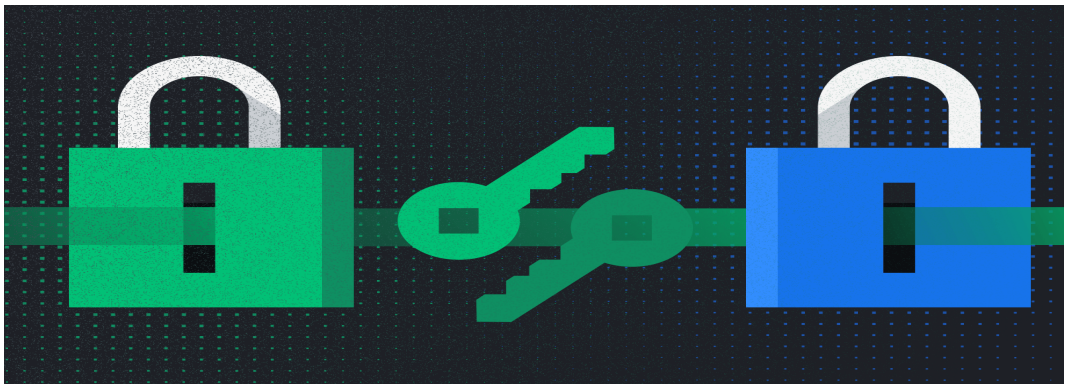
INTERNATIONAL HELLENIC UNIVERSITY

FACULTY OF ENGINEERING

DEPARTMENT OF INFORMATION AND ELECTRONIC ENGINEERING

THESIS PROJECT

«Study and development of encryption mechanisms for end-to-end 5G IoT hybrid platforms»



Student
Klodian Bardhi
Reg. Number: 174926

Supervisor
Periklis Chatzimisios
Professor

September 15, 2021

Thesis Title: Study and development of encryption mechanisms for end-to-end 5G IoT hybrid platforms
Thesis Number: 21222
Student Name: Klodian Bardhi
Supervisor Name: Periklis Chatzimisios
Undertaking Date: 30/03/2021
Completion Date: 15/09/2021

I certify that I am the author of this thesis and that any assistance I have received while preparing it is fully acknowledged and mentioned in the following sections. I have also listed any sources from which I have used data, ideas, images and text, whether they are referenced as is or paraphrased. In addition, I certify that this thesis was prepared by me personally, especially as a diploma thesis, in the Department of Information and Electronic Engineering of IHU.

The approval of this thesis by the Department of Information and Electronic Engineering of the International Hellenic University does not necessarily imply acceptance of the author's views by the Department.

«Dedicated to my family, my partner and my friends for always being by my side»

Prologue

During my college years in International Hellenic University of Thessaloniki I went through a number of courses that were very interesting and helped me develop my skills as a future software developer. However, once I was introduced to security and the IoT, I was immediately attached to them. Therefore I decided that my thesis would be based on one of those topics or better yet, a combination of the security and IoT which is a very trending and compelling concept.
Moreover, this thesis is set to explore and examine the security aspects of IoT platforms, more specifically the end-to-end encryption of 5G IoT infrastructures. The technical part of this thesis mainly focuses on the creation of an MEC IoT platform and applying encryption best practices, which I learned from the my research into the topic, ensuring end-to-end secure communication. I have constructed the thesis in a such a manner that it would first help the reader understand the main principles of security through the detailed literature review and then understand how the theory can be applied in real 5G MEC application. In general, understanding and being able to deploy an encryption mechanism for IoT purposes is not an easy task, as you have to keep in mind that the encryption needs to both lightweight and strong enough to uphold most, if not all, security threats. This research has been very valuable and informative for me and has lead me to an interesting career path of security which I am very keen on. I hope that through this thesis I can pass the same amount of information and make the reader interested in the topic.

«Study and development of encryption mechanisms for end-to-end 5G IoT hybrid platforms»

«Klodian Bardhi»

Abstract

In the emerging field of Internet of Things (IoT) we observe new and innovative technologies being implemented such as the use of 5G, which compared to previous telecommunication protocols, offers a more effective and fast communication. However, the area of Information Technology (IT) has always been targeted by malevolent attackers attempting to redeem or destroy what is valuable to the system. The rapid evolution of IoT and 5G opens up new weak spots and and room for malicious attacks to go through. Therefore it is very important to keep the system secure with security modules. One of these modules is the encryption mechanism, which protects data while they are stored or even when being transmitted over the communication channels. Once encryption is combined with other security methods, such as authentication and authorization, the system becomes robust and impenetrable. In this thesis I am going to review the various encryption mechanisms that already exist, analyze them, find the best solution and then implement it to a real 5G IoT platform that was also created from scratch. The encryption mechanism is based on the HTTPS protocol and embeds encyption's best practices while also maintaining a lightweight profile hence it can be configured inside IoT devices, which are typically low powered sensors. The proposed encryption method is later tested on a 5G testbed which includes IoT devices, the proposed framework and the use of real 5G spectrum. With these experiments I can determine the amount of impact that encryption had to the system and the overall performance of the infrastructure, making improvements wherever is possible. The overall scope of this research is to create an end-to-end secure communication based on the 5G IoT platform, encrypting every possible component of the communication process between the two ends.

