

Salamanca, Gto., a 21 de Junio del 2021.



M. en I. HERIBERTO GUTIÉRREZ MARTIN
JEFE DE LA UNIDAD DE ADMINISTRACIÓN ESCOLAR
PRESENTE.-

Por medio de la presente, se otorga autorización para proceder a los trámites de impresión, empastado de tesis y titulación al alumno(a) Septián Hernández José Antonio del **Programa de Maestría en** Maestría en Ingeniería Eléctrica y cuyo número de **NUA** es: 800562 del cual soy director. El título de la tesis es: Cifrados Asimétricos Ligeros Post Cuánticos para Redes Inalámbricas de Sensores.

Hago constar que he revisado dicho trabajo y he tenido comunicación con los sinodales asignados para la revisión de la tesis, por lo que no hay impedimento alguno para fijar la fecha de examen de titulación.

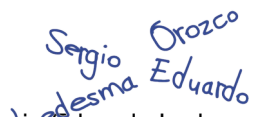
ATENTAMENTE



Dr. Juan Pablo Ignacio Ramírez Paredes

NOMBRE Y FIRMA
DIRECTOR DE TESIS
SECRETARIO

NOMBRE Y FIRMA
DIRECTOR DE TESIS


Dr. Sergio Eduardo Ledesma Orozco

NOMBRE Y FIRMA
PRESIDENTE


Dr. Juan Carlos Gómez Carranza
NOMBRE Y FIRMA
VOCAL

Llenar en computadora con ayuda del oficio de modalidad.

La modalidad de tesis es única para los posgrados

Nivel:	
Licenciatura	
Maestría	X
Doctorado	

Modalidad:	Tesis
-------------------	-------

Año:	2021
-------------	------

Marcar con una X

Poner el número de año p.e. 2015

Información sobre Obtención de Grado Académico:

Nombre	Septién Hernández José Antonio
NUA	800562
Programa	Maestría en Ingeniería Eléctrica (Instrumentación y Sistemas Digitales)

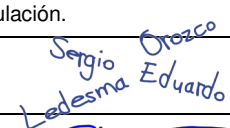
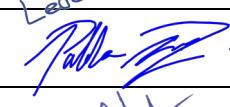

Para modalidades con Jurado completar la siguiente información:

Lugar, hora y fecha de la presentación

Lugar	Salamanca	Firma y sello de autorización de reservación de lugar.
Hora	16:00 hrs.	
Fecha	5 de Julio de 2021	

Título del trabajo	Cifrados Asimétricos Ligeros Post Cuánticos para Redes Inalámbricas de Sensores
---------------------------	---------------------------------------------------------------------------------

Jurado

	Nombre con grado académico completo: p.e. Doctor en Informática Industrial Nombre Apellido Paterno Apellido Materno	Firma de autorización para realización de examen de grado o titulación.
Presidente	Doctor en Filosofía Sergio Eduardo Ledesma Orozco	
Secretario	Doctor en Ingeniería Eléctrica Juan Pablo Ignacio Ramírez Paredes	
Vocal (1)	Doctor en Ciencias de la Computación Juan Carlos Gómez Carranza	
Vocal 2 (Doctorado)		
Vocal 3 (Doctorado)		

Asesoría

Director del trabajo	Doctor Juan Pablo Ignacio Ramírez Paredes
Codirector	Doctor Marco Antonio Contreras Cruz

(No llenar para uso exclusivo de la Coordinación.)

Valida (nombre y firma): _____

Una vez terminado de llenar imprimir en dos tantos (uno para entregar al iniciar el trámite de autorización del examen de grado o titulación y otro para firma de recibido).



UNIVERSIDAD DE GUANAJUATO

**CAMPUS IRAPUATO - SALAMANCA
DIVISIÓN DE INGENIERÍAS**

**Cifrados Asimétricos Ligeros Post Cuánticos
para Redes Inalámbricas de Sensores.**

TESIS PROFESIONAL

**QUE PARA OBTENER EL GRADO DE:
Maestría en Ingeniería Eléctrica (Instrumentación y Sistemas Digitales)**

PRESENTA:

Ing. Septién Hernández José Antonio

DIRECTORES:

Dr. Ramírez Paredes Juan Pablo Ignacio

Dr. Contreras Cruz Marco Antonio



UNIVERSIDAD DE GUANAJUATO

**CAMPUS IRAPUATO - SALAMANCA
ENGINEERING DIVISION**

Lightweight Post-Quantum Asymmetric Ciphers
for Wireless Sensor Networks

PROFESSIONAL THESIS

TO OBTAIN THE DEGREE OF:

Master in Electrical Engineering (Instrumentation and Digital Systems)

PRESENTS:

BEng. Septién Hernández José Antonio

SUPERVISORS:

Dr. Ramírez Paredes Juan Pablo Ignacio

Dr. Contreras Cruz Marco Antonio

Dedicatoria

A mi abuela Juanita. Ya no estás físicamente con nosotros, pero el amor que nos profesaste siempre nos acompañará donde sea que nos encontremos.

A mi padrino Gerardo. Gracias por todo tu apoyo y cariño que me diste desde niño.

A mi abuelo José Antonio. Por la ayuda que, a tu manera, le diste a mi papá.

D.E.P.



Al doctor Víctor Ayala.

The Road Not Taken

*Two roads diverged in a yellow wood,
And sorry I could not travel both
And be one traveler, long I stood
And looked down one as far as I could
To where it bent in the undergrowth;
Then took the other, as just as fair,
And having perhaps a better claim,
Because it was grassy and wanted wear;
Though as for that the passing there
Had worn them really about the same,
And both that morning equally lay
In leaves no step had trodden black
Oh, I kept the first for another day!
Yet knowing how way leads on to way,
I doubted if I should ever come back.
I shall be telling this with a sigh
Somewhere ages and ages hence:
Two roads diverged in a wood, and I—
I took the one less traveled by,
And that has made all the difference.*

Robert Frost.

Agradecimientos

A mis asesores, el doctor Juan Pablo y el doctor Marco, por la guía que me han dado. Por presentarme un tema del cual conocía muy poco, y dejarme explorarlo a mis anchas. Pude seguir mi personalidad curiosa y pasar de apenas conocer el campo, a tener unos resultados concretos.

A la doctora Magali del Infotec, por recibirme para realizar estancias de investigación. Se quedó abierta la puerta para futuras colaboraciones entre el laboratorio y el Infotec.

A mis papás, Laura y Toño, por todo el apoyo que me han dado desde siempre. En especial por el apoyo recibido durante esta pandemia que tan rara ha sido para mí; tantos cambios y tan profundos.

A mis hermanas, Verónica, Ana Laura y Andrea por estar ahí siempre que las necesite y por el apoyo que me dan. Por acompañarme en este proceso llamado vida, y prestarme ayuda siempre que la necesité. Espero que podamos seguir compartiendo momentos y experiencias por mucho tiempo más.

A mi abuelo, por los consejos dados, por la guía que me das, todas las experiencias vividas, por la ayuda y el apoyo que me diste cuando lo necesité. Pasaste por momentos muy difíciles, y eso nos mostró tu fortaleza para sobrellevar las cosas. Estuve ahí para tí en ese momento tan difícil. Estaré ahí para apoyarte y sostenerte siempre que lo necesites.

A mi primo Gerardo, aunque convivimos menos durante este periodo, siempre estuve ahí cuando quería platicar o convivir contigo. Espero que el apoyo que te he dado haya sido suficiente para tí durante esos momentos que tan difíciles fueron para tí. Siempre vas a tener un amigo y confidente conmigo.

A toda mi familia, tíos y primos. Con algunos nos mantuvimos unidos durante los tiempos difíciles que pasaron, con otros nos separamos. Cada situación me enseñó otras formas de ver y vivir, y como manejarlas. Gracias a todos los que nos mostraron apoyo cuando lo necesitamos.

A mis compañeros y amigos de maestría, Gustavo, Fernando y Juan, por acompañarme durante la maestría y ayudarme a resolver dudas cuando las tenía.

A todos mis amigos y compañeros que conocí durante mi tiempo que estuve realizando la maestría, cada experiencia que tuve con ustedes, mala o buena, me mostró una parte de mí.

A mi psicóloga Marcela, por ayudarme y guiarme en este arduo trabajo de autoconocerme, sanar viejas heridas, y en guiarme a través de esas sombras llamadas miedo. Por ayudarme a conocer esta característica de mí que no entendía, pude darle un nombre, y lo que es más, entender este rasgo de mí que pocos parecen ver, aún menos entender, y que yo no podía explicarme (de nombre *Sensory-Processing Sensitivity*).

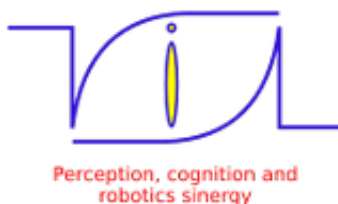
Agradecimientos Institucionales

A la Universidad de Guanajuato, a través de la División de Ingenierías del Campus Irapuato-Salamanca, por proporcionarme instalaciones y material para mi desarrollo integral como estudiante de maestría.



UNIVERSIDAD DE
GUANAJUATO

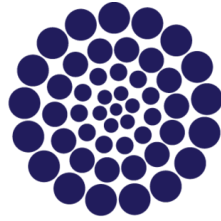
Al Laboratorio de Visión, Robótica e Inteligencia Artificial por proporcionarme un espacio de trabajo para el desarrollo de mi tesis.



Al Infotec unidad Aguascalientes por haberme recibido para realizar estancias de investigación.



Al Consejo Nacional de Ciencia y Tecnología (CONACyT), por el apoyo financiero provisto durante la realización de la maestría, a través de la beca 936284.



CONACYT

Consejo Nacional de Ciencia y Tecnología

Contents

1	Introduction	1
2	Background	3
2.1	The Internet of Things	3
2.1.1	IoT communication protocols at the application layer	5
2.1.2	Basic IoT devices at the object layer	6
2.1.3	Security for the IoT	7
2.2	Cryptography	8
2.3	Public-key cryptography and key exchange mechanism	8
2.3.1	Encryption security	10
2.4	Post-quantum cryptography	11
2.4.1	Quantum computers and the Shor’s algorithm	11
2.4.2	Existing post-quantum cryptosystems	12
2.5	Post-quantum cryptosystem standards	13
2.5.1	Currently competing cryptosystems	14
2.6	The transport layer security protocol	15
2.6.1	TLS handshake protocol	15
2.6.2	Crypto libraries implementing the TLS protocol	17
3	Methodology	18
3.1	Introducing post-quantum cryptosystems to IoT	18
3.2	Post-quantum cryptosystems suitable for IoT	19
3.2.1	Lattice-based NIST post-quantum cryptosystems	19
3.2.2	Algorithms specification	19
3.2.3	NIST’s post-quantum cryptosystems suitable to IoT devices	28
3.3	The IoT prototype	28
3.3.1	The proposed system	28
3.3.2	The nodes	29
3.3.3	The gateway	30
3.3.4	The broker	31
3.4	Measuring the cryptosystems performance	32
3.4.1	Variables of interest	32
3.4.2	Profiling the variables	32
3.4.3	Data exploration and tests execution	34
3.5	Postamble	35

4	Results	36
4.1	Measuring the cryptosystems' performance	36
4.1.1	Memory and CPU Performance	36
4.1.2	Selecting the first three KEMs	39
4.2	Testing the mechanisms on the IoT prototype	39
4.2.1	Performance on the number of packets	40
4.2.2	Performance on packet size	40
4.2.3	Performance on connection's duration.	43
4.3	Guidelines on selecting post-quantum KEM	46
5	Conclusions	51
5.1	Future Work	52
	Appendices	53
A	Basic Mathematics for Cryptography	54
A.1	Groups	54
A.2	Integer Rings	55
A.3	Fields	55
A.4	Lattices	55
B	List of acronyms	57

Resumen

El Internet de las Cosas es el siguiente paso en la evolución del Internet, en el cual se pretenden conectar una gran variedad de dispositivos a la Internet con el objetivo de recabar datos del entorno que faciliten la toma de decisiones.

Al igual que todos los sistemas informáticos, el Internet de las Cosas enfrenta amenazas externas que deben ser consideradas desde las primeras etapas de diseño del dispositivo, considerando además que los dispositivos usados tienen una cantidad limitada de recursos.

Un requisito necesario para el Internet de las Cosas, es asegurar que los datos transmitidos por la Internet estén protegidos ante posibles intervenciones externas no autorizadas, en especial ante intervenciones por Computadoras Cuánticas, hardware que es posible que llegue en los próximos años.

En este trabajo se considerará el impacto en rendimiento y uso de recursos que pueden tener los distintos criptosistemas post cuánticos en dispositivos con recursos restringidos, a la vez que se dan guías de selección para usar los criptosistemas en dispositivos con recursos restringidos.

Abstract

The Internet of Things is the next step in the evolution of the Internet, at which it is pretended to connect a wide variety of devices to the Internet with the objective of gathering data from the environment, with the objective of assisting in the decision-making process.

As with every information system, the Internet of Things faces external threats which most be considered from the early stages of device design, considering as well that such devices are usually resource-constrained in nature.

One such threat is securing the data transmitted through the Internet, impeding third parties to discover its contents. One such special threat are the quantum computers, hardware that may soon be widely available in the near future.

In this work, we begin considering the impact on performance and resource consumption that post-quantum cryptosystems might have on devices with low resources, and provide some guidelines for appropriately selecting a suitable one for such hardware.

Chapter 1

Introduction

The Internet of Things (IoT) is the next step in the evolution of the Internet, in which all types of devices will be able to connect to it. These devices will not necessarily have human supervision, so they will have to operate autonomously. The deployment of such devices will create an ecosystem that should be resilient against several external perturbations and threats. One such external threat is securing the communication between the different endpoints to protect unauthorized third parties from intervening and access the transmitted data.

An emerging field in Computer Science and Physics is Quantum Computation. In 1985, David Deutch proposed the first Universal Quantum Computer model by expanding the Church-Turing hypothesis to a physical principle [1]. Later, in 1994, the mathematician Peter Shor proposed an algorithm that uses some of the properties of quantum computers to solve the discrete-logarithm and integer factorization problems. The algorithm poses a threat to classical cryptosystems that use such problems for guaranteeing security. As the development of quantum computers made its realization more feasible, the need to create a new standard for the post-quantum era emerged. So, the US National Institute of Standards (NIST) began creating such a new standard.

When quantum computers come to existence (if ever), the IoT endpoints, build up from devices with very limited computation, communications, and energy resources, known as resource-constrained devices, will also be threatened by this new hardware. So, it is necessary to start adopting such standards to the IoT endpoints, especially the public-key cryptosystems. Symmetric-key algorithms are not as threatened by this new hardware, since doubling the current key sizes would be enough to protect them.

Most of the current efforts on post-quantum cryptography have been in developing cryptosystems that are resistant against quantum adversaries. There is little attention to the development or adaptation of post-quantum cryptosystems to resources-constrained devices.

In this work, we enable post-quantum security for the IoT by studying how the new NIST standards might impact on resource-constrained devices, especially from the computation and communication standpoint of view. From this study, we can obtain insights on how the cryptosystems perform on such devices and then give some guidelines for using and selecting an appropriate one. We study those cryptosystems available at the NIST standardization process. We do this by first determining, according to the literature, which

type of post-quantum cryptosystems is most suitable to resource-constrained devices. We then select a popular crypto library to use with the cryptosystem for key exchange, select a frequently used IoT protocol that uses such library, and finally develop a prototype IoT system to test in a real-world scenario.

We briefly present here work related to post-quantum cryptosystems for the IoT. There has been little work regarding post-quantum cryptosystems for IoT devices. On [2], the authors propose an adaptation of a key management scheme known as Identity-Based Encryption (IBE) to post-quantum cryptosystems. Specifically, they adapt a lattice-based one to the IBE scheme. In [3], the authors propose a compression and error correction algorithm for ciphertexts generated by lattice-based cryptosystems. The method the authors proposed reduces the ciphertext size without losing information, improving the performance of the decryption process. In [4], the author presents a survey of current efforts to create post-quantum cryptosystems, including institutions and universities efforts and different standardization initiatives currently in existence. The most significant initiatives are those by the European Telecommunication Standards Institute (ETSI), the Internet Engineering Task Force (IETF), and the National Institute of Standards (NIST). The author presents the different types of post-quantum cryptosystems from the NIST's second round, compares them, gives the keys' sizes, claimed quantum security, and claimed classical security. Fernández-Caramés also shows the CPU performance of various cryptosystems on a variety of platforms used for IoT devices.

This work is organized as follows. In chapter 2, we begin by introducing some basic concepts on the IoT and cryptography. We start by introducing the IoT, its essential components, architectural design, and the two most basic IoT devices. We then proceed to introduce cryptography and give a basic definition of cryptography, its two basic types, and then introduce public-key cryptography. We also briefly introduce quantum computers, post-quantum cryptography, and why we need them. We then introduce the standardization process, and finally, the Transport Layer Security protocol.

In chapter 3, we begin by introducing the suitable post-quantum cryptosystems for IoT devices, present the algorithm specification for lattice-based cryptosystems from the NIST, and give the versions used in the study. We then introduce the proposed IoT prototype used for the tests, its hardware and software components, and how they interact among each other. Finally, we present the variables to study the performance of the cryptosystem, how we present the data, and how we perform the tests.

Chapter 4 presents the results of testing the different available ciphers' performance on resource-constrained devices, first in an independent manner and then integrated with a crypto library. We also evaluate the ciphers on a real-world application, in which a server communicates directly with an IoT device. We then give insights into the ciphers' usage on resource-constrained devices. Finally, in chapter 5, we present the conclusions and give some possible future directions for continuing the work's development. In appendix A we present some fundamental mathematical tools for the construction of cryptosystem, and in appendix B we present a list of acronyms used throughout the text.

Chapter 2

Background

This chapter presents the basics for using and understanding both IoT devices and the post-quantum cryptosystems. We begin by introducing the concept of the Internet of Things by giving some definition, introducing its comprising elements, its architectural design, some IoT protocols, and the two basic types of devices for the IoT. We then present some desirable security attributes for it.

We briefly introduce the field known as cryptography, present its definition, and its two types. We then present public-key cryptography and key exchange mechanisms with their primitive components. We present the desirable security attributes for the public-key cryptosystems.

Immediately after that, we introduce post-quantum cryptography by indicating why it is needed, how it came to existence, its two main types, and the main representatives for each type of cryptosystem. We then introduce the process for creating a new standard for the post-quantum era and the contenders. The chapter ends by introducing the TLS protocol, which will use the post-quantum cryptosystems to secure its communications.

2.1 The Internet of Things

The Internet of Things, or IoT, is a new paradigm considered to be the next step in the evolution of the Internet, at which it is intended to connect a great variety of devices. This new paradigm will create an ecosystem that will facilitate data gathering from the environment, its modification, and will assist in the decision-making process through data analysis.

