.

[18] Heer, T., Garcia-Morchon, O., Hummen, R., Keoh, S. L., Kumar, S. S., and Wehrle, K. Security challenges in the ip-based internet of things. *Wireless Personal Communications*, **61** (2011), 527.

[19] Houria, A., Abdelkader, B. M., and Abderezzak, G. A comparison between the secp256r1 and the koblitz secp256k1 bitcoin curves. *Indonesian Journal of Electrical Engineering and Computer Science*, **13** (2019), 910.

[20] Kerlink. Wirnet ifemtocell-evolution. `https://www.kerlink.com/product/wirnet-ifemtocell-evolution/`.

[21] Kharlampidi, V. Framework7 - full featured framework for building ios, android & desktop apps. `https://framework7.io`.

[22] Kim, Y., Perrig, A., and Tsudik, G. Tree-based group key agreement. *ACM Transactions on Information and System Security (TISSEC)*, **7** (2004), 60.

[23] kmackay. micro-ecc. `https://github.com/kmackay/micro-ecc` (2020).

[24] Lauter, K. The advantages of elliptic curve cryptography for wireless security. *IEEE Wireless communications*, **11** (2004), 62.

[25] Lenstra Jr, H. W. Factoring integers with elliptic curves. *Annals of mathematics*, (1987), 649.

[26] Liao, L. Group key agreement for ad hoc networks. *IACR Cryptol. ePrint Arch.*, **2006** (2006), 6.

[27] Marais, J. M., Malekian, R., and Abu-Mahfouz, A. M. Lora and lorawan testbeds: A review. In *2017 Ieee Africon*, pp. 1496–1501. IEEE (2017).

[28] McGrew, D., Igoe, K., and Salter, M. Fundamental elliptic curve cryptography algorithms. *Internet Engineering Task Force RFC*, **6090** (2011), 1.

[29] Microchip. Atsamr21g18a - wireless modules. `https://www.microchip.com/wwwproducts/en/ATSAMR21G18A`.

[30] Microsoft. Microsoft azure. `https://azure.microsoft.com/en-us/`.

[31] Mössinger, M., Petschkuhn, B., Bauer, J., Staudemeyer, R. C., Wójcik, M., and Pöhls, H. C. Towards quantifying the cost of a secure iot: Overhead and energy consumption of ecc signatures on an arm-based device. In *2016 IEEE 17th International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, pp. 1–6. IEEE (2016).

[32] Nakov, S. Ecdh key exchange - pratical cryptography for developer. `https://cryptobook.nakov.com/asymmetric-key-ciphers/ecdh-key-exchange`.

[33] Nir, Y., Josefsson, S., and Pegourie-Gonnard, M. Elliptic curve cryptography (ecc) cipher suites for transport layer security (tls) versions 1.2 and earlier. *Internet Requests for Comments, RFC Editor, RFC*, **8422** (2018).

[34] Nordic. nrf51422 - bluetooth low energy, ant and 2.4 ghz soc. `https://www.nordicsemi.com/Products/Low-power-short-range-wireless/nRF51422`.

[35] Nordic. nrf52832 - versatile bluetooth 5.2 soc. `https://www.nordicsemi.com/Products/Low-power-short-range-wireless/nRF52832`.

[36] Nordic. nrf52840 - bluetooth 5.2 soc. `https://www.nordicsemi.com/Products/Low-power-short-range-wireless/nRF52840`.

[37] OpenJsFoundation. Express - node.js. `https://expressjs.com`.

[38] OpenJsFoundation. Node-red. `https://nodered.org`.

[39] Porambage, P., Braeken, A., Schmitt, C., Gurtov, A., Ylianttila, M., and Stiller, B. Group key establishment for enabling secure multicast communication in wireless sensor networks deployed for iot applications. *IEEE Access*, **3** (2015), 1503.

[40] Raza, U., Kulkarni, P., and Sooriyabandara, M. Low power wide area networks: An overview. *IEEE Communications Surveys & Tutorials*, **19** (2017), 855.

[41] RIOT-OS. Riot os crypto module. `https://api.riot-os.org/group__sys__crypto.html`.

[42] RIOT-OS. Riot-os. `https://github.com/RIOT-OS/RIOT` (2020).

[43] Ronen, E., Shamir, A., Weingarten, A.-O., and O'Flynn, C. Iot goes nuclear: Creating a zigbee chain reaction. In *2017 IEEE Symposium on Security and Privacy (SP)*, pp. 195–212. IEEE (2017).

[44] Sanchez-Gomez, J., Garcia-Carrillo, D., Marin-Perez, R., and Skarmeta, A. F. Secure authentication and credential establishment in narrowband iot and 5g. *Sensors*, **20** (2020), 882.

[45] Sanchez-Iborra, R., Sánchez-Gómez, J., Pérez, S., Fernández, P. J., Santa, J., Hernández-Ramos, J. L., and Skarmeta, A. F. Enhancing lorawan security through a lightweight and authenticated key management approach. *Sensors*, **18** (2018), 1833.

[46] Shaikh, J. R., Nenova, M., Iliev, G., and Valkova-Jarvis, Z. Analysis of standard elliptic curves for the implementation of elliptic curve cryptography in resource-constrained e-commerce applications. In *2017 IEEE International Conference on Microwaves, Antennas, Communications and Electronic Systems (COMCAS)*, pp. 1–4. IEEE (2017).

[47] Sicari, S., Rizzardi, A., Grieco, L. A., and Coen-Porisini, A. Security, privacy and trust in internet of things: The road ahead. *Computer networks*, **76** (2015), 146.

[48] Spirakis, P., Chatzigiannakis, I., and Liagkou, V. Providing end-to-end secure communication in low-power wide area networks (lpwans). *LNCS*, (2018).

[49] STMicroelectronics. Stm32f103re - mainstream performance line, arm cortex-m3 mcu with 512 kbytes of flash memory, 72 mhz cpu, motor control, usb and can. `https://www.st.com/en/microcontrollers-microprocessors/stm32f103re.html`.

[50] STMicroelectronics. Stm32l072cz - ultra-low-power arm cortex-m0+ mcu with 192-kbytes of flash memory, 32 mhz cpu, usb. `https://www.st.com/en/microcontrollers-microprocessors/stm32l072cz.html`.

[51] Suárez-Albela, M., Fraga-Lamas, P., and Fernández-Caramés, T. M. A practical evaluation on rsa and ecc-based cipher suites for iot high-security energy-efficient fog and mist computing devices. *Sensors*, **18** (2018), 3868.

[52] TTI. Regional parameters. `https://www.thethingsnetwork.org/docs/lorawan/regional-parameters.html`.

[53] Wu, Y., Guo, G., Tian, G., and Liu, W. A model with leaf area index and trunk diameter for lorawan radio propagation in eastern china mixed forest. *Journal of Sensors*, **2020** (2020).

[54] Wuille, P. Bip32: Hierarchical deterministic wallets. *https:,//github.com/bitcoin/bips/blob/master/bip-0032. mediawiki*, (2012).

[55] Xing, J., Hou, L., Zhang, K., and Zheng, K. An improved secure key management scheme for lora system. In *2019 IEEE 19th International Conference on Communication Technology (ICCT)*, pp. 296–301. IEEE (2019).

[56] Yang, X. Lorawan: vulnerability analysis and practical exploitation. *Delft University of Technology. Master of Science*, (2017).