«Μελέτη και ανάπτυξη μηχανισμών κρυπτογράφησης για από άκρο σε άκρο υβριδικές πλατφόρμες 5G IoT »

«Κλοντιάν Μπάρδι»

Περίληψη

Στον ολοένα αναπτυσσόμενο τομέα του Διαδικτύου των Πραγμάτων (αγγλικά Internet of Things ή IoT) παρατηρούμε ότι εφαρμόζονται συνεχώς νέες και καινοτόμες τεχνολογίες όπως η χρήση του 5G, το οποίο σε σύγκριση με τα προηγούμενα τηλεπικοινωνιακά πρωτόκολλα, προσφέρει μια πολύ πιο αποτελεσματική και γρήγορη επικοινωνία. Ωστόσο, ο τομέας της πληροφορικής ήταν πάντα στόχος κακόβουλων επιθέσεων που προσπαθούσαν να υποκλέψουν ή να καταστρέψουν ό, τι είναι πολύτιμο για το σύστημα. Η ταχεία εξέλιξη του IoT και του 5G εμφανίζεται νέα αδύναμα σημεία και δίνει περισσότερο χώρο στον επιτιθέμενο για κακόβουλες επιθέσεις. Επομένως, είναι πολύ σημαντικό να διατηρείται το σύστημα ασφαλές χρησιμοποιώντας μεθόδους ασφαλείας. Μία από αυτές τις μεθόδους είναι ο μηχανισμός κρυπτογράφησης, ο οποίος προστατεύει τα δεδομένα όταν αυτά παραμένουν αποθηκευμένα ή ακόμα και όταν μεταδίδονται μέσω των καναλιών επικοινωνίας. Μόλις η κρυπτογράφηση συνδυαστεί με άλλες μεθόδους ασφαλείας, όπως ο έλεγχος ταυτότητας και η εξουσιοδότηση, το σύστημα γίνεται ισχυρό και αδιαπέραστο. Σε αυτή τη διατριβή πρόκειται να αναθεωρήσω τους διάφορους μηχανισμούς κρυπτογράφησης που υπάρχουν, να τους αναλύσω, να βρω την καλύτερη λύση και στη συνέχεια να την εφαρμόσω σε μια πραγματική πλατφόρμα 5G IoT η οποία και αυτή είναι αυτοσχεδiαστη και πρωτότυπη. Ο μηχανισμός κρυπτογράφησης βασίζεται στο πρωτόκολλο HTTPS και ενσωματώνει τις βέλτιστες πρακτικές της κρυπτογράφησης διατηρώντας παράλληλα ένα ελαφρύ προφίλ, επομένως μπορεί να διαμορφωθεί και σε συσκευές IoT, οι οποίες είναι συνήθως αισθητήρες χαμηλής ισχύος. Η προτεινόμενη μέθοδος κρυπτογράφησης δοκιμάζεται αργότερα σε εργαστήριο 5G το οποία περιλαμβάνει συσκευές IoT, την προτεινόμενη πρωτότυπη πλατφόρμα και τη χρήση πραγματικού φάσματος 5G. Με αυτά τα πειράματα μπορώ να καθορίσω το μέγεθος του αντίκτυπου που είχε η κρυπτογράφηση στο σύστημα και τη συνολική απόδοση της υποδομής, κάνοντας βελτιώσεις όπου είναι δυνατόν. Σκοπός αυτής της έρευνας είναι η δημιουργία μιας ασφαλούς απο άκρο σε άκρο σύνδεσης πάνω σε πλατφόρμα 5G IoT, κρυπτογραφώντας κάθε πιθανή εφαρμογή και εξάρτημα που περιέχονται στην διαδικασία της επικοινωνίας μεταξύ των δύο άκρων.

Dissertation Acknowledgements

Throughout the writing of this thesis, I had great influence and technical assistance and I am very proud that I had the chance to work and be supported by such wonderful people.
First, I would like to thank my supervisor Dr. Periklis Chatzimisios, who helped me shape the study topics and methods with the great experience he has in the subject. Your kind comments and feedback encouraged me to think deeper and improve my work.
I would like to acknowledge my colleagues and supervisors from my internship at University of Bristol for their great collaboration. Particularly, I would like to thank my supervisors there, Mr. Alex Mavromatis and Mr. Antonis T Vafeas. Thank you guys for always being helpful and always providing me with chances to further extend my studies and expand my overall knowledge & experience in the field of IT.
I would also like to show my gratitude to all my professors who taught me valuable lessons and guided me through my studies in college, providing me with all the knowledge and tools that are required by a future software developer.
Additionally, I would like to thank my parents and people close to me for their invaluable advice and moral support throughout my college years. You've always been there for me and had a huge impact on my life decisions for the best.
Finally, I would like of course to thank my colleague and close friend Savvas Mantzouranidis who provided me with technical, and more importantly, moral support throughout our college years, outside and during work.