The academia and industry have not yet reached a consensus on the definition of the IoT. There have been several attempts to define the IoT, focusing mainly on three aspects: "things-oriented" definition, centered around the objects that will sense and modify the environment; "Internet-oriented" based on the networking infrastructure; and finally, a "semantic-oriented" based on the set of technologies required to represent, store, search, interconnect, and organize the data generated by the objects.

A definition provided by [5] and which tries to encompass the three above aspects, is the following: *"A scalable heterogeneous global network of augmented devices with self-organizing capabilities; an infrastructural network adaptable to existing as well as*

future enabling technologies; that behaves as a multi-agent system, with agents having cognizance towards the cyber-bio-physical environment; and interacts as a social network using smart interfaces to achieve an objective or set of objectives”.

The definition tells us that the IoT is composed of objects deployed to the physical world, like sensors, actuators, RFID tags, all the way to cloud servers, and, to achieve the desired goal, which is data-based decision-making, all the components should work in harmony. It also tells us that the IoT should be an ecosystem as autonomous as possible and be capable of responding and adapting to change with little or no human intervention.

In order for the IoT ecosystem to work properly and achieve its desired goals, five stages, as shown in Figure 2.1, need to exist [5], [6], [7]: the first one is known as *Sensing*, responsible from gathering the data for the environment and transferring to other platforms and devices. The second stage is known as *Communications*, which is the set of technologies required for connecting the different devices and the transfer of data. The third stage is *Computation*, which is the fusion of processing units with software that provides the IoT with computation capabilities. The next stage is *Services*, or all the functionality required for the proper functioning of the ecosystem, like aggregation services, collaborative services, among others. The final stage is *Semantics*, that provides context to the data gathered with the objects. These stages define a possible path that data could travel through, from the environment to the final user, to meet its needs and goals.



Figure 2.1: Stages for data gathering, transmission, and interpretation in the context of the Internet of Things.

It is also necessary an architecture that allows us to deploy the devices in an ordered fashion [5], [6]. There have been several proposals by the community, with different number of layers and functions, with the most accepted one consisting of five layers, as shown in Figure 2.2. The first layer, called *Object Layer*, consists of all the objects and its components required for the gathering of data. The second layer, called *Object Abstraction Layer*, and provides means of interacting with them. The third layer is called *Service Management Layer*, which handles all the services functioning on the ecosystem, and connects the upper layers with the lower ones. The next layer is called *Application Layer*, the one accessible to the final user. This layer presents all the information to the user, and allows it to interact with the ecosystem so it performs the tasks the user requires. The final layer, called *Business Layer*, handles all the subsystems and components involved to assure a proper working of the entire ecosystem.

This architecture represents a reference one, and depending on the application, it may or not implement the whole architecture. The objects that comprise the object layer, can vary in a wide range depending on the purpose of the application. There also exists several



Figure 2.2: Common architectural design for the IoT ecosystem, composed of five layers.

protocols at the different layers of the architecture, especially on the application layer. In the next section we introduce some commonly used protocols existing at the application layer.

2.1.1 IoT communication protocols at the application layer

Several protocols exist in the Application Layer for delivering the required services to the end-user. The most popular ones are the following:

- **Constrained Application Protocol (CoAP):** It is targeted for devices that use the UDP communication protocol. CoAP is a web transfer protocol based on REST (Representational State Transfer) and allows a simple way of exchanging data between it and the HTTP protocol. It is designed for devices with low power, computation, and communication capabilities.
- **Message Queue Telemetry Transport (MQTT):** Protocol built on top of the TCP protocol, MQTT aims at devices with unreliable or weak links. It uses the publish/-subscribe pattern, with a broker to which all the other devices are connected and send the messages. The broker's function is to collect all the clients' information and send them the requested information. The devices have three types: the broker, a publisher, sending data to the broker, and a subscriber, which requests the broker's data.
- **Extensible Messaging and Presence Protocol (XMPP):** Design for multimedia data and instant messaging, allows the user to communicate with each other by sending

messages over the Internet regardless of the operating system they use. It operates in a decentralized fashion and connects a client to a server using an XML stanza.

- **Advanced Message Query Protocol (AMQP):** An IoT protocol focused on message-oriented environments and operated over the TCP protocol. Communications are handled with two components: exchanges and messages queue. Exchanges route the messages to the appropriate queues.
- **Data Distribution Service (DDS):** A publish-subscribe protocol for real-time machine to machine communications. It does not use a central broker or server; rather, it uses multicasting, providing excellent Quality of Service and reliability to the applications.

Protocols like the XMPP have built-in security properties and can be extended to use with other protocols. Other protocols like MQTT and CoAP rely on external crypto libraries to provide security.

In the next section we briefly introduce the two basic objects that comprise the IoT and give a brief description of each.

2.1.2 Basic IoT devices at the object layer

At the Object Layer resides all the devices that compose the "things" part of the IoT. This layer transmits the sensor-generated data and sends actuating commands to the objects. The objects can have a wide range of types, from tiny motes¹ with limited resources to complete microprocessors with operating systems and a wide range of functionalities.

Of the available objects, two crucial technologies enable the IoT to achieve the goal of accessing the data and information anywhere, anytime: the Radio Frequency Identifier (RFID) and sensor networks [7].

The Radio Frequency Identifier (RFID) technology [8] has the purpose of identifying objects via a tag. It has three components: an RFID tag, an RFID reader, and a server. The reader function is to transmit a radio signal to the card for gathering the object information, which then sends to the server.

Sensor Networks

The other crucial technology for the IoT is a Sensor Network, specifically a Wireless Sensor Network (WSN). A WSN is a set of nodes capable of processing and sensing data while having wireless communication capabilities. A sensor node is a low-powered, low-cost, resource-constrained device with an integrated computational unit, whose function is to carry out simple computations and transmits the data [9].

A WSN has three main components as shown in Figure 2.3, a sensor field, in which the sensor nodes are scattered and from which data is collected; a sink or gateway that collects the data from the sensor field and processes it in different ways accordingly to

¹A mote, short for Remote, is a wireless transceiver that also acts as a remote sensor. They have minimal capabilities and can be down to the size of a small coin.

the desired functionality; and a Task Manager Node or server, which indicates the Sink which function to perform and presents the data to the user.

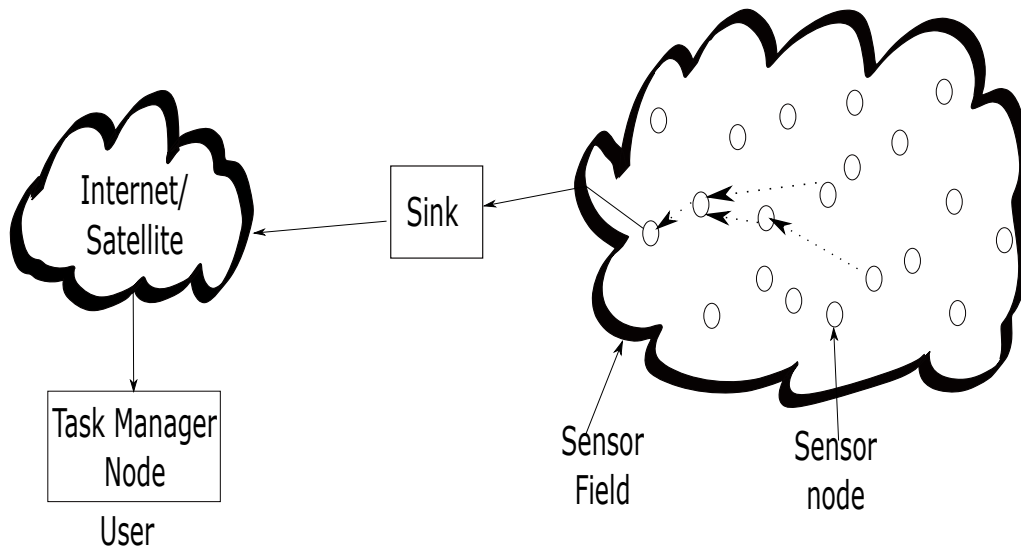


Figure 2.3: Components of a WSN. From the right, a cloud of sensors is deployed on the field. The collected data is then sent to the Sink, of which several could exist. The Sink sends the required data through the Internet or Satellite connection to the Task Manager Node, controlled by a user. The user indicates to the server what function to perform. [9]

Most applications emerge from these two IoT devices, although some applications make no use of such technologies. A typical and sometimes mandatory requirement for IoT applications is security, or to protect the devices from external threats at different architectural design levels. Next, we define several attributes desirable for IoT applications and devices.

2.1.3 Security for the IoT

Several security requirements exist to consider a device or an application to be secured. The essential attributes, known as the CIA-triad, defines in a high-level form the security requirements a device or application should comply to be considered secure [10], [11].

- **Confidentiality:** Only the users in possession of the corresponding keys can view the message's contents.
- **Integrity:** Allows the user to notice when the message was modified by a third-party during transmission, and thus preventing tampering on the message.
- **Authenticity:** Signing of the message using the public key or with the use of hash functions.

Other attributes exist to provide a higher level of trust in the application, device, or system. They bring to the system means for tracing any possible malfunction or unautho-

rized intervention. The additional attributes are accountability, auditability, trustworthiness, non-repudiation, and privacy. A way of satisfying these requirements, especially for communication links, is cryptography, which we introduce in the next section.

2.2 Cryptography

Since the beginning of human existence, there has been a need to exchange information among pairs for different purposes. However, some of the information exchanged is considered secret, so the need to protect it from unauthorized third-parties emerged, and with that, cryptography.

Cryptography can be regarded as the science and art of hiding information, so only those with granted access can know its contents. Several solutions have existed and continue to exist. Yet, with the advent of computers and telecommunication means, the problem of hiding information extended to the realm of computers. For solving these problems, cryptography was adapted to computers via encryption algorithms and schemes. Encryption is defined as the process of information hiding by using a physical law, in the case of non-telecommunication means, or a mathematical law, in telecommunication.

There exist two types of encryption processes:

- Symmetric-key encryption: It uses the same key for both encryption and decryption of the message.
- Asymmetric-key encryption: there is a key for encrypting the message and another key for decrypting it.

Traditionally, for the case of symmetric-key encryption, the keys used for the encryption and decryption process are generated with pseudo-random number generators and then distributed in a secure manner, which can be through physical means, through a trusted third party, or with the use of previous existing symmetric keys.

However, as the number of computers connected to the network increased, it becomes less and less feasible to use traditional key exchange mechanisms to have a shared key among all of them. It thus became necessary to create another mechanism by which to interchange keys without the necessity for physical transport or a secure channel. It is now that public-key cryptography emerges as a solution to it.

2.3 Public-key cryptography and key exchange mechanism

Classical-key distribution techniques, such as physical key distribution and the usage of previous symmetric keys, were becoming insufficient given the continuous growth of connections and existing computers. With that problem in mind, in 1976, the researchers Whitfield Diffie and Martin Hellman proposed a solution that later became the de-facto one to the key-distribution problem: the public-key cryptography and the Diffie-Hellman key exchange mechanism [12].

The authors propose solving the exchange of keys over an insecure channel [12] by using problems considered to be NP or computationally hard. An NP problem has non-polynomial-bounded solution, either in time, memory, or both, that solves it. It is said that a computationally-hard problem is either NP, or the time or memory it takes to be solved is impractical with current technology.

The idea of using these problems is to find a function for which computing it in the forward fashion is easy (can be done with current technology), but computing the inverse falls within the NP or hard problems realm. Such functions are called one-way functions.

Public-key Cryptography (PKC) proposes using a set of two keys: a public key, which is accessible to all users, and a private key, available only to the user that generated the keys. A typical PKC consists of four associated algorithms:

- *Setup*(1^λ): Receives as input a security parameter λ , which is then used to generate all the parameters p necessary to instantiate the problem on which the cryptosystem is based, and then it uses those parameters to generate the keys.
- *KeyGeneration*(p): Receives as input the instantiated parameters from the previous algorithm, and generates the associated public key pk and private key sk . It returns as output such keys.
- *Encryption*(m, pk): Receives as input a plain text or message m and the public key pk , and generates a ciphertext c by encrypting the message using the associated problem to the cryptosystem.
- *Decryption*(c, sk): Receives as input a ciphertext c and the private key sk , and retrieves the plaintext m .

The *Setup* and the *KeyGeneration* algorithms are executed when the communication channel is firstly open; the other two algorithms are used along with all the duration of the communication.

With the usage of an NP or hard problem and the four algorithms previously mentioned, a public-key or asymmetric-key cryptosystem can be built, which then can be used for exchange of information, exchange of symmetric keys, and for digital signatures.

The other proposed solution, whose only function is exchanging keys, is known as a Key Exchange Mechanism (KEM). Both users want to communicate, have access to a public set of parameters, from which a shared secret is generated and used for the exchange of keys and information. A typical KEM consists of a tuple of algorithms (Key-Gen, Encapsulation, Decapsulation), along with a finite keyspace K .

- *KeyGeneration*: A probabilistic key generation algorithm that output a public key pk and a private key sk .
- *Encapsulation*(pk): A probabilistic encapsulation algorithm that takes as input a public key pk , and output an encapsulation c , sometimes called ciphertext, and a shared secret $ss \in K$.
- *Decapsulation*(c, sk): A (usually deterministic) decapsulation algorithm that takes as input an encapsulation c and a secret key sk , and output a shared secret ss' .

A common technique for constructing a KEM is using a public-key cryptosystem and then applying it to a cryptographic transformation. This way, the cryptosystems designers' can create a KEM using a public-key cryptosystem, so it is unnecessary to construct a new KEM each time a new one is needed.

The first proposed solution is known as the Diffie-Hellman Key Exchange [12], which uses the discrete-logarithm problem for guaranteeing security. Another solution is known as the Rivest, Shamir, Adleman (RSA) cryptosystem, which uses the large-integer factorization problem [13].

Having a way to construct public-key cryptosystems and key exchange mechanisms, it is necessary to have a formal way of indicating whether it provides security or not, and at what level. In the next section we introduce some basic concepts for indicating the security of a cryptosystem and key exchange mechanism.

2.3.1 Encryption security

To provide a way of indicating the security level of the cryptosystems and KEMs, and how resilient they are to different attacks, the community defined several concepts, and desirable security properties [14].

Firstly, a cryptosystem is broken via an *attack*, which is performed by an *adversary*. An adversary is a randomized polynomial-time algorithm interacting with the cryptosystem in some way. How the adversary interacts with the cryptosystem is known as the *attack model*. The attacker also has an *attack goal*.

The most severe attack model for a cryptosystem is known as *total break*, in which the adversary computes the private key. The following properties, known as *security properties*, describe the way a cryptosystem protects itself from external attacks.

- One-way encryption (OWE): The adversary cannot compute the message m from a given ciphertext c .
- Semantic Security: The adversary cannot learn information about a message from ciphertext c , besides possibly the length of the message.
- Indistinguishability (IND): Given the encryption c of any of the two messages m_0 and m_1 of the same length, the adversary cannot distinguish from which message c came.

The following are security features that indicate the security level of the cryptosystem, and types of attacks that it should withstand.

- Passive Attack/Chosen Plain Text Attack (CPA): The adversary only has access to the public key.
- Lunchtime attack/Chosen Ciphertext Attack (CCA1): The adversary has access to the public key and can ask for the decryption of the ciphertext of its choosing during the first stage of the attack.

- Adaptive Chosen Ciphertext Attack (CCA2): The adversary has access to the public key and to an oracle that can decrypt any ciphertext of the attacker's choosing. The only constraint to the oracle is that it returns \perp when passed the challenge ciphertext from the second stage.

The security settings, or theoretical strength, of any cryptosystem is determined by first establishing the desired security property (e.g. IND), and then determining the type of attack it should resist (e.g. CPA or CCA2) according to the goals of the cryptosystem. There should be as well a formal proof that the cryptosystem indeed complies with such settings. The highest security setting for any cryptosystem is the IND-CCA2, as it resists any real-world attack, without considering those based on physical means. These attributes apply to all cryptosystems, as changing an adversary from a *classical* adversary to a *quantum* one is (theoretically) easy.

2.4 Post-quantum cryptography

With the arrival of quantum computers and the Shor's algorithm, a need for public-key cryptosystems that are quantum-safe emerged.

2.4.1 Quantum computers and the Shor's algorithm

The main limiting factor of a classical computer is its discrete nature; that is, it is defined only on a finite number of states. This discrete-nature imposes a barrier to the solutions it can solve. Some can be expensive in resources (time, memory, speed) or fall outside the problems it can solve (e.g. accurate physical simulations).

On the other hand, a Quantum Computer is a physical device whose construction and definition uses physical laws and, as such, has a continuous nature. This new construction allows expanding the problems that can be efficiently solved, although there are some still outside its reach.

In [1], David Deutch introduces the definition of a Universal Quantum Computer by expanding the Church-Turing hypothesis to a physical principle, named the Church-Turing principle, allowing this way the Quantum Computer to be physically realizable.

David Deutsch presents a Quantum Computer with several properties that are not present in classical computers. One of such properties is its capability to perform parallel processing in a limited fashion. A single processor from the machine can compute several tasks in parallel, contrary to classical processors, which can only perform one function at a time. This property allowed the mathematician Peter Shor to introduce in 1994 an algorithm that ultimately solves the discrete-logarithm and large-integers factorization problems [15].

It starts by generating a random number n . If n is a factor of N , we stop, as the algorithm found the answer. If not, we proceed with the following. Using the first property, and the randomly generated number, we can get the factors as follows: set $g^p = mB + 1 \leftarrow g^p - 1 = mB$, from which we have that $(g^{p/2} - 1)(g^{p/2} + 1) = mB$, then use the Euclidean algorithm to compute the GCD between N and both $g^{p/2} - 1$, $g^{p/2} + 1$. So our task is now to find p , for which we make use of the Quantum Computer.

Its first function is to compute g^x for all possible $x \in \mathbb{Z}$, keeping track of the computation by storing both x and g^x . For a fixed m , we compute the difference between all the g^x and $m * N$ to obtain the residue r . We save both x and r . We then measure $r = 1$ to make the other answers cancel each other out, getting thus a series of numbers of the form: $a_1, r + a_2, r + a_3 \dots$

Here, we use property number 2; all the numbers are $a_i = a^{x_i+p}$ for some x_i . This series has a period p , or frequency $f = 1/p$. At this point, we cannot measure the Quantum Computer's output directly, as this would give us a random number of the form mentioned above, and from which we cannot obtain p .

We obtain p by applying a Quantum Fourier Transform, which gives us the desired frequency and the desired period p . This algorithm can be extended to the discrete logarithm problem by using the following function: $f : Z_p \times Z_{p-1} \rightarrow G, f(a, b) = g^a x^{-b}$ and setting $(a, b) = (r, 1)$.

2.4.2 Existing post-quantum cryptosystems

The introduction of Shor's algorithm and the eventual development of Quantum Computers pose a threat to cryptosystems based on the discrete logarithm and the integer factoring problem. There is a need to develop new cryptosystems for which there is no adaptation of Shor's algorithm. Such cryptosystems are known as post-quantum cryptosystems [16].

Current post-quantum cryptosystems are based on two types of mathematical constructions: based on codes and information theory, and based on lattices and polynomial algebra. Each type has two main representatives from which all other proposals are derived. For the cryptosystems based on information theory, the main representative is known as *The McEliece Cryptosystem* [17], which uses the difficulty of recovering a message in the presence of t errors as the security problem.

For lattice-based cryptosystems, the main representative is known as *The NTRU cryptosystem*, which we describe in the following section.

The NTRU cryptosystem

In the 1990s, Hoffstein, Pipher, and Silverman introduced a cryptosystem that later proved resistant to Quantum Computers and was named NTRU [18].

It is based on polynomial algebra and the reduction modulo of two integer numbers p and q , and its security is based on two problems:

- The ability to mix polynomials independently of the module.
- The difficulty of finding extremely short vectors on a lattice (the shortest vector problem or SVP).

For the cryptosystem to work, it is necessary to have the following parameters: three integers (N, p, q) , with $\gcd(p, q) = 1$, and $q \gg p$; four sets $\mathcal{L}_f, \mathcal{L}_g, \mathcal{L}_\Phi, \mathcal{L}_m$ of polynomials of degree $N - 1$, and the ring $R = \mathbb{Z}[X]/(X^N - 1)$ of integer polynomials modulo $(X^N - 1)$. The operation \otimes denotes multiplication in R .

The cryptosystem definition is as follows:

- *Setup*: Select randomly two polynomials $f, g \in \mathcal{L}_g$, with the constrained that f must have inverse modulo p and q .
- *KeyGeneration*: Compute the inverse of f modulo p and q :

$$F_q \otimes f \equiv 1 \pmod{q} \quad (2.1)$$

$$F_p \otimes f \equiv 1 \pmod{p} \quad (2.2)$$

Compute the following: $h = F_q \otimes g \pmod{q}$ The public key corresponds to the polynomial h , and the private key is the set of polynomials f, F_p .

- *Encryption*: Given a message $m \in \mathcal{L}_m$, randomly select a polynomial $\phi \in \mathcal{L}_\phi$. Using h , compute the ciphertext $c \equiv p(\phi \otimes h) + m \pmod{q}$
- *Decryption*: Given a ciphertext, compute first: $a \equiv f \otimes c \pmod{q}$. The coefficients of a are chosen to be in the interval $[-q/2, q/2]$. Recover the message by computing: $m \equiv F_p \otimes a \pmod{p}$.

A correlated problem with SVP is known as the learning with errors (LWE) problem, first proposed by Regev *et al* [19]. For this, an unknown polynomial $p(n) = O(n^c)$, for some constant c , a prime number $p = p(n) \leq \text{poly}(n)$, and a list of equaitons with *errors* are given:

$$\begin{aligned} \langle \mathbf{s}, \mathbf{a}_1 \rangle &\approx_\chi b_1 \pmod{p} \\ \langle \mathbf{s}, \mathbf{a}_2 \rangle &\approx_\chi b_2 \pmod{p} \\ &\vdots \end{aligned}$$

where $\langle \cdot, \cdot \rangle$ denotes the inner product module p , and the a_i are chosen independently and uniformly from \mathbb{Z}_p^n . $b_i \in \mathbb{Z}_p$, are defined as $b_i = \langle \mathbf{s}, \mathbf{a}_i \rangle + e_i$, where each $e_i \in \mathbb{Z}_p$ is chosen independently according to χ . The problem consist in recovering \mathbf{s} from these equations. From this problem or a variant of it, all other cryptosystems based on lattices are derived.

Next, we will briefly mention a current process for creating a standard post-quantum cryptosystem. As mentioned, the cryptosystems submitted to the process are of either type: code-based or lattice-based.

2.5 Post-quantum cryptosystem standards

Two reasons exist for initiating a process to standardized a Post-Quantum cryptosystem to use in Internet communications:

- The rapid pace at which the community is developing Quantum Computers along with the existence of Shor's algorithm posing a threat to classical cryptosystems.
- The time to deploy a new standard is considerably large.

For those reasons, the United States National Institute of Standards and Technology (NIST) initiated a process in 2017 [20] to create a new public-key and digital signature standard for the post-quantum era. This standard may contain one or more cryptosystems, as well as one or more signature algorithms.

The process started in December 2017 with a selection of 69 proposals out of the 82 submissions. The first round lasted until January 2019, afterwards the number of contestants was reduced to 26. The second round of selection began on January 30, 2019, and ended in July 22, 2020.

As mentioned, the NIST standardization process has cryptosystems that are alternatives for existing public-key or key exchange mechanisms and algorithms for digital signatures. In this work, we focus only on those algorithms for public-key encryption and key exchange.

The NIST provided three categories that the cryptosystems should meet to be considered for the standardization process. The first category, and the most influential one, is security. The NIST established that cryptosystems should meet the *semantically secure* criterion and have strength level IND-CCA2. In the case of ephemeral use, NIST allows a weaker security setting: IND-CPA. It also provided another set of categories that the algorithms should meet. For more information, the reader is referred to [21].

Other criteria to be considered on the submitted cryptosystems are cost and performance. Standards are aimed to be used in a wide range of applications, and factors such as the computational efficiency of the different operations, RAM usage, and packet transmission are also considered.

The third criterion to consider for the selection of the cryptosystems is the algorithm and implementation characteristics. The implementations may provide access to the cryptosystem via physical attacks such as side-channel attacks and power analysis.

Currently, the standardization process is in its third round, with 15 remaining candidates in consideration.

2.5.1 Currently competing cryptosystems

Of the 26 cryptosystems available in the second round, only 15 candidates passed to the third round. Of these, seven were selected as finalists, and eight as alternative cryptosystems. Table 2.1 indicates the final candidates for round three, including both the signature algorithms and the public-key/KEM algorithms, while Table 2.2 indicates the alternatives.

Table 2.1: The currently finalist of the NIST standardization process.

Public-Key Encryption/KEM	Digital Signatures
Classic McEliece	CRYSTALS-DILITHIUM
CRYSTALS-KYBER	FALCON
NTRU	Rainbow
Saber	

For the main contenders, there is one cryptosystem based on error-correction codes (Classic McEliece [22]), and three based on lattices (Crystal-Kyber [23], NTRU [24],

Table 2.2: The currently alternative candidates of the NIST standardization process.

Public-Key Encryption/KEM	Digital Signatures
BIKE	GeMSS
FrodoKEM	Picnic
HQC	SPHINCS+
NTRU Prime	
SIKE	

SABER [25]). For the alternative contenders, three are based on error-correcting codes (Bike [26], HQC [27], Sike [28]) and two on lattices (FrodoKEM [29], NTRU Prime [30]).

Once a new post-quantum cryptosystem standard is created, it will be necessary to integrate it into existing security protocols for network communications usage. One such protocol is Transport Layer Security, and it provides means of protecting the communication links.

2.6 The transport layer security protocol

The transport layer security (TLS) is an application protocol that provides security features to computer network communications and is a successor of the Secure Socket Layer (SSL) protocol. Any other application that requires it can use this protocol.

The protocol defines a set of ciphers, schemes for key exchange and public-key cryptography, authentication schemes, and data compression functions that provide the security features needed in computer communications and data transmission.

The TLS protocol consists of two main components: the TLS Handshake protocol, allowing the communicating parties to agree in security parameters for the TLS Record protocol; and the Record protocol, for protecting the the traffic among the parties involved in the communication. Other protocols are defined within TLS that provide additional features and handle cases that might reside outside of the scope of the two main components, for more information on those protocols see the RFC8446 [31].

We are interested on the handshake protocol, as this protocol uses the key exchange mechanisms for the parameter agreement. In the following subsection we give a brief description of the protocol.

2.6.1 TLS handshake protocol

As mentioned, the handshake protocol negotiates the security parameters of a connection. Figure 2.4 shows the basic flow of a full TLS handshake. Three phases are present in the flow:

- **Key Exchange:** Phase at which the client initiates the connection, establishes the shared key material and selects the cryptographic parameters.

- **Server Parameters:** The server establishes other handshake parameters, like the application-layer in support, whether the client is authenticated, among others.
- **Authentication:** The server authenticates itself (and optionally, the client does as well) and provides key information and handshake integrity.

After this negotiation, the data transmission can begin.

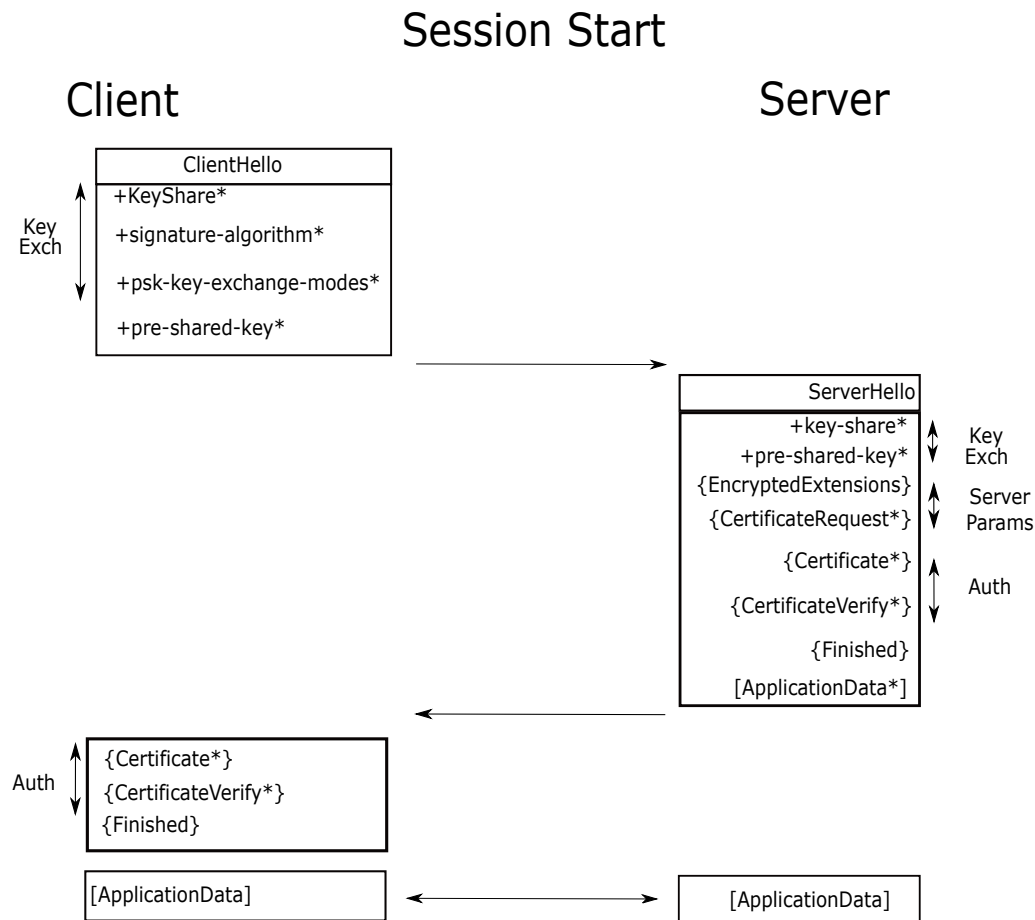


Figure 2.4: The full flow of a basic handshake for TLS v1.3 protocol. The client starts with the Key Exchange; then the server performs the three phases at once; and finally, the client authenticates itself, if required. After finishing the handshake, both endpoints share a common key, and data can be transmitted. Fields with * are optional.

On the ClientHello message, the client must include a randomly generated nonce², a list of the TLS versions supported by the client, a list of supported symmetric-key ciphers and data compression functions, and a set of Diffie-Hellman key shares. Other potentially useful parameter may be included.

The server includes in its ServerHello message the key share parameters, the server parameters, and authentication data, which includes a Certificate field and a CertificateVerified field. It includes a Finished field, indicating that the handshake ended successfully.

²A number used only once

After the client received the ServerHello message, it proceeds to authenticate the server by verifying that the parameters at the Certificate field are valid, and that the message ServerHello itself is valid.

With this phase, the handshake protocol finishes, and both the client and the server proceeds to exchange the application information.

2.6.2 Crypto libraries implementing the TLS protocol

Several libraries exist that implement the TLS protocol. In the following list, we present some of the most common implementations of the protocol.

- OpenSSL: a free implementation [32].
- Java Secure Socket Extension: Java implementation in the Java Runtime Environment, supporting TLS 1.1 and 1.2, and 1.3 for Java 11 [33].
- GnuTLS: a free, LGPL licensed, implementation [34].
- BoringSSL: An OpenSSL fork adapted to Chrome/Chromium, Android and other Google applications and programs build on top of them [35].
- LibreSSL: A fork of OpenSSL by the OpenBSD project [36].
- mbedTLS: Tiny SSL library implementation for embedded devices that is designed for ease of use [37].
- wolfSSL: TLS/SSL library for embedded applications, with a strong focus on speed and size [38].

The most popular one is the OpenSSL library.

We introduced in this chapter the basics for the Internet of Things, cryptography, and the Post-Quantum cryptosystems. We briefly introduce cryptography, especially public-key cryptography, and gave some context for the need of public-key Post-Quantum cryptosystems, introducing its two main types and representatives. We introduced the process for creating a new standard for the Post-Quantum era that will affect protocols like the TLS protocol. As Quantum Computers' existence will affect all the communications, we introduced the IoT as a way of indicating the need to start adapting the Post-Quantum cryptosystems to protect the communications from the beginning.

In the following chapter, we present the materials and methods used to develop this work, the NIST post-quantum cryptosystems, the IoT prototype, and the performance measurement.

Chapter 3

Methodology

In this chapter, we begin by presenting the post-quantum cryptosystems that, according to the literature, are suitable to use in resource-constrained and IoT devices.

From the NIST standardization process, we indicate which cryptosystems are suitable to IoT devices, present their algorithm specification, and considering their keys' sizes and some theoretical aspects, present the versions we considered are better suited for the devices in consideration.

We then present the IoT prototype, along with its hardware and software components, that we used for making the tests series. Finally, we present the variables we choose to study and how we study them. We also present the way we make the data exploration.

3.1 Introducing post-quantum cryptosystems to IoT

Not all KEMs are suitable for resource-constrained devices, as some mechanisms require a considerable amount of computational resources to execute at the required security level. The complexity of the most used classical cryptosystem, RSA, is $O(m^3)$ for decryption and $O(m^2)$ for encryption, where $m = \log(N)$ is the size in bits of the modulus N . This makes RSA unsuitable on resource-constrained devices.

A common choice is to use Elliptic Curve Cryptography, which requires smaller keys' sizes, reducing the resources needed to perform the cryptographic operations. Nevertheless, considering that Quantum Computers are currently under development and the existence of Shor's algorithm, it is reasonable to have from the beginning post-quantum cryptosystems that are easily adaptable to resource-constrained devices — considering that the currently available ones are hard to adapt.

For this reason, we perform a study of the current post-quantum cryptosystems under the NIST competition process, which would allow us to gain information on the adaptability of the cryptosystems to resource-constrained devices. In the following section, we introduce the post-quantum cryptosystems that are lattice-based from the NIST standardization process, and indicate which we will use.

3.2 Post-quantum cryptosystems suitable for IoT

We established that there exist cryptosystems that have post-quantum resistance, namely the lattice-based and the code-based cryptosystems. The latter suffers from having a greater key size (in the order of few megabytes), making them unsuitable to resource-constrained devices. On the other size, the lattice-based cryptosystems have smaller key sizes, and, according to the literature, are more suitable to resource-constrained devices [39], so we choose them for use with IoT devices.

All the mechanisms submitted to the NIST standardization competition, are build on top of public-key cryptosystems, making calls to such ones for completing the operations.

3.2.1 Lattice-based NIST post-quantum cryptosystems

As mentioned, the NIST process has five KEM that are based on lattices. Following, we present a brief description of the specification of each one. We present its security level, the problem on which it is based, relevant parameters, and the description of the algorithms involved in the KEM.

3.2.2 Algorithms specification

We begin by introducing the NTRU cryptosystem; then introduce the SABER cryptosystem, CRYSTAL-KYBER, FrodoKEM, and finally the NTRU-LPRIME one.

NTRU Algorithm Specification

The NTRU KEM is a lattice-based cryptosystem whose security problem is based on the Ring Learning with Errors (RLWE)¹ introduced by [40]. It has IND-CCA2 security and is constructed from a deterministic public-key encryption scheme that is OW-CPA secure.

NTRU is the fusion of two previous submissions from round 1: NTRUEncrypt and NTRU-HRSS-KEM. The fusion implied a unified design from the submissions, varying only in the parameters. It has two sets of parameters: NTRU-HPS following from the NTRUEncrypt submission and NTRU-HRSS following the NTRU-HRSS-KEM submission. Relevant parameters for the KEM are shown in Table 3.1.

Table 3.1: Relevant parameters of the NTRU KEM.

	ntruhs2048509	ntruhs2048677	ntruhs4096821	ntruhrss701
Public-key bytes	699	930	1230	1138
Private-key bytes	935	1234	1590	1450
Ciphertext bytes	699	930	1230	1138
Shared-key bytes	256	256	256	256
Security Category	1	3	3	5

The components of the KEM are:

¹RLWE is a variant of the problem *Learning With Errors* that make cryptosystems based on lattices more efficient.

1. KeyGeneration

- Input: A bit string *seed*.
- Output: Private key, public key.
- Operations:
 - (a) Parse *seed* as $fg_{bits} || prf_{keys}$. The operator $||$ implies concatenation.
 - (b) Set

$$\begin{aligned} & (packed_dpke_private_key, packed_public_key) \\ & = DPKE_key_pair(fg_bits) \end{aligned}$$

DPKE_key_pair is the function that generates the key pair for the PKE over which the KEM is based.

- (c) Set

$$packed_private_key = packed_private_key || bits_to_bytes(prf_key)$$

- (d) Output (*packed_private_key*, *packed_public_key*).

2. Encapsulate

- Input: Public key.
- Output: Shared key, ciphertext.
- Operations:
 - (a) Let *coins* be a string of uniform random bits .
 - (b) Set $(r, m) = Sample_rm(coins)$.
 - (c) Set $packet_rm = pack_s3(r) || pack_s3(m)$.
 - (d) Set

$$\begin{aligned} & shared_key \\ & = Hash(byte_to_bits(packed_rm, 8 * dpke_plaintext_bytes)) \end{aligned}$$

- (e) Set

$$\begin{aligned} & packed_ciphertext \\ & = DPKE_Encrypt(packed_public_key, packed_rm) \end{aligned}$$

- (f) Output (*shared_key*, *packed_ciphertext*).