Table of Contents

List of Figures

List of Tables

Abbreviations

3DES	Triple Data Encryption Standard
AES	Advanced Encryption Standard
API	Application Programming Interface
CoAP	Constrained Application Protocol
CSP	Content Security Policy
CSR	Certificate Signing Request
DB	Database
DES	Data Encryption Standard
DH	Diffie Hellman
DoS	Denial-of-Service
GIL	Global Interpret Lock
HSTS	HTTP Strict Transport Security
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IHU	International Hellenic University
IoT	Internet of Things
KPI	Key Performance Indicator
MEC	Mobile Edge Computing
MitM	Man-in-the-Middle
MiM	Man-in-Middle
MQTT	MQ Telemetry Transport
NFV	Network Function Virtualization
NIST	National Institute of Standards and Technology
o-RTT	Zero Round Trip Time
OCB	Orion Context Broker
OS	Operating System
PKC	Public-Key Cryptography
QoS	Quality of Service
RC4	Rivest Cipher 4
RDBMS	Relational Database Management System
RFID	Radio Frequency Identification
RTT	Round Trip Time
SCA	Side-Channel Attack
SDK	Software Development Kit
SIL	Smart Internet Labs
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
SQLi	Structured Query Language injection
SSL	Secure Sockets Layer
TLS	Transport Layer Security
UI	User Interface
UoB	University of Bristol
WAF	Web Application Firewall
XSS	Cross-site Scripting

Chapter 1: Introduction

As human evolution progresses through time, so do the amenities of a person that are used on a regular basis. Usually you would see people holding a smartphone, tablet or laptop which are connected to the Internet and share data with the world web. Imagine if other devices such as the car, fridge, container or even your living room couch could became ”smart”, in a meaning that they could also connect to a network and send or receive data. This is the vision for the Internet of Things (IoT) technology [1]. The single definition of the IoT does not yet exist as various research groups around the world have defined it differently. However, some common concepts can be gleaned from their work. IoT covers many aspects of life, from connected homes and cities to connected cars and roads along devices that track individual behavior and use the collected data for ”push” services. Someone mentioned that nearly trillion devices will be connected to the Internet by 2025, and that mobile phones are defined as the heart of applications that interconnect all these ”things”. In other words, what this means is that the IoT refers to the billions of physical devices around the world that are now connected to the Internet, all of which collect and share data. Thanks to the emergence of ultra-cheap computer chips and the ubiquity of wireless networks, everything from tiny pills to big airplanes can become part of the IoT. Connecting all these different objects and adding sensors to them adds a degree of digital intelligence to otherwise dumb devices, allowing them to transmit data in real time without involving humans. The IoT is making the structure of the world around us smarter and more responsive, integrating the digital world and the physical world [1, 2].

1.1 Background

However, With the development of the IoT, new security problems have emerged and traditional security problems have become more serious Moreover, data collecting devices can become the target of attackers. The main reason is the heterogeneity and large scale of the object. For example, the goal of an IoT application in an enterprise is to improve productivity and serviceability by collecting data from a large number of sensors installed in production equipment, analyzing it and managing it autonomously in real time. The process of sharing such a large amount of data starts with the device itself, which needs to securely communicate with the IoT platform. The platform integrates data from many devices and applies analytics and applications to share the most valuable data. What IoT achieves is taking the classic Internet, sensor networks and cellular networks to another level, because everything is connected to the Internet. One aspect that needs to be considered is to ensure that there are issues related to confidentiality, data integrity, and authenticity due to security and privacy issues which needs to be addressed. If sensor data is tampered with during the communication process, invalid analysis results and control failure will be triggered, as such an event could result in serious damage. Moreover, since measurement data and management teams are trade secrets of manufacturing and management know-how, leak prevention is also important from a competitive standpoint. Even if there are no obvious problems, it is important to consider the consequences of threats that may arise in the future. Various studies have shown that IoT networks already face various security challenges [3–5], including identity verification, authorization, information leakage, privacy, verification, tampering, interference, eavesdropping, etc.

Encryption is key component in the process of extinguishing the threat of various cyber attacks as it is

already used as a data link layer standard in communication systems and effectively reclaimed in application layer to ensure end-to-end data security. Regardless of the communication system, protocol or application used, encryption effectively protects data travelling from two points ensuring data integrity and confidentiality while trying to keep the communication overhead a compact as possible so the overall performance is not impended [6].