3. Decapsulate

- Input: Private key, ciphertext.
- Output: Shared key.
- Operations:

(a) Parse $packed_private_key$ as

$$packed_f || packed_fp || packed_hq || prf_key$$

(b) Set

$$\begin{aligned} & (packed_rm, fail) \\ & = DPKE_Decrypt(packed_private_key, packed_ciphertext) \end{aligned}$$

(c) Set

$$\begin{aligned} & shared_key \\ & = Hash(bytes_to_bits(packed_rm, 8 * dpke_plaintext_bytes)) \end{aligned}$$

(d) Set

$$\begin{aligned} & random_key \\ & = Hash(bytes_to_bits(prf_key, prf_key_bytes) \\ & \quad || bytes_to_bits(packed_ciphertext, 8 * kem_ciphertext_bytes)) \end{aligned}$$

(e) If $fail = 0$, output $shared_key$, else output $random_key$.

SABER Algorithm Specification

Saber is a lattice-based KEM believed to offer resistance to Quantum Computers. Its security is based on the hardness of solving the problem Module Learning with Rounding (MLR) problem and offers IND-CCA. The KEM is built over a PKE that offers IND-CPA security. Relevant parameters for the KEM are shown in Table 3.2

Table 3.2: Relevant parameters of the SABER KEM.

	LightSaber	Saber	FireSaber
Public-key bytes	672	1568	736
Private-key bytes	992	2304	1088
Ciphertext bytes	1312	3040	1472
Shared-key bytes			
Security Category	1	3	5
Classical	169	244	338
Quantum	153	226	308

The components of the KEM are:

1. KeyGeneration:

- $(seed_A, \mathbf{b}, \mathbf{s}) = \text{Saber.PKE.KeyGen}()$.
- $pk = (seed_A, \mathbf{b})$.
- $pkh = \mathcal{F}(pk)$.

- $z = \mathcal{U}(\{0, 1\}^{256})$.
- **return** $(pk := (seed_{\mathbf{A}}, \mathbf{b}), sk := (z, pkh, pk, \mathbf{s}))$.

2. Encapsulation:

- $m = \mathcal{U}(\{0, 1\}^{256})$.
- $(\widehat{K}, r) = \mathcal{G}(\mathcal{F}(pk), m)$.
- $c = \text{Saber.PKE.Encrypt}(pk, m; r)$.
- $K = \mathcal{H}(\widehat{K}, c)$.
- **return** (c, K) .

3. Decapsulation:

- $m' = \text{Saber.PKE.Decap}(\mathbf{s}, c)$.
- $(\widehat{K}', r') = \mathcal{G}(pkh, m')$.
- $c' = \text{Saber.PKE.Enc}(pk, m'; r')$.
- **if** $c = c'$ **then**
- **return** $K = \mathcal{H}(\widehat{K}', c)$.
- **else**
- **return** $K = \mathcal{H}(z, c)$.

CRYSTAL-KYBER Algorithm Specification

The *Cryptographic Suite for Algebraic Lattices* (CRYSTAL) encompasses two cryptographic primitives: KYBER, an IND-CCA2-secure KEM, and Dilithium, a strong digital signature. KYBER is a lattice-based KEM whose security resides on the hardness of solving the Module Learning With Errors (MLWE) problem and is built over an IND-CPA-secure public-key cryptosystem. Relevant parameters are shown in Table 3.3:

Table 3.3: Relevant parameters for the CRYSTAL-KYBER KEM.

	Kyber512	Kyber768	Kyber1024
Public-key bytes	1632	2400	3108
Private-key bytes	800	1184	1568
Ciphertext bytes	736	1088	1568
Shared-key bytes	32	32	32
Security Category	1	3	5
Classical Security	111	181	254
Quantum Security	100	164	230

The components of the KEM are:

1. **KeyGeneration:**

- $z \leftarrow \mathcal{B}^{32}$. Uniform random sampling a 32-byte array.
- $(pk, sk') := \text{KYBER.CPAPKE.KeyGen}()$. Generate the keys from the Kyber CPA PKE.
- $sk := (sk' || pk || H(pk) || z)$, where H is the hash function SHA3-256.
- **return** (pk, sk) .

2. Encapsulate:

- $m \leftarrow \mathcal{B}^{32}$.
- $m \leftarrow H(m)$.
- $(\bar{K}, r) := G(m || H(pk))$. G is the hash function SHA3-512
- $c := \text{KYBER.CPAPKE.Enc}(pk, m, r)$.
- $K := \text{KDF}(\bar{K}, ||H(c))$. KDF, key derivation function, instantiated with SHAKE-256.
- **return** (c, K) .

3. Decapsulate:

- $pk := sk + (12 * k * (n/8))$.
- $h := sk + (24 * k * (n/8)) + 32 \in \mathcal{B}^{32}$.
- $z := sk + (24 * k * (n/8)) + 64$.
- $m' = \text{KYBER.CPAPKE.Dec}(s, (u, \mathbf{v}))$, where $c = (u, \mathbf{v})$.
- $(\bar{K}', r') := G(m' || h)$.
- $c' := \text{KYBER.CPAPKE.Enc}(pk, m', r')$.
- **if** $c = c'$ **then**
- **return** $K := \text{KDF}(\bar{K}' || H(c))$.
- **else**
- **return** $K = \text{KDF}(z || H(c))$.
- **end if**

FrodoKEM Algorithm Specification

FrodoKEM is a lattice-based KEM whose security resides on the hardness of solving the Learning With Errors (LWE) problem. It is designed for IND-CCA security and is built over a PKE scheme with IND-CPA security. The authors designed this KEM following a conservative approach, preferring simplicity and security over performance and optimization, resulting in a KEM with the largest key sizes and execution times.

The KEM has three variants, as described in the Table 3.4. For each variant, the authors have two implementations: one with AES and another with SHAKE². The logic

²Both are compression functions.

for this is: the KEM is faster with AES when there is a hardware implementation, while the KEM is faster with SHAKE when the implementation is purely software.

Relevant parameters are shown in Table 3.4, and its components of FrodoKEM are described as follows:

Table 3.4: Relevant parameters for the FrodoKEM KEM.

	FrodoKEM-640	FrodoKEM-976	FrodoKEM-1344
Public-key bytes	9616	15632	21520
Private-key bytes	19888	31296	43088
Ciphertext bytes	9720	15744	21632
Shared-key bytes	16	24	32
Security Category	1	3	5
Classical Security	144	209	274
Quantum Security	103	150	196

1. KeyGeneration

- Choose uniformly random seed \mathbf{s} .
- Generate pseudorandom seed $seed_A$.
- Generate the matrix \mathbf{A} via $\mathbf{A} \leftarrow \text{Frodo.Gen}(seed_A)$.
- Generate pseudorandom bit string $(r^{(0)}, r^{(1)}, \dots, r^{(2n\bar{n}-1)})$.
- Sample the error matrix \mathbf{S} .
- Sample the error matrix \mathbf{E} .
- Compute $\mathbf{B} \leftarrow \mathbf{AS} + \mathbf{E}$.
- Compute $\mathbf{b} \leftarrow \text{Frodo.Pack}(\mathbf{B})$.
- Compute $\mathbf{pkh} \leftarrow \text{SHAKE}(seed_A || \mathbf{b}, \text{len}_{\mathbf{pkh}})$
- Return public key $pk \leftarrow seed_A || \mathbf{b}$ and secret key $sk' \leftarrow (s || seed_A || \mathbf{b}, \mathbf{S}, \mathbf{pkh})$.

2. Encapsulation

- Choose a uniformly random key μ .
- Compute $\mathbf{pkh} \leftarrow \text{SHAKE}(pk, \text{len}_{\mathbf{pkh}})$.
- Generate pseudorandom values

$$seed_{SE} || k \leftarrow \text{SHAKE}(\mathbf{pkh} || \mu, \text{len}_{seed_{SA}} + \text{len}_k)$$

- Generate pseudorandom bit string $(r^{(0)}, r^{(1)}, \dots, r^{(2\bar{m}n + \bar{m}\bar{n} - 1)})$.
- Sample error matrix \mathbf{S}' .
- Sample error matrix \mathbf{E}' .
- Generate $\mathbf{A} \leftarrow \text{Frodo.Gen}(seed_A)$.

- Compute $\mathbf{B}' \leftarrow \mathbf{S}'\mathbf{A} + \mathbf{E}'$.
- Compute $\mathbf{c}_1 \leftarrow \text{Frodo.Pack}(\mathbf{B}')$.
- Sample error matrix \mathbf{E}'' .
- Compute $\mathbf{B} \leftarrow \text{Frodo.Unpack}(\mathbf{b}, n, \bar{n})$.
- Compute $\mathbf{V} \leftarrow \mathbf{S}'\mathbf{B} + \mathbf{E}''$.
- Compute $\mathbf{C} \leftarrow \mathbf{V} + \text{Frodo.Encode}(\mu)$.
- Compute $\mathbf{c}_2 \leftarrow \text{Frodo.Pack}(\mathbf{C})$.
- Compute $\mathbf{ss} \leftarrow \text{SHAKE}(c_1 || c_2 || k, \text{len}_{ss})$.
- Return ciphertext $c_1 || c_2$ and shared secret \mathbf{ss} .

3. Decapsulate

- $\mathbf{B}' \leftarrow \text{Frodo.Unpack}(c_1)$.
- $\mathbf{C} \leftarrow \text{Frodo.Unpack}(c_2)$.
- Compute $\mathbf{M} \leftarrow \mathbf{C} - \mathbf{B}'\mathbf{S}$.
- Compute $\mu' \leftarrow \text{Frodo.Decode}(\mathbf{M})$.
- Parse $pk \leftarrow \text{seed}_A || b$.
- Generate pseudorandom values $\text{seed}_{SE'} || k'$.
- Generate pseudorandom bit string $(r^{(0)}, r^{(1)}, \dots, r^{(2\bar{m}n + \bar{m}\bar{n} - 1)})$.
- Sample error matrix \mathbf{S}' .
- Sample error matrix \mathbf{E}' .
- Generate $\mathbf{A} \leftarrow \text{Frodo.Gen}(\text{seed}_A)$.
- Compute $\mathbf{B}'' \leftarrow \mathbf{S}'\mathbf{A} + \mathbf{E}'$.
- Sample error matrix \mathbf{E}'' .
- Compute $\mathbf{B} \leftarrow \text{Frodo.Unpack}(\mathbf{b}, n, \bar{n})$.
- Compute $\mathbf{V} \leftarrow \mathbf{S}'\mathbf{B} + \mathbf{E}''$.
- Compute $\mathbf{C}' \leftarrow \mathbf{V} + \text{Frodo.Encode}(\mu')$.
- if $B' || C = B'' || C$ then
- Return shared secret $\mathbf{ss} \leftarrow \text{SHAKE}(c_1 || c_2 || k', \text{len}_{ss})$.
- else
- Return shared secret $\mathbf{ss} \leftarrow \text{SHAKE}(c_1 || c_2 || s, \text{len}_{ss})$.

Table 3.5: Relevant parameters for the NTRUPLPrime KEM.

	sntrup653	ntrulpr653	sntrup761	ntrulpr761	ntrulpr857	sntrup857
Public-key bytes	994	897	1158	1039	1184	1322
Private-key bytes	1518	1125	1763	1294	1463	1999
Ciphertext bytes	897	1025	1038	1167	1312	1184
Shared-key bytes	32	32	32	32	32	32
Security Category	2	3	4	2	3	4
Classical Security	174	176	208	210	237	239
Quantum Security	160	160	180	185	208	209

NTRU Prime Algorithm Specification

NTRU Prime provides two KEMS: "Streamlined NTRU Prime" and "NTRU LPrime." NTRU Prime is a lattice-based cryptosystem whose security is based on the Ring Learning With Errors (RLWE) problem, and it provides IND-CCA2 protection. The Streamlined NTRU Prime KEM is constructed from a deterministic PKE scheme with OW-CPA security. NTRU LPrime is built from a deterministic PKE with OW-CPA security, named NTRU LPrime Expand, which in turn, is constructed from a randomized PKE.

Relevant parameters are shown in Table 3.5, and its components are as follows:

The components of NTRU LPrime are the following:

1. KeyGeneration

- Compute $(K, k) \leftarrow \text{KeyGen}()$, where *KeyGen* comes from the PKE scheme.
- Encode K as a string $\underline{K} \in \underline{\text{PublicKeys}}$.
- Encode k as a string $\underline{k} \in \underline{\text{SecretKeys}}$.
- Generate a uniform random number $\rho \in \underline{\text{Inputs}}$.
- Output $(\underline{K}, (\underline{k}, \underline{K}, \rho))$.

2. Encapsulate

- Input $\underline{K} \in \underline{\text{PublicKeys}}$. Decode \underline{K} , obtaining $K \in \text{PublicKeys}$.
- Generate a uniform random number $r \in \underline{\text{Inputs}}$. Encode r as a string $\underline{r} \in \underline{\text{Inputs}}$.
- Compute $c = \text{Encrypt}(r, K) \in \underline{\text{Ciphertexts}}$. Encode c as a string $\underline{c} \in \underline{\text{Ciphertexts}}$.
- Compute $C = (\underline{c}, \text{HashConfirm}(r, \underline{K})) \in \underline{\text{Ciphertexts}} \times \underline{\text{Confirm}}$.
- Output $(C, \text{HashSession}(1, r, C))$.

3. Decapsulate

- Input $C = (\underline{c}, \gamma) \in \underline{\text{Ciphertexts}} \times \underline{\text{Confirm}}$, and $\underline{k}, \underline{K}, \rho \in \underline{\text{SecretKeys}} \times \underline{\text{PublicKeys}} \times \underline{\text{Inputs}}$.
- Decode \underline{c} , obtaining $c \in \underline{\text{Ciphertexts}}$.

- Decode \underline{k} , obtaining $k \in \text{SecretKeys}$.
- Compute $r' = \text{Decrypt}(c, k) \in \text{Inputs}$, where *Decrypt* comes from the PKE scheme.
- compute $\underline{r}', \underline{c}', \underline{c}'$ as in the Encapsulate operation.
- If $C' = C$, output **HashSession**(1, \underline{r}, C). Otherwise output **HashSession**(ρ, C).

The components of Streamlined NTRU Prime are the following:

1. KeyGeneration

- Compute

$$(K, k) \leftarrow \text{KeyGen}'()$$
 where *KeyGen'* comes from the PKE scheme.
- Encode K as a string $\underline{K} \in \text{PublicKeys}'$.
- Encode k as a string $\underline{k} \in \text{SecretKeys}$.
- Generate a uniform random $\rho \in \text{Inputs}$.
- Output $(\underline{K}, (\underline{k}, \underline{K}, \rho))$.

2. Encapsulate

- Input $\underline{K} \in \text{PublicKeys}'$.
- Decode \underline{K} , obtaining $K \in \text{PublicKeys}'$.
- Generate a uniform random $r \in \text{Inputs}$.
- Encode r as a string $\underline{r} \in \text{Inputs}$.
- Compute $c = \text{Encrypt}'(r, K) \in \text{Ciphertexts}$. Encode c as a string $\underline{c} \in \text{Ciphertexts}$.
- Compute $C = (\underline{c}, \text{HashConfirm}(\underline{r}, \underline{K})) \in \text{Ciphertexts} \times \text{Confirm}$.
- Output $(C, \text{HashSession}(1, \underline{r}, C))$.

3. Decapsulate

- Input $C = (\underline{c}, \gamma) \in \text{Ciphertexts} \times \text{Confirm}$, and $\underline{k}, \underline{K}, \rho \in \text{SecretKeys} \times \text{PublicKeys}' \times \text{Inputs}$.
- Decode \underline{c} , obtaining $c \in \text{Ciphertexts}$.
- Decode \underline{k} , obtaining $k \in \text{SecretKeys}$.
- Compute $r' = \text{Decrypt}(c, k) \in \text{Inputs}$, where *Decrypt* comes from the PKE scheme.
- compute $\underline{r}', \underline{c}', \underline{c}'$ as in the Encapsulate operation.
- If $C' = C$, the output **HashSession**(1, \underline{r}, C). Otherwise output **HashSession**(ρ, C).

3.2.3 NIST's post-quantum cryptosystems suitable to IoT devices

Complying with a requirement from the NIST, the capability of the mechanism to run on a wide variety of devices, each mechanism has different versions with different levels of security.

The security levels allowed from the NIST, are categorized from level 1 (equivalent 128-bit cipher security) to level 5 (equivalent 512-bit symmetric security.) The lower the security level, the smaller the keys' sizes. Considering that the mechanisms' execution time is directly proportional to the keys' size, choosing those implementation with the lowest security level is convenient for use with resource-constrained devices.

From each mechanism, the versions with the smaller keys' sizes are:

- For SABER, LightSaber with a public-key size of 672, private-key size of 992, and ciphertext of size 1,312 bytes. Its expected security is category 1.
- For CRYSTAL-KYBER, Kyber512 with a public-key size of 1,632, a private-key size of 800, and ciphertext of 736 bytes. Its claimed security level is 1.
- For NTRU, NTRUhs2048509 with a public-key size of 699, a private key-size of 935, and a ciphertext size of 699 bytes. It provides security of level 1.
- For NTRU Prime, NTRULPr653 with a public-key size of 897, a private-key size of 1,125, and ciphertext of 1,025 bytes. Its expected strength falls in category 2.
- For FrodoKEM, FrodoKEM640 with a public-key size of 9,616, a private-key size of 19,888, and ciphertext size of 9,720 bytes. It provides security of level 1.

We study the previously mentioned mechanisms to know their performance on resource-constrained devices.

3.3 The IoT prototype

An objective of this work is to test the performance of the different mechanisms into IoT devices. As stated previously, two basic types of IoT devices exist a WSN and RFID tags. We focus primarily on the WSN, as the working principle is the same for both, a component that communicates with a server for information transmission, and a node in a WSN provides more computation capability than a tag of an RFID system.

A WSN consists of three components: a server, a gateway or sink, and a set of nodes, known as the sensor field. The gateway and the server communicate over the Internet. In contrast, the nodes communicate with the gateway using a wide variety of wireless technologies, like NFC, Bluetooth, or LoRa. In the following, we will introduce the system implemented for the IoT prototype and its components.

3.3.1 The proposed system

The proposed system for testing is shown in Figure 3.1 with all the blocks that comprises it. Its three main components are: a sensor field from which data is gathered; a gateway

for collecting the data from the sensor field, a client for displaying it; and a cloud server on which a broker instance from the MQTT protocol is mounted.

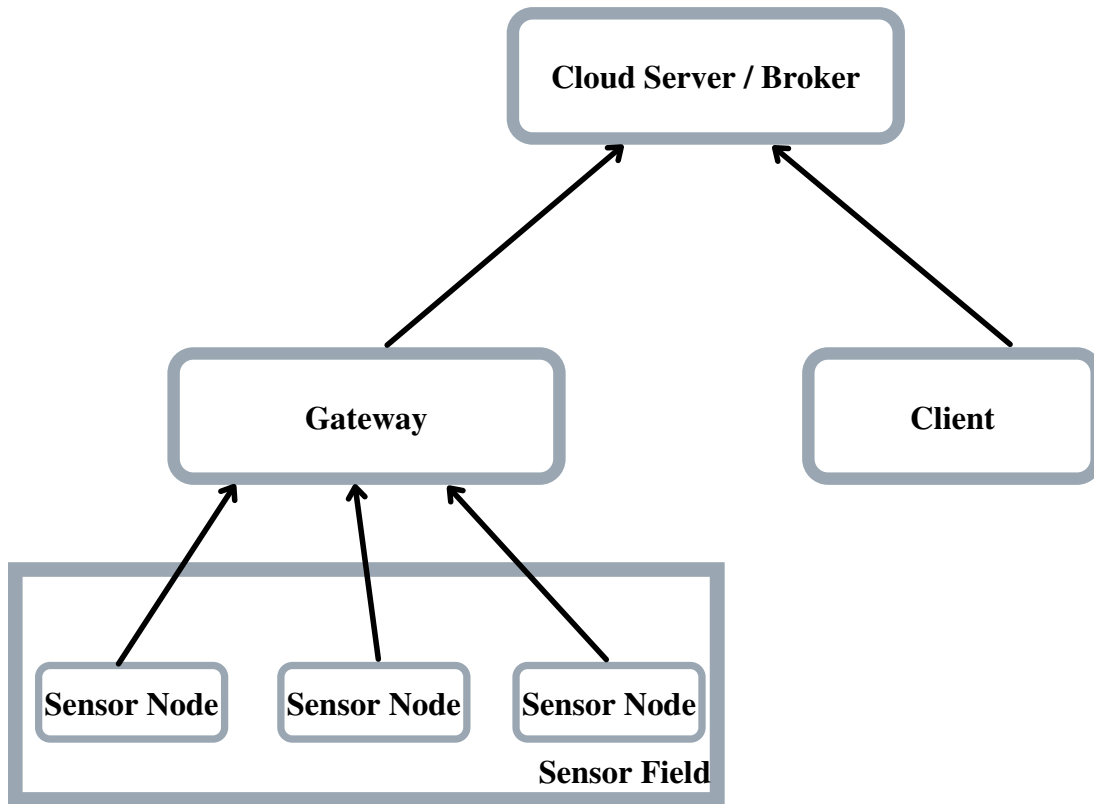


Figure 3.1: A general overview of the system architecture, showing all the necessary components to have a minimal IoT prototype working: A server, a gateway, and set of node.

The broker runs on a cloud server, accessed globally. The system uses technology available on the market and can be easily replicated. In the following, we provide a brief description for each component.

3.3.2 The nodes

The first component of the system is the sensor field, comprise of several sensing nodes. The nodes consist of three physical components: a processing unit, a communication unit, and a sensing unit.

For the processing unit, we decided to use the Arduino Nano with the ATmega 328P microcontroller. For the communication unit, we used the module RFM69HCW compatible with the LoRa standard. It has a reach of up to 400 m and an operating frequency of 915 MHz. For the sensing unit, we used the DHT 22 sensor for humidity and temperature.

We also need to access and control the hardware from the processing unit. For using the RFM module, we use the library provided by RadioHead, specifically the version for

unreliable datagram [41]. For the DHT sensor, we used the library provided by Adafruit [42]. All the programs were written using the Arduino IDE available on its web page. Figure 3.2 shows the node’s diagram with its three elements.

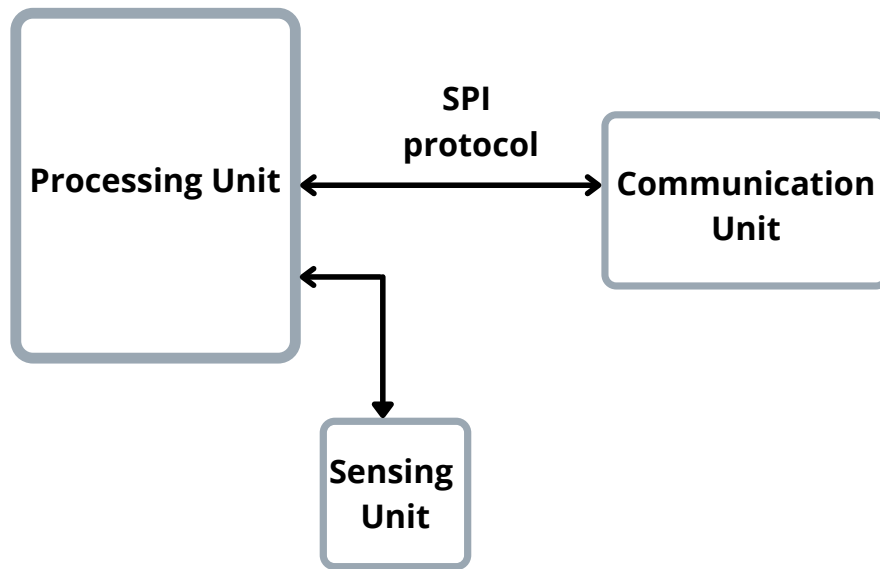


Figure 3.2: The block diagram of the sensor node. It consists of three parts, the processing unit, which uses an Arduino Nano; the communication unit, an RFM69HCW; and a sensing unit, the DHT22 sensor.

3.3.3 The gateway

Another system’s essential part is the gateway, whose function is to collect data from the sensors and send it to the broker. We use the gateway as well for testing the mechanisms and getting the necessary data for measure the performance and give the guidelines.

We used three hardware components for creating the gateway: an RFM69HCW module for receiving the data from the sensor nodes, an Arduino Uno that controls the RFM, and a Raspberry Pi 3B+. The Raspberry communicates with the Arduino via a USB cable and the UART protocol.

The gateway’s software has two components, the software for the Arduino and the software for the Raspberry. The Arduino receives the packets from the nodes via the RFM modules and sends them to the Raspberry via USB, and its diagram is shown in Figure 3.3. The Raspberry runs an MQTT client, which receives the data from the Arduino and then sends the data to the broker.

For the Arduino, we used once again the library by RadioHead for controlling the RFM [41]. For the Raspberry, we used the following libraries:

- OpenSSL with the post-quantum cryptosystems integrated.
- Modified Paho C MQTT client, to add support to the TLS 1.3 API, necessary for using the post-quantum cryptosystems.

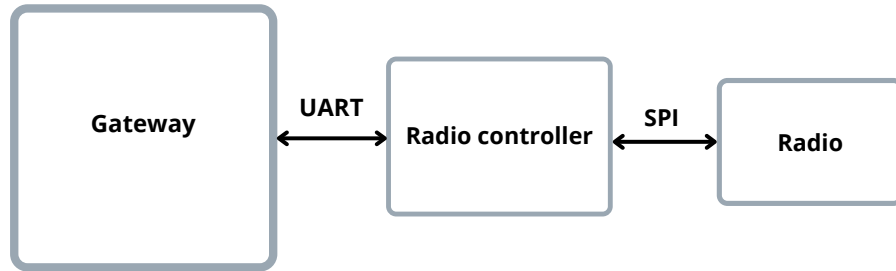


Figure 3.3: The block diagram for the gateway part of the system. No direct connection between the gateway and the radio was possible, so an intermediary was necessary. For the gateway, we used a Raspberry Pi 3B+; for the radio controller, we used an Arduino Nano; and an RFM69HCW for the radio.

We also have a client running on a separate computer, which subscribes to the broker to receive the broker’s data. It runs as well with the following libraries:

- OpenSSL with the post-quantum cryptosystems integrated.
- Modified Paho C MQTT client, same as the gateway.

Both clients have access to the CA certificate so they can complete the handshake with the broker appropriately.

3.3.4 The broker

The final part of the system is the broker. The broker runs on an Azure cloud system running Ubuntu 18.04.5 LTS. The broker is an instance of Mosquitto, modified to use the TLS 1.3 interface provided by OpenSSL. The OpenSSL library integrates the post-quantum mechanisms, same as with the clients. The broker also has access to the following certificates:

- Server certificate.
- Server private key.
- CA certificate.

The broker is configured to only receive connections on port 8883 and accept the post-quantum KEMs and the available elliptic curves.

3.4 Measuring the cryptosystems performance

On resource-constrained devices, one of the most limiting factors in the design is the energy source used by the device. Usually, the devices are designed to operate with batteries with a minimal capacity, such as watch batteries, 9V batteries, or alike. The components of the device are designed to use as little battery as possible, including the software.

3.4.1 Variables of interest

For software implementations, it is not easy to directly measure the impact that the code has on energy consumption, as different subsystems might be operating concurrently when a particular code is executed. However, it has been demonstrated that, in embedded devices, the subsystems that most use energy are: radios (such as Bluetooth and Wi-Fi), and the CPU. If we can minimize the radios' and CPU usage of our program, we might indirectly diminish the device's energy consumption [43].

It is possible to measure how many CPU cycles a particular piece of code uses during execution on software implementations. We can measure the size and quantity of packets a program transmits when using the radios, giving us metrics of the program's performance and can be used for optimization purposes.

Another essential variable considered in the design of software for embedded devices, which has less impact on energy consumption but more on resource consumption on the software side, is memory usage. Minor use of memory allows having more tasks executing concurrently, or if the number of tasks is fixed, it allows hardware usage with less capability. Hardware with fewer capabilities can have two impacts on the design: less energy consumption, and minor manufacturing costs.

For the reasons exposed previously, and considering that the software will be communicating via the Internet (and thus using the Wi-Fi), we considered that the previously mentioned variables of interest are the most important to study. We now describe how to measure such variables.

3.4.2 Profiling the variables

For profiling the different KEMs, we implemented a program with the following characteristics:

- It allows the user to select at compile time what resource to profile: memory or CPU.
- It allows the user to select at compile time the KEM to test.
- It allows the user to select at compile time the platform to test the KEM on an x86 architecture or an ARM architecture.
- It has a function to profile each operation separately: Key Generation, Encapsulation, Decapsulation.

- For profiling the CPU usage, it executes each operation 2,000 times (arbitrarily chose); and, for the memory, it executes each operation only once.
- The CPU's results are stored on a CSV file (name provided by the user).

For executing the tests, a static library containing all the corresponding KEM functions should be created and stored on the same file as the code. The program executes the corresponding operations to measure the memory, leaving out all the code necessary for profiling the CPU. For CPU usage, we have two functions that return the number of cycles, depending on the platform chosen. For x86 architectures, we have an assembly instruction that returns the timestamp counter of the CPU: *rdtsc*. In ARM architecture, we use the system's call *clock_gettime* and convert the result to cycles. We call this function before and after the call to the operation and then subtract the results to obtain the total number of cycles used. For getting the time in milliseconds, we use the system's call *gettimeofday* and use before and after executing the corresponding operation. We convert the result to milliseconds and subtract the results to get the total time.

For measuring the memory's usage of a program, there exists a command-line tool called Valgrind. This tool allows profiling the memory of a program to obtain information such as memory leaks, heap and stack usage, memory error detection, and many other functions. It provides a set of tools to accomplish such a goal. We are interested in a tool called *massif*, which profiles the heap and stack usage. For profiling the memory, we used the following command:

```
valgrind --tool=massif --stacks=yes --time-unit=B  
--massif-out-file=outputfile cmd
```

Where *-time-unit=B* tells *massif* to use the number of bytes allocated on the heap, the option *-massif-out-file* tells the program where to store the information. *massif* only profiles the heap by default, so to completely measure the memory's usage, the option *-stacks=yes* tells the tool to profile the entire memory (heap and stack). *cmd* is the program to profile. We run once the program for profiling the memory, as in every run, the same results are returned. For more information on valgrind, we recommend to visit its web page or check the man page [44].

For monitoring the connection, we used Wireshark, and a lightweight alternative it provides for resource-constrained devices. Wireshark is a program that allows the user to monitor the usage of different network interfaces, such as wireless interfaces, Ethernet, Bluetooth, among others. It provides data such as the protocol in use (TCP, UDP, SSL, among others), statistics on the connection (such as duration, number of packets, among others), and allows to trace a connection, among many others functionalities. Its lightweight alternative for resource-constrained devices is named *tshark* and runs on the command-line. It provides the same basic functionalities as Wireshark and can store the data on a file that later can be loaded with Wireshark.

For profiling the connection, we used the following command:

```
sudo tshark -i wlp2s0b1 -t ad -w outputfile host ip
```

The option *-i* tells *tshark* which interface to monitor, *-tad* tells *tshark* to print the timestamp in absolute date, *-w* indicates the file to store the data captured, *host* tells the IP to monitor. Then, using Wireshark, we opened the file generated by the command and got statistics from the connection; specifically, we obtain information about the number

of packets sent, the number of bytes per connection, and the connection duration milliseconds. For more information on *tshark* or Wireshark, we recommend to visit the Wireshark web page, or to read the corresponding man page [45].

3.4.3 Data exploration and tests execution

Having a way to collect the data from the KEMs' performance, we should obtain information from it, specifically obtain guidelines in the use of the KEMs in resource-constrained devices.

For giving the guidelines, we first execute the tests for obtaining data on the cryptosystems' performance. Then we obtain a summary of the performance via a set of statistics. The proposed statistics are: mean, maximum, and standard deviation. The mean, maximum, and standard deviation have the same units as the variable of interest. With those statistics, we could get an idea of the overall performance of the ciphers.

A way of quickly obtaining information is via graphs, so the next step is to plot the data on a graph. Different graphs can be used for that purpose. The mean and maximum can be plotted on bar graphics; the bar graphs can give us an idea of the magnitude of the statistics or variable and compare different values of the same statistic or variable. Plotting the standard deviation in a bar graph could give us an idea of how much each KEM's performance deviates from the mean, and which deviates the least.

Other information can be obtained from graphics that are not obtainable from statistics. Specifically, we can obtain information about the behavior over time of the variable of interest. A line graph is used to observe the behavior of the variable, plotting the data obtained from all the iterations. We could get other information from such a line graph, for example, how much the variable deviates from the mean and thus gaining more insight. The third step is to plot the data on a line graph for each variable.

For executing Wi-Fi tests, it is required that the mechanisms are already integrated into the OpenSSL library, as it is required a fully functional implementation of the TLS protocol. On the other side, executing the tests for CPU and RAM can be done without any external library. For those reasons, the performance tests were divided into two parts, the first part for testing the CPU and the RAM usage, and the second part for the Wi-Fi usage.

In the second part, we also compare the post-quantum mechanisms' performance against classical ciphers. In particular, because the TLS protocol version 1.3 removed RSA for key exchange, we compared the KEMs against elliptic curves with similar security levels: the elliptic curve P-256, and the elliptic curve X25512, both with key size of 256 bits.

For both tests series, we apply these three steps several times; first, on a PC to gain a general overview of each variable's behavior, and then on the Raspberry Pi, to obtain the real data on a device with fewer resources than an average computer. It is worth mentioning that we also considered theoretical aspects of the different cryptosystems, precisely the strength, security level in bits, and keys' sizes.

3.5 Postamble

In this section, we introduced the materials and methods used for the development of this work. We introduced the software and hardware components we used, the proposed IoT system, and the post-quantum cryptosystems currently in the standardization process. We also presented the method for gaining the required information.

In the next section, we present the results of the data's exploration. We then present the results of the tests in the form of figures. We present first the memory results regarding the memory performance and then the CPU ones, and after this, we select three out of the five available at the process for testing on the IoT prototype. We then proceed to test the KEMs on the prototype, especially from the connection stand-point of view. We then review all the results to present the guidelines for selecting an appropriate KEM for resource-constrained devices.

Chapter 4

Results

We now begin the study of the KEMs suitable for resource-constrained devices. To this end, we will restate the theoretical considerations for selecting it, and develop an empirical study by measuring several variables of interest. Finally, we present a proof-of-concept, where we compare the post-quantum ciphers with classical ciphers.

We begin by studying the different KEMs' CPU and memory performance, as this experiment does not require integrating the KEMs into OpenSSL.

After knowing the KEMs' performance and the theoretical security parameters, we present the three most suitable ones, giving preference to the empirical performance rather than the theoretical security parameters.

We then proceed to test the selected KEMs on an IoT prototype. We test the ciphers with a real-world scenario using a commonly-used crypto library, an IoT application protocol, and a cloud server. With all these, we give some recommendations on the ciphers, which to use, and under which circumstances.

4.1 Measuring the cryptosystems' performance

We begin by studying the CPU and memory usage for each KEM. We measure how much time and cycles each of the operations take (Key Generation, Encapsulation, and Decapsulation). Besides, we measure the memory used for each KEM as a whole.

4.1.1 Memory and CPU Performance

First, we start by analyzing the amount of memory used for each KEM. For the memory, we focused only on the maximum amount; this limits the device's minimum requirements.

Figure 4.1 shows the maximum amount of memory used for each KEM. We can see that the KEM that uses the least memory is LightSaber, followed by NTRU Prime, then NTRU, and Kyber. The KEM that uses the most memory is FrodoKEM, with a difference of two orders of magnitude with respect to Kyber512 (the second mechanisms that uses the most memory.)

Figure 4.2 shows the behavior of memory access over time. This information can also help us select the cipher, as more time using the memory could imply more CPU usage.

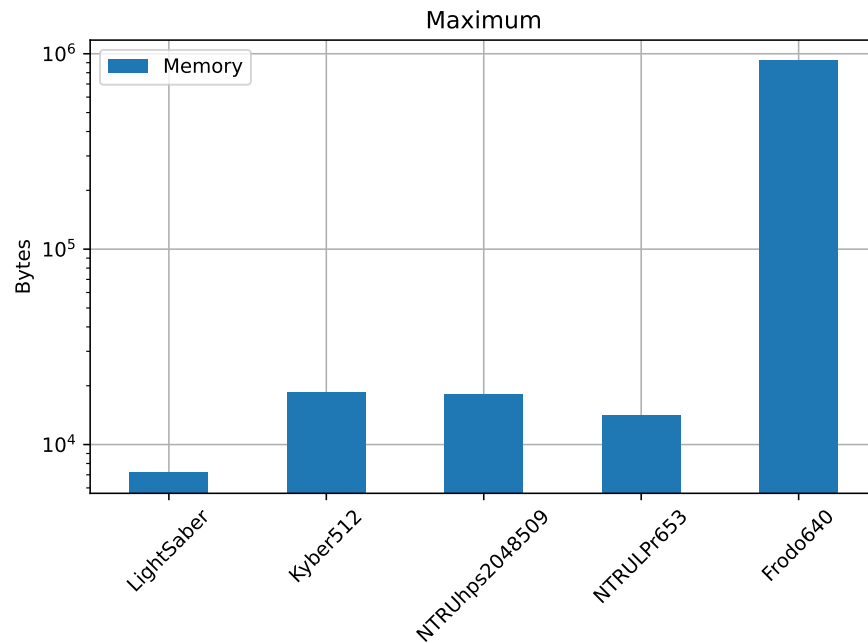


Figure 4.1: The figure shows the maximum amount of memory in bytes each mechanism uses. The y -axis has logarithmic scale.