1.2 Context

From a high-level perspective, IoT consists of three components, namely hardware, middleware and presentation [7]. The hardware is composed of sensors and actuators, the Middleware provides storage and calculation tools and the presentation provides accessible interpretation tools on different platforms. It is not possible to process data collected from billion sensors thus middleware solutions have been proposed to help a sensor decide the most important data for processing [8]. International IoT architecture does not offer a sufficient margin to obtain the necessary actions involved in the process of authentication, authorization and data integrity. Integrity of the data also becomes vital and requires particular attention to maintain its reliability.

Even though a number of IoT-focused security solutions exist [9], each system may differ regarding many aspects. Most of the security cases in IoT are focused on a generalized paradigm that hypothetically an IoT platform would be based on, hence is is not wise for a system administrator to build their system focused on security mechanisms that have been analyzed and projected for many years to the public eye, which implies that attackers have also studied the flaws of the particular solution [10]. That is why many private IoT networks are urged to study and develop the course of the security methods that are going to be implemented to the system based on the needs of their application and not head for the easy to implement security elements.

Security in IoT differs from the security that is normally found in normal IT networks. Along the IoT industry, the IoT sensors have developed as well, when compared to the devices we were used to seeing some years back, meaning they are capable of processing data far more efficiently. However, we should not forget that these devices are usually micro sensors of low computational and electrical power which usually rely on batteries and have a limited life expectancy, so the processing power is not the same as a desk or server computer. Therefore the security overhead of the data transferred in these devices should be drawn in so the performance of these low powered micro-sensors would not hinder during transmission. That is why end-to-end communication (Figure 1.1 [11]) in IoT should use a lightweight encryption algorithm to achieve end-to-end security and lower power consumption in low-resource devices, therefore smart devices with few resources are adaptable and the footprint of lightweight cryptography is much smaller than that of classic cryptography. The demand for lightweight encryption has been widely addressed [6, 9, 12–14], and several simple cryptography algorithms have been developed which provide the possibility to establish a stronger network connection on smart devices with fewer resources [15], but do not always take advantage of trade-offs between security and efficiency.

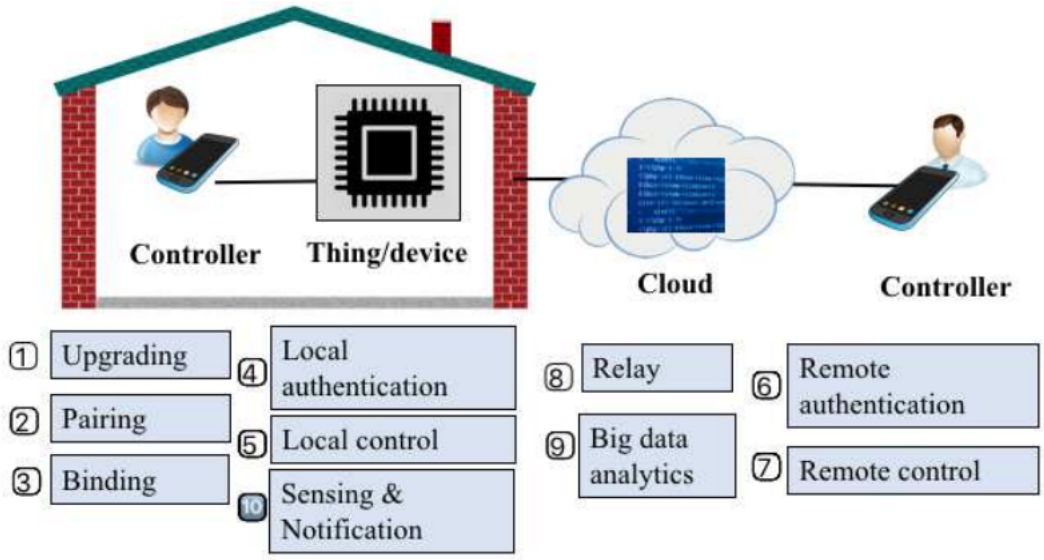


Figure 1.1: End-to-end perspective of IoT [11]

1.3 Significance, Scope and Definitions

As I mentioned, an IoT platform is composed from three components. Regarding hardware, a set of 5G network equipment is needed which will be able to serve the promised performance of 5G, including devices that will connect to this network in order to transmit data. The middleware should be able to handle hundreds of requests during the communication of devices and should be able to authorize all of the devices that are sending all these request, while also storing the valuable data in the system with integrity and authenticity. Some basic definition analysis regarding the characteristics of security are shown in Table 2.1. Lastly the presentation of data to the public should be showcased through a client-server schema that follows the request and receive concept, while also taking care of authorization and authentication of the user that requested the data. All of this communication among the three components and even inside the system should be kept safe and secure. This is where encryption is introduced in order to prevent any data leak or interception and take measure of any unexpected attacks [3–6, 9].