We can see that the KEM that uses the memory for the least amount of time is LightSaber, about half the time with respect to the one that uses the most memory, NTRUP. By memory time usage, Saber is followed by Kyber, NTRU, FrodoKEM, and NTRUP. Although FrodoKEM uses less memory than NTRU and NTRUP, it is not an option for resource-constrained devices, as the maximum amount memory is more significant than the others.

We can start thinking about discarding FrodoKEM for the memory requirements and consider only the other four. We will still show the CPU usage of the five KEMs, as we can have a better-informed decision from such data.

Now we analyze the KEMs' CPU performance. From the four statistics computed, the one that gives us the most information is the mean, as it tells us how the variable behaved over time. Figure 4.3 shows the mean execution time for each operation: Key Generation, Encryption, and Decryption. The units are the number of cycles.

Figure 4.3 shows that the two best mechanisms are LightSaber and Kyber512, the last mechanism has the best execution time overall. Although NTRU has a Key Generation execution time more significant than the FrodoKEM, the other two operations take less CPU time.

Analyzing the total CPU time, we can see as well that FrodoKEM is the second slowest one, with slowest one being NTRU Prime.

Table 4.1 presents the standard deviation from each operation execution time. We can see that the KEM that presents the least dispersion is Kyber512, followed by a LightSaber. The data follow a similar pattern to that of the mean, having NTRU Prime the most dispersion.

Having that information, we can now select three ciphers to be used in resource-

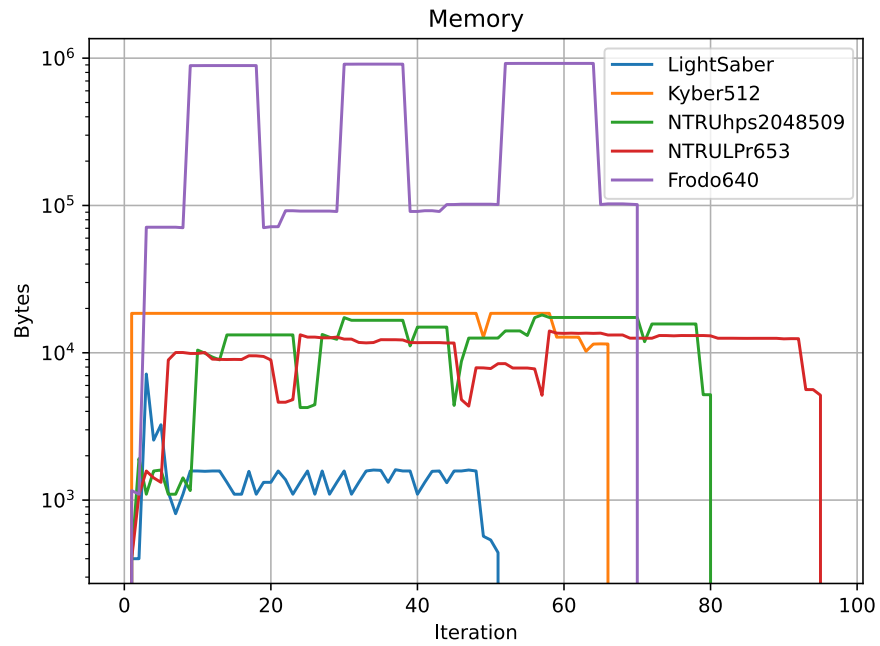


Figure 4.2: The memory behavior through time for each of the mechanisms, indicating for how long each mechanism accessed the memory. The y -axis has logarithmic scale.

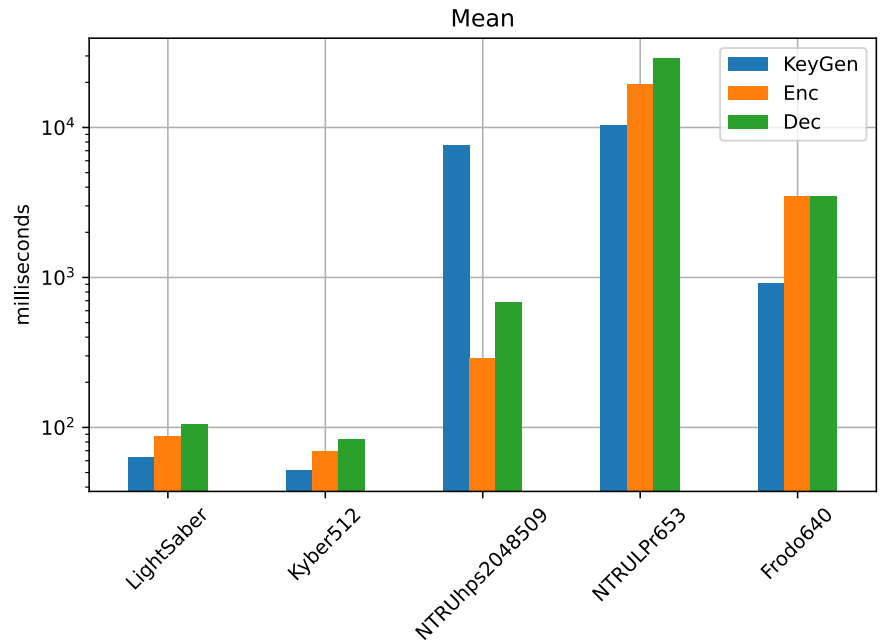


Figure 4.3: Mean usage of the CPU for each operation of each mechanism.

Table 4.1: The standard deviation of CPU usage, for each of the KEMs and each operation.

KEM	KeyGeneration	Encryption	Decryption
LightSaber	13.5401	17.7455	21.0055
Kyber512	10.3901	13.3879	15.1921
NTRUhs2048509	333.2433	18.2681	37.3123
NTRULPr653	912.384	1,519.0519	2,042.7712
FrodoKEM640	34.1741	55.5502	54.5737

constrained devices and integrate them into OpenSSL.

4.1.2 Selecting the first three KEMs

From the previous results, we can now select three mechanisms to be used in resource-constrained devices and integrate them in OpenSSL. In Table 4.2 the performance summary of the different mechanisms is shown. Kyber512 is the best performing mechanism on all aspects, memory usage, time of memory access, and CPU usage.

Table 4.2: Summary of the different cryptosystems performance.

KEM	RAM Usage (bytes)	Memory Access	Total CPU usage (ms)
LightSaber	994	50	255.35
Kyber512	18,528	65	204.04
NTRUhs2048509	18,080	70	8,598.91
NTRULPr653	14,064	90	58,149.07
FrodoKEM640	921,360	95	7,823.76

We can see as well that, although FrodoKEM is the third that accesses the least time the memory, it is the one that uses the most memory and the second slowest in CPU usage. So, by these two later variables, we can discard FrodoKEM as an option.

We are left with LightSaber, NTRU, and NTRU Prime. NTRU Prime is the second that uses the least memory, but the slowest in terms of CPU usage, and the one that accesses the most memory. The only favoring point for NTRU Prime is that it has security strength level 2. However, considering that we prefer the KEMs' empirical performance and the last two tests' results, we can discard this KEM.

So, the mechanisms to use are the following: NTRUhs2048509, Kyber512, and LightSaber. Now we proceed to test the KEMs on the prototype system with an existing crypto library and IoT protocol.

4.2 Testing the mechanisms on the IoT prototype

Now we present the results of the second set of tests, where we measure the Wi-Fi performance of the selected post-quantum mechanisms along side the elliptic curves.

4.2.1 Performance on the number of packets

We begin by presenting the number of packets transmitted during the connection. We present the maximum and mean value, the standard deviation, and the behavior through time. Figures 4.4 and 4.5, show the maximum and mean value for the number of packets transmitted in the connection.

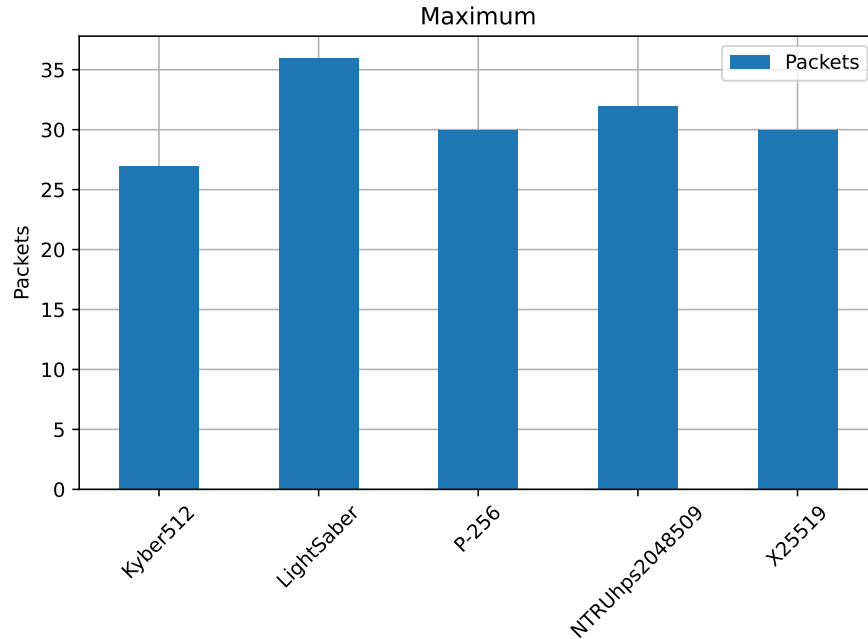


Figure 4.4: The maximum number of packets transmitted during the connection, from a total of 1,000 executions.

The KEM that transmits the least amount of packet is Kyber512, with a maximum number of 27 packets and a mean of 24 packets per connection. It is followed by the curves X95512 and P-256, both having a maximum of 30 packets and a mean of 27.92 and 27.93, respectively. The two that use the most packets are NTRU followed by LightSaber.

The curve P-256 presents the least deviation with 0.36, followed by the curve X95512 with 0.41, and finally Kyber512 with a value of 0.48. The last two are LightSaber with a value of 0.58 and NTRUhs2048509 with a value of 0.92.

Looking at the standard deviation, we can see that the elliptic curves are the best mechanisms; but in terms of the number of packets, Kyber512 outperforms the rest, with the elliptic curves in the middle and the other two post-quantum mechanisms at the end.

4.2.2 Performance on packet size

Now, we present the performance of the cryptosystems in terms of packet size. In Figure 4.7, we plot the maximum value in bytes of the packet, and in Figure 4.8, we show the mean value.

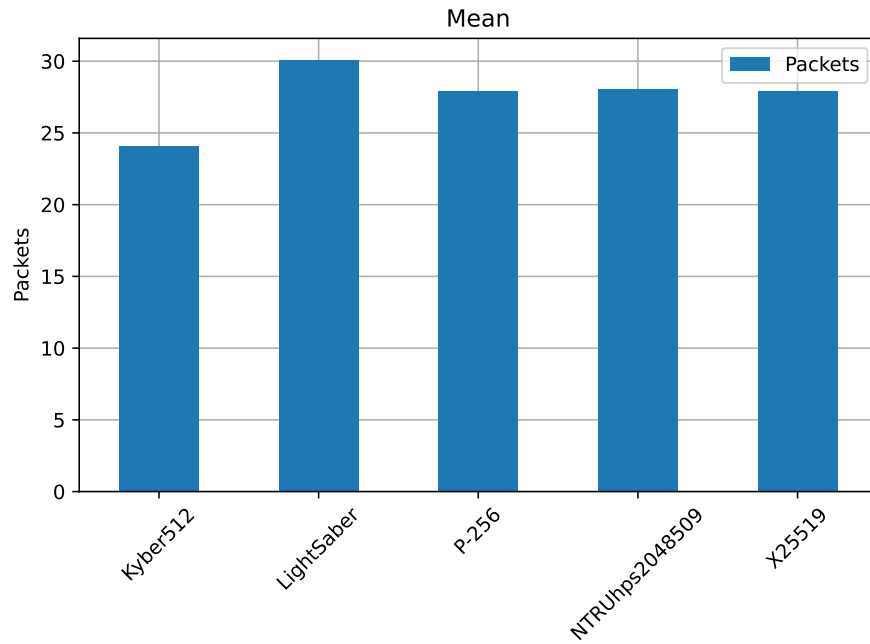


Figure 4.5: The mean number of packets transmitted during the connection, from a total of 1,000 runs.

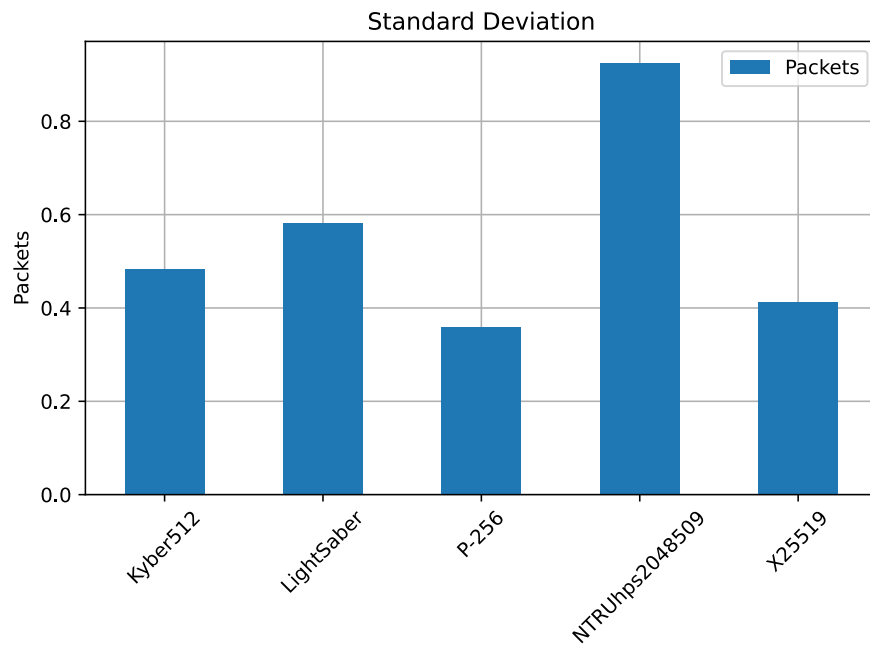


Figure 4.6: The standard deviation for the number of packets sent over a connection for each KEM. From this image we can know how much variation expect during the initial connection.

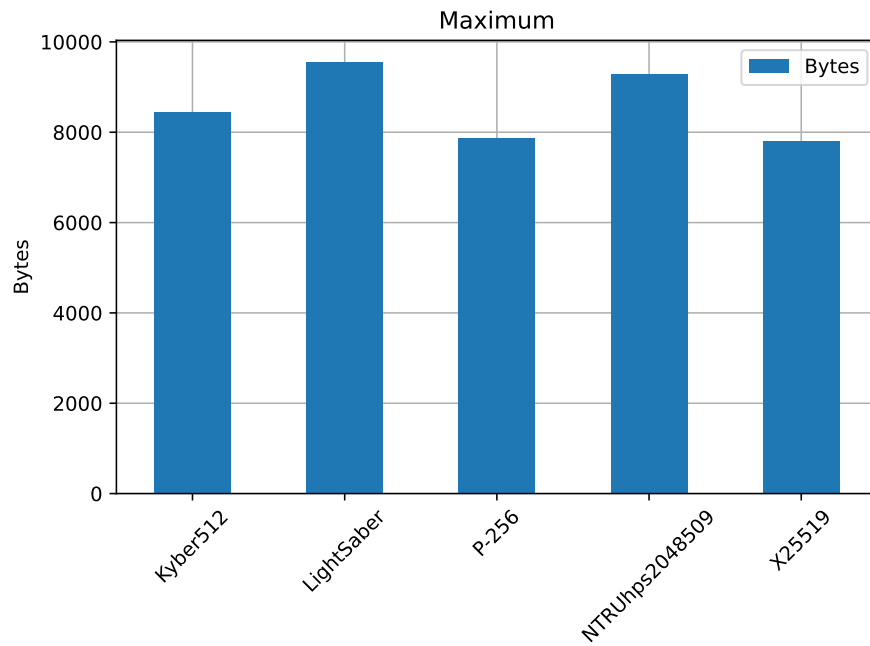


Figure 4.7: The maximum number of bytes transmitted during the connection, for each of mechanism involved.

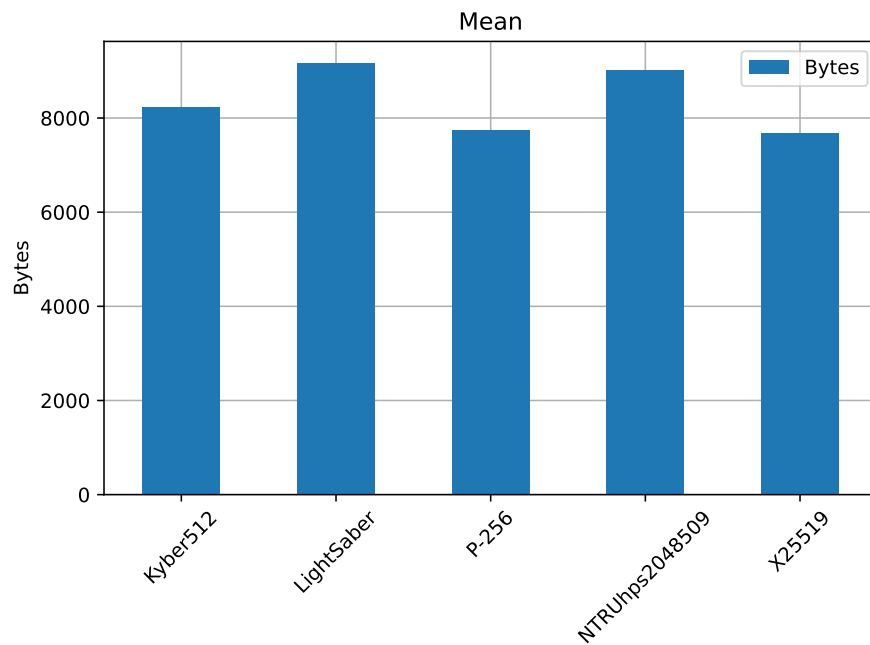


Figure 4.8: The average number of packets used for the connection. They present a similar behavior as the maximum number of packets, with the elliptic curves performing better than the post-quantum mechanisms.

We can see that the KEM whose packet uses the most bytes is LightSaber, followed by NTRUhs2048509. The KEM that uses the least amount of bytes is the elliptic curve X25519, followed by the curve P-256. The KEM Kyber512 falls in between the post-quantum mechanisms and the elliptic curves.