The aim of this B.Sc. thesis is to explore the different encryption mechanisms that exist, analyze their main functionalities and technologies used in the development of those mechanisms. Of course all the encryption methods that will be mentioned have been used and some of them are still being used by many corporations and well known applications. However, each method should be questioned and compared to the rest of the method-candidates in order to decide which is the best approach for the system and how well that specific mechanism is going to contribute to the overall infrastructure when it comes to security. In order to test the functionality of the mechanism, a set of experiments have to be conducted and test how well the proposed copes with the rest of the system.

Having said that, there still needs to be a 5G platform that can apply the encryption and other security elements. For this reason, over the course of my internship in University of Bristol [16], our team in Smart Internet Labs [17] developed from scratch a 5G hybrid infrastructure and chose the best practices and technologies the market has to offer in order to maximize the performance when combined with the

5G network of the laboratory. The proposed framework consists of a main body where data is processed and saved, alongside the security components which are in charge of applying security elements to the system all while being protected in a private visualized network. The rest of the framework refers to the users that are going to request the data and the network of devices which collect and send metrics back to the main system.

1.4 Thesis Outline - Contribution

The thesis begins with an introduction to security, encryption and the encryption diversity, followed by the framework rundown and the experiments that were conducted. More precisely, in Section 2.1 the basics of general security is going to be analyze and what are some of the most well known security risks that threaten modern systems. In the following section, I am exploring the fundamentals of encryption, the diversity of encryption mechanisms, protocols and methodologies, as well as their use in various technological fields. The last section of Chapter 2.1 unravel and reviews other security surveys regarding 5G and MEC infrastructures. The main infrastructure of the system we developed is presented in Chapter 3, where every component of the framework is examined in each section and explaining their usability and importance to the system as well as their security parameters. Deeper into the chapter, the encryption method is thoroughly described, how it is created, what is its purpose and how it is implemented in the system. Lastly, on Chapter 4 the experimentation results are unfolded along the testbed configuration schema that was used for these experiments. The main contributions of this thesis are summarized and listed as follows:

- **Development of encryption mechanism**: For the purpose of this project a custom encryption mechanism was created based on a combination of modern security protocols and widely used ciphers.
- **Configuration of private 5G network and load balancing techniques**: The system was configured through a containerized private network and server/load balancer application in order to organize each component properly.
- **Creating Middleware API**: The Middleware API is introduced to the framework in order to serve the multiple IoT devices requests.
- **Implementation of the encryption mechanism to the system**: After developing the ideal encryption method, it is then configured throughout the communicating ends and inside the whole network.
- **Introduction to authorization and authentication**: The security in the infrastructure is also enhanced by authorization and authentication techniques.
- **IoT device emulator**: A device emulator was created to mimic the basic functions of a IoT device without having the expenses and compatibility problems of real one. Chapter 3.1.10 includes an in depth guide of the emulator.
- **Metrics collection and visualization**: The data of each device are collected through an API which are later used to visualize the metrics to the user.
- **Database organizing**: The database that was chosen uses a NoSQL data structure which is very flexible and adaptable hence queries are performed rather faster due to a well organized database.
- **Testbed and experimentation**: The testbed that was used for the experimentation process is described in Chapter 4.1 which is used to retrieve all sort of information regarding the framework and encryption mechanism.
- **System optimization**: During the experimentation phase, tweaking was also involved to both the system and encryption configuration so the performance would be improved.

Chapter 2: Literature Review

2.1 Introduction to Security

Security is a word that many people are familiar with, which indicates that something is safe and therefore can be accepted as it is believed to be trusted. For instance, people trust that the buildings they go in will not collapse because they are ”secured” by new and enforced foundations. However, if a building is old and not taken care of through the years, people will be sceptical about trusting that building as its ”security” seems unstable for someone to use. The same scepticism also applies to computer security. People will trust sites,organisations and applications based on the security that they offer, because what people need is privacy, confidentiality and certainty in order to feel safe when going online [18, 19].

Whitman and Mattord [20], define information or cyber security as ”the protection of information and its critical elements, including the systems and hardware that use, store, and transmit that information” ([20], p. 8). These writers also point out a number of important aspects of information that contribute to the usefulness of security in organizations. These characteristics include *Confidentiality*, *Integrity* and *Availability* of information as stated in the definition of ISO/IEC 27002 [21], which are also known as the *CIA Triad* or *triangle*, but Whitman and Mattord [20] state that security should not be limited to these three characteristics alone. Accuracy, authenticity, utility and possession should also be taken into consideration when referring to information security characteristics, which I analyze and give a definition from my perspective based on what I have drawn from the literature review in Table 2.1.