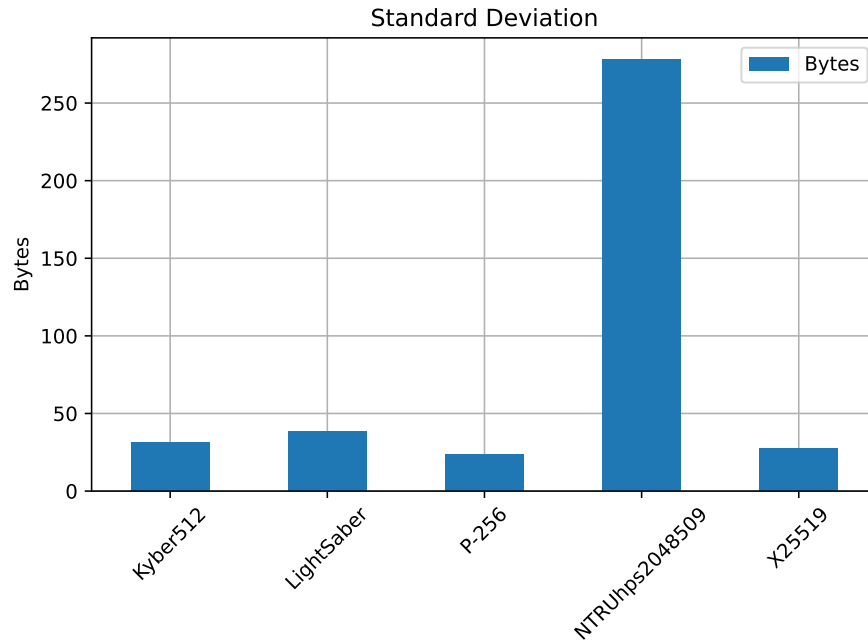


Figure 4.9: The standard deviation for the different mechanisms involved in terms of the number of bytes used per connection.

The standard deviation shows that the KEMs with the least deviation on the number of bytes are the elliptic curves, with a deviation of 23.5 for the curve P-256 and 26.2 bytes for the curve X25512, approximately. Then comes the KEMs Kyber512 with a deviation of 32.2 bytes and LightSaber with 38.5 bytes, approximately. Figure 4.9 shows the comparison between the standard deviation between the different KEMs. The one with the highest deviation is NTRUhs2048509.

In this analysis, the elliptic curves outperform the post-quantum cryptosystems, with the curve X95512 being the best. However, from the post-quantum KEMs, the Kyber512 KEM still outperforms the other two. Now we compare the duration of the connection for each mechanism.

4.2.3 Performance on connection's duration.

We present the performance results in terms of the connection's duration, beginning with the connection's maximum value, then the mean, and finally the standard deviation.

Figures 4.10 and 4.11 show the maximum and mean duration for each KEM. We can see that with the Kyber512 KEM, the handshake was the fastest, with a maximum duration of 2.9 milliseconds and a mean of 0.2 milliseconds. Next comes the KEMs P-256 and LightSaber, with a maximum value of 15.1 and 15.86 milliseconds and a mean

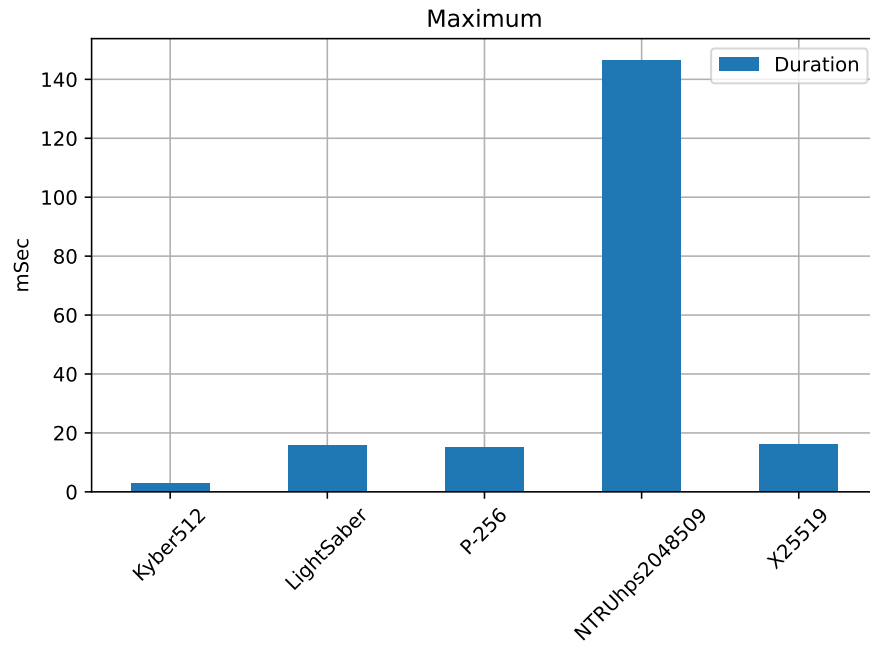


Figure 4.10: The maximum duration of the connection for each mechanism in use.

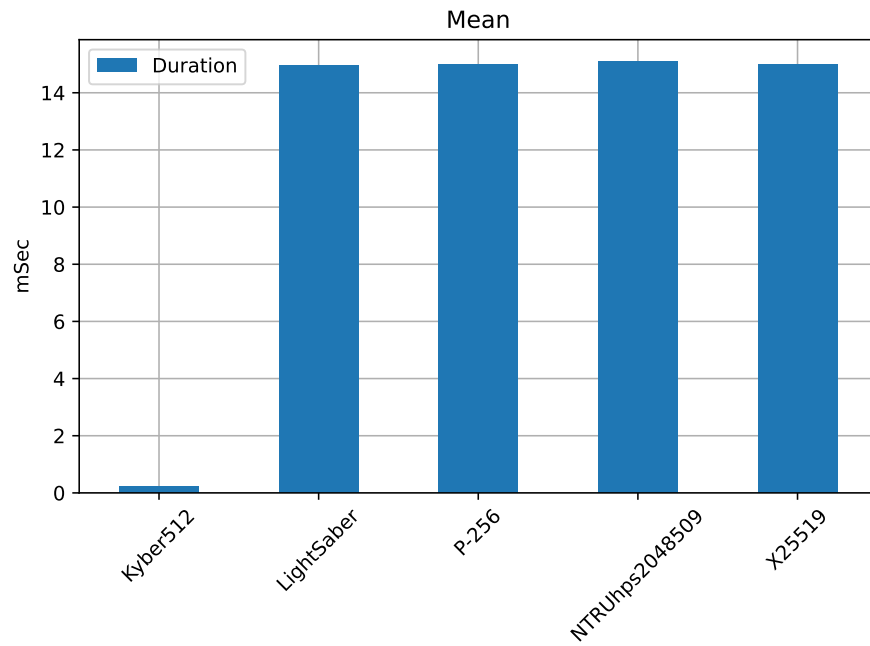


Figure 4.11: The mean value of the duratoin of the connection for all the mechanisms involved.

of 14.97 and 14.98 milliseconds. The duration is almost the same, especially considering the mean. Then comes the curve X25519 with a maximum value of 1.51 and a mean of 14.99 milliseconds. The mean value of the last mechanism is also very close to that of the curve P-256 and LightSaber. The KEM that takes the most amount of time is NTRUhs2048509, with a maximum of 146.49 and a mean of 15.1 milliseconds.

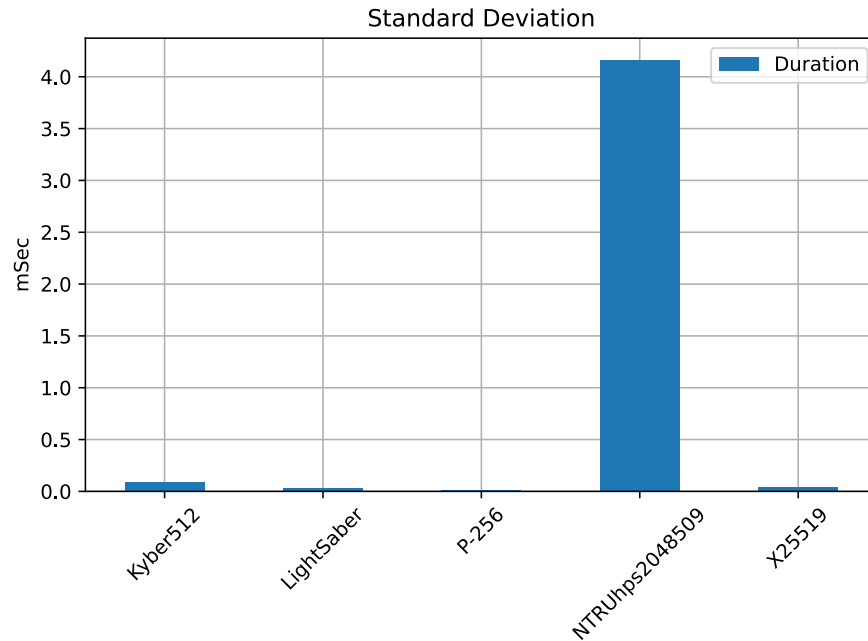


Figure 4.12: The standard deviation for the different KEMs involved in terms of the duration. The deviation for all the KEMs, except NTRUhs2048509, is minimal. We can expect a fairly constant duration for all the KEMs.

Considering the duration, we see some differences concerning the packet size. On the packet size, we saw that the elliptic curves outperformed the post-quantum mechanisms. Now, the KEM Kyber512 has the minimum duration, while it has the most deviation (after NTRU), while the curve P-256 has the least deviation but is slower in the handshake than Kyber512, with a difference of almost 12 milliseconds.

Considering the mean, maximum duration, and standard deviation, the key exchange method that best performs is Kyber512, as even considering the standard deviation, the maximum value it can reach is still less than the curve P-256.

Considering only the post-quantum mechanisms, Kyber512 still outperforms in practice the other two mechanism and even outperforms the classical KEMs on the handshake duration. On the number of bytes transmitted during the connection, the curves outperform the post-quantum mechanisms.

In the following, we will briefly discuss the theoretical strength of the different KEMs, give a final comparison between the classical and post-quantum mechanisms, and give some guidelines to select the proper mechanism for IoT devices.

4.3 Guidelines on selecting post-quantum KEM

Knowing the different cryptosystems' performance and theoretical strength, we can now present some guidelines to follow when using post-quantum cryptosystems in resource-constrained devices. We begin by presenting a summary of the theoretical strength and how the different cryptosystems performed.

From the selected KEMs in the first test series, we eliminated FrodoKEM640 and NTRULpr653, given that those use more resources in terms of CPU and memory usage. We kept Kyber512, NTRUhs2048509, and LightSaber. Both the elliptic curves and the post-quantum mechanisms have 128 bits security level and IND-CCA2 strength.

We saw as well that, in terms of CPU and memory usage, the two best performances are NTRUhs2048509 and LightSaber, followed by Kyber512. We present a summary of each KEM's performance in Table 4.3.

Table 4.3: Performance of the different KEMs in terms of CPU usage in milliseconds, and memory usage in number of bytes.

KEM	KeyGeneration	Encryption	Decryption	Memory usage
Kyber512	50.0715	67.6643	81.3896	18,528
LigthSaber	61.2086	84.3791	101.2356	994
NTRUhs2048509	7,618.4692	287.8924	684.0030	18,080

We now present a summary of the initial connection data and results. Recall that we studied the number of packets transmitted, their size in bytes, and the connection duration. In all of those, without considering the elliptic curves, the KEM Kyber512 outperformed the other two mechanisms.

Kyber512 outperformed NTRUhs2048509 and LightSaber, with the least amount of packets transmitted and the least deviation, followed by the KEMs NTRUhs2048509 and LightSaber. It also outperformed the elliptic curves. Table 4.4 presents a brief summary of the performance of the KEMs, in terms of the number of packets transmitted.

Table 4.4: A summary of the performance of the different cryptosystems in terms of the number of packets transmitted. On the top we present the best performing, that is, the one that uses the least amount of packets. The last is the worst performing.

KEM	Mean	Maximum	Standard Deviation
Kyber512	24.086	27	0.2326
X25519	27.926	30	0.4129
P-256	27.938	30	0.3579
NTRUhs2048509	28.05	32	0.9249
LightSaber	30.086	36	0.5818

In terms of the number of packets, we saw that the best performing is Kyber512, followed by NTRUhs2048509 and the LightSaber. In this case, the elliptic curves outperformed the post-quantum KEMs. Table 4.5 presents a summary of the KEMs' performance in terms of bytes per connection.

Table 4.5: This table summarizes the performance of the different packets in terms of size in bytes. The best performing is on the top, and the worst on the bottom.

KEM	Mean	Maximum	Standard Deviation
X25519	7,681.331	7,807	27.1832
P-256	7,748.311	7,876	23.5715
Kyber512	8,236.481	8,442	31.2484
LightSaber	9,168.695	9,556	38.4691
NTRUhs2048509	9,016.969	9,284	278.1027

In the final summary we provide the duration of the connection. Recall that all the connections had a constant behavior with a small amount of deviation. Kyber512 still outperformed the other mechanisms, including the elliptic curves, with a duration around 2 ms. The other had a very similar mean, except for NTRUhs2048509. Table 4.6 presents a summary of the connection's duration.

Table 4.6: A summary of the duration of the connection for each KEM. It is shown the mean, maximum and standard deviation for each KEM.

KEM	Mean	Maximum	Standard Deviation
Kyber512	0.2133	2.8836	0.0931
P-256	14.9884	15.1039	0.0147
LightSaber	14.9727	15.8695	0.0353
X25519	14.9901	16.2665	0.0455
NTRUhs2048509	15.1022	146.4901	4.1642

Now we give some guidelines to select the appropriate post-quantum mechanism for devices with low resources. Table 4.7 summarizes the results from the tests done on the usage of the different resources, and Table 4.8 summarizes the theoretical aspects of the cryptosystems.

We are mainly interested in the performance aspects of the KEMs and, considering that the components that most consume energy on resource-constrained devices are the CPU and the radios, we will consider the following hierarchy when giving a guideline on selecting a KEM: first the CPU and Wi-Fi performance, then memory usage, and finally the theoretical aspects of the ciphers.

Depending on the constraints, several post-quantum cryptosystems could be chosen. If a minimal usage of the Wi-Fi and the CPU is the objective, Kyber512 would be the option, as it gives us the fastest connection and minimum CPU usage, without considering the elliptic curves. It also has the smallest private key of all. As a disadvantage, we can see that Kyber512 has a greater public-key size than LightSaber and NTRUhs2048509, and it is vastly outperformed in terms of memory usage by LightSaber, and slightly by NTRUhs2048509, it is even outperformed by NTRULPr652.

If minimal memory usage is the goal, LightSaber would be the best option, as it outperforms both Kyber512 and NTRUhs2048509. It does not present an advantage in terms of Wi-Fi usage having similar values to NTRUhs2048509 and greater than Ky-

ber512. In terms of CPU usage, it is slower than Kyber512 and has a similar value to NTRUhs2048509. LightSaber also has the smallest public-key size than the others.

From the three mechanisms used in the IoT prototype, NTRUhs2048509 is the worst performing, so it is recommended to use it only when no other alternatives are left. NTRULPr653 is also an excellent option to consider, as in terms of CPU and memory usage performs similar to Lightsaber and NTRUhs2048509, its main disadvantage is that it is an alternative candidate and it would take longer to reach the status *standard*. FrodoKEM640 is completely ruled out, as the amount of memory and CPU it requires is far more significant than the other KEM, exceeding the typical resources available for a resource-constrained device.

Table 4.9 presents each KEM's advantages and disadvantages and some guidelines for selecting it.

When the device has restrictions in resources, especially in energy, use the KEM Kyber512, which is the best overall. Its disadvantages are that it uses more memory than the other two post-quantum mechanisms. Use LightSaber when the memory is the most limiting factor, as it uses the least. Use it if the energy constraints can be traded off with the resource utilization. Again, use NTRUhs2048509 when no other options are left. The only advantage is that it uses less Wi-Fi than LightSaber, but the difference is minimal.

Consider that these cryptosystems are not yet an approved standard, so usage in production environments is not recommended yet. For this reason, it is recommended to currently make use of the elliptic curves for the handshake, considering as well that there is more work on optimizations for elliptic curves for resource-constrained devices than there is for the post-quantum mechanisms. Also, the standardization process has not concluded yet, so the authors of the different KEMs might propose future optimizations, changing the performance data for each KEM.

Table 4.7: Summary of the implementation performance of the key exchange mechanisms. We show memory and CPU usage, and bytes, packets and the duration of the connection.

Cryptosystem	Memory Usage (bytes)	CPU Usage Total (ms)	Bytes per connection (mean)	Packets per connection (mean)	Connection's Duration (mean)
Kyber512	18,528	204.0401	8,236	24	0.2133
LightSaber	994	255.3597	9,168	30	14.9727
NTRUhs2048509	18,080	8,598.9120	9,016	28	14.9884
NTRULPr653	14,064	58,149.07331	N/A	N/A	N/A
FrodoKEM640	921,360	7,823.7653	N/A	N/A	N/A
P-256	N/A	N/A	7,748	28	14.9884
X25519	N/A	N/A	7,681	28	14.9901

Table 4.8: Summary of the theoretical aspects of the key exchange mechanisms considered so far, including key size, security level in bits, and theoretical strength.

Cryptosystem	Theoretical Strength	Security Level (bits)	Key Size (Public/Private)
Kyber512	IND-CCA	128	1,623/800
LightSaber	IND-CCA2	128	672/992
NTRUhs2048509	IND-CCA2	128	699/935
NTRULPr652	IND-CCA2	192	897/1,125
FrodoKEM640	IND-CCA2	128	9,616/19,888
P-256	IND-CCA2	128	256
X25519	IND-CCA2	128	256

Table 4.9: General guidelines for selecting the appropriate post-quantum cryptosystem for resource-constrained devices, according to its performance and security.

KEM	Advantages	Disadvantages	Guidelines
Kyber512	<ul style="list-style-type: none"> Minimal usage of CPU Minimal usage of Wi-Fi Smallest private key Fastest handshake Finalist in the NIST process 	<ul style="list-style-type: none"> Uses the most memory Greater public-key size 	<ul style="list-style-type: none"> Use for minimal energy requirement Use for fast computing and handshake Use when sufficient memory available
LightSaber	<ul style="list-style-type: none"> Strong security Smaller public-key size Finalist in the NIST process 	<ul style="list-style-type: none"> Greater Wi-Fi usage Greater CPU usage 	<ul style="list-style-type: none"> Use when little memory is available Use when there is sufficient energy available Use when strong security can be traded with energy and resource requirements
NTRUhs2048509	<ul style="list-style-type: none"> Strong security Smaller private-key size Minimal usage of memory Finalist in the NIST process 	<ul style="list-style-type: none"> Worst performing overall 	<ul style="list-style-type: none"> Use when no other available

Chapter 5

Conclusions

The cryptosystems currently used for key exchange in the industry, were not designed with resource limitation in mind, making them hard to adapt to resource-constrained devices, requiring a significant amount of computational resources. For that reason, and considering that the Internet of Things is increasingly becoming more popular, we considered necessary to study the performance of existing post-quantum cryptosystems and how they might impact on devices with low resources.