There are plenty of other definitions that provide a substantial understanding of cybersecurity hence I have listed some of them which is a result of my literature review of the meaning behind information,computer and cyber security:

- “The art of ensuring the existence and continuity of the information society of a nation, guaranteeing and protecting, in Cyberspace, its information, assets and critical infrastructure.” (Canongia & Mandarino [22])
- “The activity or process, ability or capability, or state whereby information and communications systems and the information contained therein are protected from and/or defended against damage, unauthorized use or modification, or exploitation.” (NICCS [23])
- “The state of being protected against the criminal or unauthorized use of electronic data, or the measures taken to achieve this.” (Oxford University Press [24])
- “Cybersecurity entails the safeguarding of computer networks and the information they contain from penetration and from malicious damage or disruption.” (Lewis James A. [25])
- “Cybersecurity is the collection of tools, policies, security concepts, security safeguards, guidelines, risk management approaches, actions, training, best practices, assurance and technologies that can be used to protect the cyber environment and organization and user’s assets.” (International Telecommunication Union (ITU) [26])
- “The body of technologies, processes, practices and response and mitigation measures designed to protect networks, computers, programs and data from attack, damage or unauthorized access so as to ensure confidentiality, integrity and availability.” (Public Safety Canada [27])

Table 2.1: Definitions of information security characteristics

Characteristics	Definition
Confidentiality	Maintaining authorized limitations on data access and disclosure in place, including safeguards for personal privacy and proprietary data.
Integrity	Ensures that data is real, accurate and unaltered from attempts of data tampering.
Availability	Availability refers to the likelihood that a system will deliver the correct data as expected when needed during the course of a task .
Accuracy	The degree of correctness in data which are a result of process, computation or transmission. How correct are the data someone received when compared to the expected outcome.
Authenticity	Whether or not an information is original. The expected quality of the information that is received being trustworthy and genuine.
Utility	How relevant is a piece of information to the context it relates to by questioning its usefulness and functionality.
Possession	Part of the information belongs to the user and is only acquired by them.
Privacy	Maintaining anonymity while connecting to a public service by hiding to the information that relates to personal or sensitive data from others.

According to Wood [28], information security used to be a purely technical matter. However, as the use of computers and networks evolved, the process of securing those computers and networks also had to evolve and go beyond the purely technical part. While the process of information security requires the use of certain products, it is not something that can be bought off the shelf. Information security is neither a product nor a technology, but a rather process , which can be applied on a network, database or any other application that should be kept safe [29]. All the elements mentioned above are also required when securing a private network. However, you should keep in mind that private LANs differ from the open Internet in at least two notable ways. Firstly, most devices communicate in a fixed architecture. Although some flexibility in establishing a connection is necessary, topologies are usually flat, with one access point and individual hops for all connected devices. Secondly, the topology is stable. Unlike the Internet, and especially unlike public access points connected to mobile hosts, private LAN IoT devices support a fixed or slowly changing mix of connected hosts. This provides some opportunities to establish strong connection and hide endpoints that are not available with public access points [30]. Therefore the structure of a private network itself offers a basic layer of privacy and security.

If system-level security (e.g., confidentiality, integrity, authenticity, and privacy of user data as seen in Table 2.1) is not ensured by the system, IoT applications will not be widely adopted by relevant collarbo-

rators [31]. One of the biggest security challenges arises from a particular characteristic of IoT which is devices heterogeneity. The IoT aims to connect a large number of heterogeneous devices (e.g., sensors and smart mobile phones) to enable advanced applications in different domains. This high level of heterogeneity offers great potential to impact network and protocol security [31, 32]. The inherent complexity of the IoT, with thousands of entities scattered across different contexts and applications, further complicates the development of scalable security mechanisms [32]. The IoT must be built in a way that ensures easy and secure user control. For users to fully embrace the application and reap its potential benefits, they must be confident that it does not pose major risks to their security and privacy [18, 19, 33]. In addition, Europol reported [34] in 2014 that as more and more objects become connected to the Internet and new types of critical infrastructures emerge, we can expect (more) targeted attacks on existing and emerging infrastructures, including new forms of extortion and data theft, blackmail schemes, bodily harm and possible death, and new types of botnets. In other words, the more ways there are to access the network, the more vulnerabilities there are to exploit [34, 35]. This leads us to the next topic which is the security challenges an IoT network can face from well known cyber threats all the way to attacks which give access to the physical network.

2.1.1 Security Challenges

Security and privacy are critical to the safe and reliable operation of IoT services. The number of things connected to the network for IoT services is rapidly increasing, posing a significant security risk to users and service providers. The IoT poses a variety of potential security risks that can be exploited to harm both system operations and the user's device by many aspects such enabling unauthorized access and misuse of personal information [36], creating risks to personal security [37] and facilitating attacks on other systems [38]. Perceived risks to user privacy and security, even if not realized, would seriously undermine confidence in the ability of these technologies to reach their full potential and could limit widespread adoption of the technology itself [39]. It is important to have a better look at these threats and challenges and observe the level of the damage they could cause to a system.