We chose to study the cryptosystems from the United States National Institute of Standards standardization process, as those will be deployed to industry and used by most users that connect to the Internet. From those available at the process, we choose the lattice-based ones, as those, according to the literature, are more suitable for resource-constrained and Internet of Things devices. From these lattice-based cryptosystems, we chose those with the smaller key size, because they require the least amount of computational resources.

Knowing that the most critical components in a resource-constrained device are CPU and Wi-Fi, from an energy point of view, and memory from a software standpoint of view, we proceeded to measure the performance of such variables on a device that can be considered on the edge of resource-constrained devices, a Raspberry Pi. It does not have as many resources as PC or laptop, but has more than other Internet of Things devices such as an Arduino or a PIC microcontroller. This hardware still gave us an idea of how the cryptosystems behave on devices with low capabilities and present some guidelines for selecting an appropriate one for such devices.

We saw that the cryptosystem that performs the best in terms of CPU and Wi-Fi usage is Kyber512, while LightSaber performs the best in memory consumption. For testing the cryptosystem on an Internet of Things prototype, we present data on the elliptic curves' performance. Such data allowed us to compare the possible future standards with an existing one, giving us a reference point. Although in some cases the elliptic curves performed better than the post-quantum key exchange mechanisms, in general, we saw a similar performance among the two types, and, in the case of Kyber512, it outperformed the elliptic curves.

5.1 Future Work

Except for Kyber512, we saw a similar performance between the elliptic curves and the post-quantum key exchange mechanisms, using the implementation included within OpenSSL for the elliptic curves and the implementation by the authors to the standardization process. The results could indicate the need to make more optimizations to such key exchange mechanisms, so they are more adaptable to devices with low resources.

We considered only those cryptosystems based on lattices as those are more suitable for Internet of Things devices, but remains to study the performance of the lowest security versions of the code-based cryptosystems and possibly find a way to adapt them to resource-constrained devices, in case the lattice-based ones are proved to be insecure to quantum computers attacks in the future. A study on the performance of all the versions and available key exchange mechanisms can be done as well.

Work on porting the post-quantum cryptosystems to hardware implementations could also be done as a component of a System on Chip, thus freeing resources to the software and possibly reducing the energy requirements. Research is needed to know which type of quantum-safe cryptosystems are more easily portable to hardware implementations. Whether the implementation can reduce the energy consumption, and if it is possible to have a version with the strongest security settings (level 5 security) while at the same time keeping or even reducing the energy required by their software counterparts considered in this work.

Once quantum computers come to existence, quantum-safe cryptosystems will be the norm, as all classical cryptosystems will become obsolete, so porting such cryptosystems to other areas of cryptography will be necessary. We know that only doubling the key size for symmetric-key cryptography will be sufficient for protecting the different ciphers available. For other areas that use public-key cryptography it is worth start porting such cryptosystems to it. One such example is homomorphic encryption, which currently uses elliptic curves. There is work on homomorphic encryption using lattice-based cryptosystems, but again, work for code-based cryptosystems is also essential to be considered, given that we have no formal proof of the safeness of lattice-based cryptosystems.

Appendices

Appendix A

Basic Mathematics for Cryptography

The cryptosystems presented here are built on algebraic structures, which are the union of a set A and binary operations "+" that work over the elements of the set. It also defines a set of rules, called axioms, which indicate how the binary operator should work over the set elements. The binary operation is denoted usually by "+" and is a map from the set to itself:

$$+ : A \times A \rightarrow A \quad (\text{A.1})$$

The most basic algebraic structure is known as a group, and we define it in the following. We also introduce the concept of integer rings, fields, and lattices.

A.1 Groups

Given a set G and a binary operator (+), a group is defined as $\{G, +\}$ and has the following properties. Given elements $a, b, c \in G$, we have that:

1. *Closure*: $a + b \in G$. That is, the result of applying the binary operator to any two elements of G , is still in G .
2. *Associative*: $(a + b) + c = a + (b + c) \in G$.
3. *Identity*: There exists an element $0 \in G$, such that $0 + a = a + 0 \in G, \forall a \in G$, known as the identity element of G .
4. *Inverse of elements*: $\forall a \in G$, there exists $-a \in G$ such that $a + (-a) = (-a) + a = 0$.

These are the basic properties that a group should hold to be considered one. Another property is:

5. *Commutative*: $\forall a, b \in G$, we have that $a + b = b + a \in G$.

If a group also has that property, it is said to be an abelian group.

Also, if an abelian group has exponentiation, there is an element $g \in G$ from which all other elements can be generated, then the group is called a cyclic group. The element g is called the generator.

From an abelian group, another mathematical structure can be constructed with the use of a second binary operation: a ring.

A.2 Integer Rings

Given a set G and two binary operations $(+)$ and (\times) , the set $\{G, +, \times\}$ is said to be a ring, if it is an abelian group under the operation $(+)$, and has the following properties for the (\times) :

6. *Closure*: $a \times b \in G$.
7. *Associativity*: $(a \times b) \times c = a \times (b \times c) \in G$.
8. *Distribution*: $a \times (b + c) = a \times b + a \times c \in G$.

An optional property under operation (\times) is:

9. *Commutative*: $a \times b = b \times a$.

From an integer ring, a third mathematical structure can be created, and is known as a field.

A.3 Fields

Given an integer ring $\{G, +, \times\}$ that is commutative under the operation (\times) , it is said to be a field, if it has as well the following property:

10. *Inverse*: For all $a \in G \setminus \{0\}$, there exists a^{-1} such that $a \times (a^{-1}) = (a^{-1}) * a = 0$

In the following, we introduce another mathematical structure known as *lattice*, which is used for the construction of cryptosystems suitable to the IoT.

A.4 Lattices

A lattice is a subset of the vector space \mathbb{R}^m ; a collection of vectors constructed in a particular way.

Let $\{\mathbf{b}_1, \dots, \mathbf{b}_n\}$ be a linearly independent set of (row) vectors in \mathbb{R}^m , with $n \leq m$. The lattice generated by $\{\mathbf{b}_1, \dots, \mathbf{b}_n\}$ is the set

$$L = \left\{ \sum_{i=1}^n l_i \mathbf{b}_i : l_i \in \mathbb{Z} \right\} \quad (\text{A.2})$$

of integer linear combinations of the b_i . The vectors $\{\mathbf{b}_1, \dots, \mathbf{b}_n\}$ are called a basis of the lattice.

These are the basic mathematical structures from which several others are created, such as the vector fields. Some cryptosystems are constructed using only the group structure, e.g., the RSA cryptosystem. Others make use of more complex mathematical structures like lattices or codes.

Some instantiations of the structures mentioned above used for the construction of cryptosystems include the Elliptic and Hyperelliptic Curves, which is a group constructed from equations describing certain types of curves.

Appendix B

List of acronyms

Computers Structures terms.

- **CPU** – Central Processing Unit.
- **RAM** – Random Access Memory.

Cryptography

- **CPA** – Passive Attack/Chosen Plain Text Attack.
- **CCA1** – Lunchtime Attack/Chosen Ciphertext Attack.
- **CCA2** – Adaptive Chosen Ciphertext Attack.
- **DH** – Diffie-Hellman key exchange mechanism.
- **IND** – Indistinguishability.
- **KEM** – Key Exchange Mechanism.
- **MAC** – Message Authentication Code.
- **OWE** – One-way Encryption.
- **PKC** – Public-key Cryptography.
- **RSA** – Ravist-Shamir-Adleman cryptosystem.

Hardware protocols related terms.

- **SPI** – Serial Peripheral Interface.
- **UART** – Universal Asynchronous Receiver/Transceiver.
- **USB** – Universal Serial Bus.

Institutes and Businesses.

- **ETSI** – European Telecommunication Standard Institute.

-
- **IETF** – Internet Engineering Task Force.
 - **IBM** – International Business Machines.
 - **NIST** – National Institute of Standards.

Internet of Things terms.

- **AMQP** - Advance Messaging Query Protocol.
- **CoAP** – Constrained Application Protocol.
- **DDS** – Data Distribution Service.
- **IoT** – Internet of Things.
- **MQTT** – Message Queue Telemetry Transport.
- **RTOS** – Real-Time Operating System.
- **WSN** – Wireless Sensors Network.
- **XMPP** – Extensible Messaging and Presence Protocol

Internet-related terms.

- **DTLS** – Datagram TLS.
- **IP** – Internet Protocol.
- **RFC** – Request for Comments.
- **SSL** – Secure Socket Layer.
- **TCP** – Transport Control Protocol.
- **TLS** – Transport Layer Security.
- **UDP** – User Datagram Protocol.

Wireless technologies.

- **NFC** – Near Field Communication technology.
- **RFID** – Radio Frequency Identifier.
- **Wi-Fi** – Wireless Fidelity.

Other.

- **CSV** – Comma Separate Values.

Bibliography

- [1] Deutch D. “Quantum theory, the Church-Turing principle and the universal quantum computer.” In: *Proceedings of the Royal Society of London* 400 (1985), pp. 97–117. DOI: <https://doi.org/10.1098/rspa.1985.0070>.
- [2] T. Güneysu and T. Oder. “Towards lightweight Identity-Based Encryption for the post-quantum-secure Internet of Things”. In: *2017 18th International Symposium on Quality Electronic Design (ISQED)*. IEEE. 2017, pp. 319–324. DOI: [10.1109/ISQED.2017.7918335](https://doi.org/10.1109/ISQED.2017.7918335).
- [3] M. O. Saarinen. “Ring-LWE ciphertext compression and error correction: Tools for lightweight post-quantum cryptography”. In: *Proceedings of the 3rd ACM International Workshop on IoT Privacy, Trust, and Security*. 2017, pp. 15–22. DOI: <https://doi.org/10.1145/3055245.3055254>.
- [4] T. M. Fernández-Caramés. “From pre-quantum to post-quantum IoT security: A survey on quantum-resistant cryptosystems for the Internet of Things”. In: *IEEE Internet of Things Journal* 7.7 (2019), pp. 6457–6480. DOI: [10.1109/JIOT.2019.2958788](https://doi.org/10.1109/JIOT.2019.2958788).
- [5] Singh A., Payal A., and Bharti S. “A walkthrough of the emerging IoT paradigm: Visualizing inside functionalities, key features, and open issues.” In: *Journal of Network and Computer Applications* 143 (2019), pp. 111–151. DOI: <https://doi.org/10.1016/j.jnca.2019.06.013>.
- [6] Al-Fuqaha A. et al. “Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications”. In: *IEEE Communication Surveys and Tutorials* 17.4 (2015). DOI: [10.1109/COMST.2015.2444095](https://doi.org/10.1109/COMST.2015.2444095).
- [7] Atzori L., Iera A., and Morabito G. “The Internet of Things: A survey”. In: *Computer Networks* 54 (2010), pp. 2787–2805. DOI: <https://doi.org/10.1016/j.comnet.2010.05.010>.
- [8] C. M. Roberts. “Radio frequency identification (RFID)”. In: *Computers & security* 25.1 (2006), pp. 18–26. DOI: <https://doi.org/10.1016/j.cose.2005.12.003>.
- [9] Akyildiz I.F., Sankarasubramaniam Y., and Cayirci E. “Wireless sensor networks: a survey”. In: *Computer Networks* 38 (2002), pp. 393–422. DOI: [https://doi.org/10.1016/S1389-1286\(01\)00302-4](https://doi.org/10.1016/S1389-1286(01)00302-4).

-
- [10] Mosenia A. and Jha N.J. “A Comprehensive Study of Security of Internet-of-Things”. In: *IEEE Transactions on Emerging Topics in Computing* 5 (2016), pp. 2168–6750. DOI: 10.1109/TETC.2016.2606384.
- [11] Kouicem D. E., Bouabdallah A., and Lakhlef H. “Internet of thing security: A top-down survey”. In: *IEEE Transactions on Emerging Topics in Computing* 5 (2016), pp. 2168–6750. DOI: <https://doi.org/10.1016/j.comnet.2018.03.012>.
- [12] Whitfield D. and Hellman M. E. “New Directions in Cryptography”. In: *IEEE Transactions on Information Theory* IT-22 (6) (1976), pp. 644–654. DOI: 10.1109/TIT.1976.1055638.
- [13] Rivest R.L., Shamir A., and Adleman L. “A Method for Obtaining Digital Signatures and Public-Key Cryptosystems”. In: *Communications of the ACM* 21 (2) (1978), pp. 120–126. DOI: <https://doi.org/10.1145/359340.359342>.
- [14] S. D. Galbraith. *Mathematics of Public Key Cryptography*. Ed. by Cambridge University Press. Cambridge, 2018.
- [15] Shor P. “Algorithms for quantum computation: discrete logarithms and factoring”. In: vol. 143. IEEE, 1994, pp. 124–134. DOI: 10.1109/SFCS.1994.365700.
- [16] D. J Bernstein and T. Lange. “Post-quantum cryptography”. In: *Nature* 549.7671 (2017), pp. 188–194. DOI: <https://doi.org/10.1038/nature23461>.
- [17] R. J. McEliece. “A public-key cryptosystem based on algebraic”. In: *Coding Thv* 4244 (1978), pp. 114–116. DOI: 19780016269.
- [18] J. Hoffstein, J. Pipher, and J. H. Silverman. “NTRU: A ring-based public key cryptosystem”. In: *International Algorithmic Number Theory Symposium*. Springer, 1998, pp. 267–288. DOI: 10.1007/BFb0054868.
- [19] O. Regev. “On lattices, learning with errors, random linear codes, and cryptography”. In: *J. ACM* 56.6 (2009). DOI: 10.1145/1568318.1568324.
- [20] *Post-Quantum Cryptography*. <https://csrc.nist.gov/Projects/post-quantum-cryptography/Post-Quantum-Cryptography-Standardization>. [Online; last access: July-2020].
- [21] et al. Alagic Gorjan Alperin-Sheriff Jacob. “Status Report on the Second Round of the NIST Post-Quantum Cryptography Standardization Process”. In: 1.1 (2020). DOI: <https://doi.org/10.6028/NIST.IR.8309>.
- [22] Bernstein D. J. et al. Albrecht M. R. “Classic McEliece: conservative code-based cryptography”. In: *Post-Quantum Cryptography Standarization Process* (2020).
- [23] Avanzi R., Bos J., and et al. Ducas L. “CRYSTAL-Kyber. Algorithm Specification And Supporting Documentation”. In: *Post-Quantum Cryptography Standarization Process* (2019).
- [24] Cheng C., Danba O., and Hoffstein J. et al. “NTRU. Algorithm Specification and Documentation”. In: *Post-Quantum Cryptography Standarization Process* (2019).
-

- [25] D’Anvers J.-P. et al. “SABER: Mod-LWR based KEM (Round 2 Submission)”. In: *Post-Quantum Cryptography Standarization Process* (2019).
- [26] Barreto P. S. L. M. et al. Aragon N. “BIKE: Bit Flipping Key Encapsulation”. In: *Post-Quantum Cryptography Standarization Process* (2020).
- [27] et al. Melchor C. A. Aragon N. “Hamming Quasi-Cyclic (HQC)”. In: *Post-Quantum Cryptography Standarization Process* (2020).
- [28] et al. Azarderakhsh R. Campagna M. “Supersingular Isogeny Key Encapsulation”. In: *Post-Quantum Cryptography Standarization Process* (2020).
- [29] Alkim E., Bos J. W., and Ducas L. et al. “FrodoKEM. Learning With Errors Key Encapsulation. Algorithm Specification And Supporting Documentation”. In: *Post-Quantum Cryptography Standarization Process* (2020).
- [30] Bernstein D. J. et al. “NTRU-prime: round 2”. In: *Post-Quantum Cryptography Standarization Process* (2019).
- [31] *The Transport Layer Security (TLS) Protocol Version 1.3*. <https://tools.ietf.org/html/rfc8446>. [Online; last access: September 23, 2020].
- [32] *The OpenSSL Library*. <https://www.openssl.org/>. [Online; last access: September 23, 2020].
- [33] *Java Secure Socket Extension (JSSE) Reference Guide*. <https://docs.oracle.com/en/java/javase/11/security/java-secure-socket-extension-jsse-reference-guide.html#GUID-93DEEE16-0B70-40E5-BBE7-55C3FD432345>. [Online; last access: September 23, 2020].
- [34] *The gnuTLS Transport Layer Security Library*. <https://www.gnutls.org/>. [Online; last access: September 23, 2020].
- [35] *BoringSSL*. <https://boringssl.googlesource.com/boringssl/>. [Online; last access: September 23, 2020].
- [36] *LibreSSL*. <https://www.libressl.org/>. [Online; last access: September 23, 2020].
- [37] *Mbed TLS*. <https://tls.mbed.org/>. [Online; last access: September 23, 2020].
- [38] *WolfSSL*. <https://www.wolfssl.com/>. [Online; last access: September 23, 2020].
- [39] R. Chaudhary, Singh Aujla G., and Zeadally S. Kumar N. “Lattice-Based Public Key Cryptosystem for the Internet of Things Environment: Challenges and Solutions”. In: *IEEE Internet of Things Journal* 6 (3) (2019), pp. 4897–4909. DOI: 10.1109/JIOT.2018.2878707.
- [40] V. Lyubahsevsky, Piekert C., and Oded R. “On Ideal Lattices and Learning With Errors over Rings”. In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2010, pp. 1–23.

- [41] *The RadioHead Library*. <https://www.airspayce.com/mikem/arduino/RadioHead/index.html>. [Online; last access: September 23, 2020].
- [42] *DHT Sensor Library*. <https://github.com/adafruit/DHT-sensor-library>. [Online; last access: September 23, 2020].
- [43] A. Carroll and G. Heiser. “An analysis of power consumption in a smartphone.” In: *USENIX annual technical conference*. Vol. 14. Boston, MA. 2010, pp. 21–21.
- [44] *Valgrind User Manual*. <https://valgrind.org/docs/manual/manual.html>. [Online; last access: September 23, 2020].
- [45] *Wireshark User Manual*. https://www.wireshark.org/docs/wsug_html_chunked/. [Online; last access: September 23, 2020].



Figure 5.1: SARS-CoV-2: Algo microscópico fue capaz de detener el mundo de forma abrupta, cambiando la forma en que vivimos y nos relacionamos. Es una lección sobre la forma moderna de vivir, ¿aprenderemos de ella?