Security threats can source from various roots and each threaten a different part of the system. Threats to confidentiality are those that result in the unintentional disclosure of sensitive information. For instance, breaches in the security of private networks may result in the accidental exposure of critical sensor data. Even apparently benign data, such as the interior temperature of a building, combined with knowledge of the air conditioning system's operating settings, may be used to identify whether a property is inhabited or not, a prelude to burglary [40]. Loss of secrecy in items such as keys and passwords exposes systems to risks of unauthorized access. Authentication risks may result in the tampering of sensing or control information. Unauthenticated system status alerts, for example, may fool a home controller into believing there is an emergency and opening doors and windows to allow for an emergency escape, while in reality they are allowing for illegal access [41]. One point that will be addressed later is the issue of automatic software updates—if they are not properly authorized, complications may occur. Threats to access are arguably the most serious. Unauthorized access to a system controller, especially at the administrator level, compromises the security of the whole system. This may occur as a result of ineffective password and key management, or as a result of illegal devices connecting to the network. Even if control is not obtained, an illegal connection to a network may be used to steal network bandwidth or cause a DoS

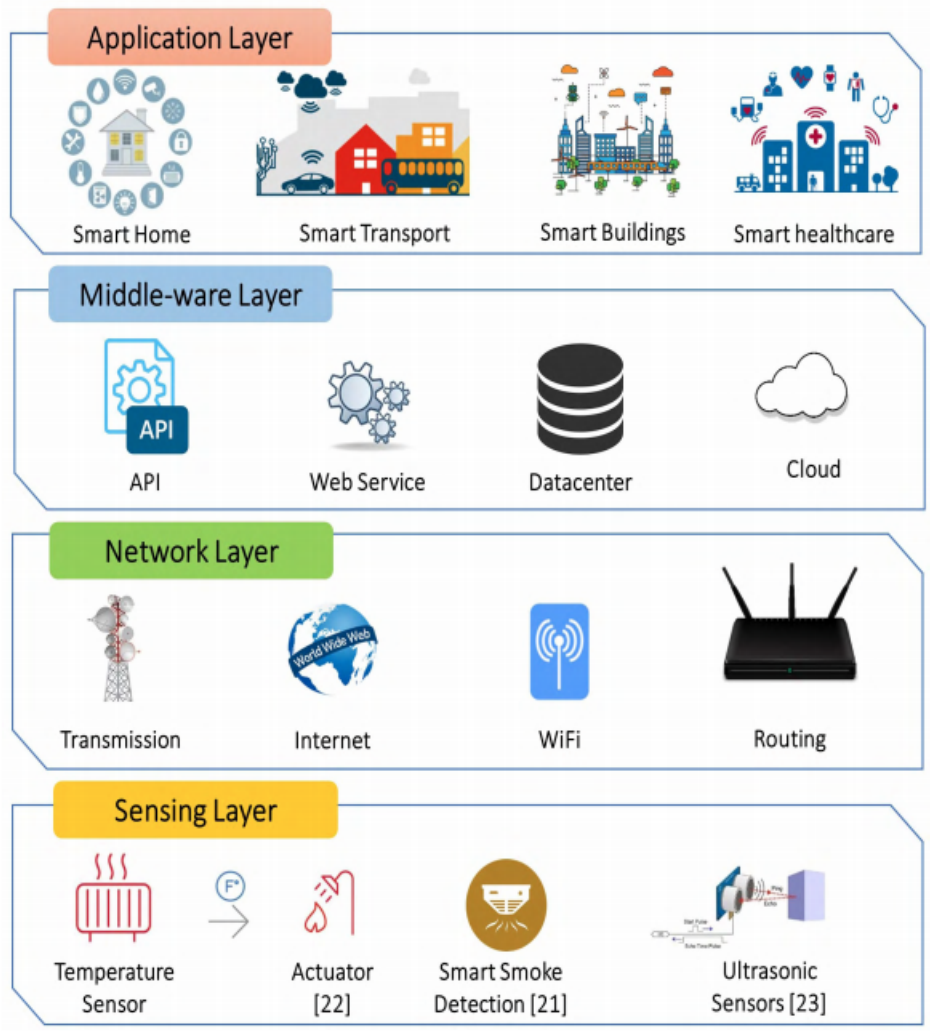


Figure 2.1: Iot platform layers [42]

attack against legitimate users. Due to the fact that many Smart Home gadgets are battery-powered and wirelessly networked with a short operating duty cycle, flooding a network with requests may result in an energy depletion attack—a kind of denial of service attack. In the next sections I am going to analyze attacks which may occur on the different part of the IoT infrastructure, such as the network access, information management system and attacks on the two ends of user and devices as well as lower levels (physical) attacks. Figure 2.1 [42] represents the layers of an IoT based infrastructure.

This kind of threats may occur in every layer (Figure 2.1 [42]) of the infrastructure and may pose serious problems to the the CIA Triad of security, rendering our system not safe to use. In the next paragraphs I am listing the possible attacks that can occur in these layers and the way they affect the system.

2.1.2 Sensor level threats

Sensor level threats refer to attacks that occur on the IoT devices which communicate with our infrastructure and are vulnerable to many security related vulnerabilities [43, 44]. The main threats of sensor level are :

Eavesdropping and Interference: IoT systems often use a distributed architecture with many nodes deployed in open settings [45]. As a consequence, these IoT apps are vulnerable to eavesdroppers. Attackers may eavesdrop on and seize data at several stages, such as data transfer or authentication.

False Data Injection Attack: Once the attacker captures the node, he or she may utilize it to inject erroneous data into the IoT system (Figure 2.2). This may provide erroneous findings and may result in the IoT application malfunctioning [46]. Additionally, the attacker may use this technique to launch a DDoS attack.

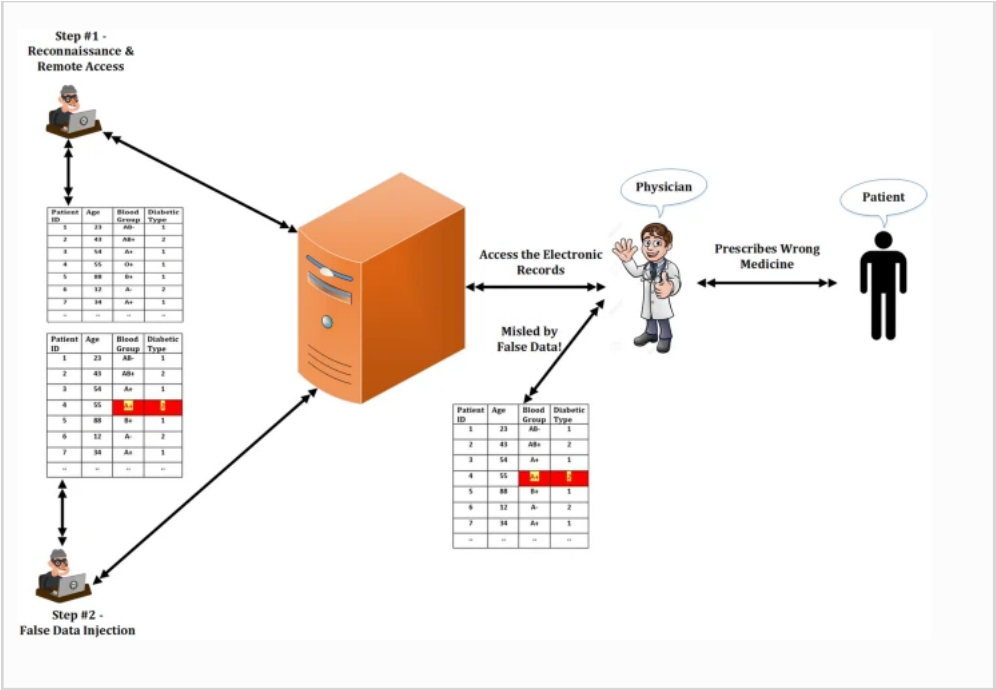


Figure 2.2: False data injection scheme [46]

Booting Attacks: During the boot phase, edge devices are susceptible to a variety of attacks. This is due to the fact that the built-in security procedures are disabled at that time. Attackers may exploit this vulnerability and attempt to attack node devices when they are rebooted. Due to the low power consumption and periodic sleep-wake cycles of edge devices, it is critical to protect the boot process in these devices.

Malicious Code Injection Attack: The attacker injects malicious code into the node's memory. Generally, IoT node's firmware or software are updated over the air, providing an opportunity for attackers to insert malicious malware. By injecting malicious code into the nodes, attackers may compel them to perform unexpected tasks or even attempt to gain access to the whole IoT system.

Capturing Nodes: IoT applications are composed of a number of low-power nodes, such as sensors and actuators. These nodes are susceptible to a number of adversarial attacks. The attackers may attempt to seize or replace the IoT node with a malicious node. Although the new node seems to be part of the system, it is really controlled by the attacker. This may result in a breach of the security of the whole IoT application [47].

Side-Channel Attack (SCA): In addition to direct attacks on nodes, a variety of side-channel attacks may

result in the disclosure of sensitive data. Processor microarchitectures, electromagnetic emission, and power usage can provide enemies with valuable information. Side channel attacks may be motivated by power consumption, lasers, timing, or electromagnetics. While implementing the cryptography modules, modern chips take different precautions to avoid these side-channel attacks.

Sleep Deprivation Attacks: In this kind of attack, attackers attempt to deplete the battery of low-power IoT edge devices. This results in a denial of service attack on the IoT application's nodes owing to a dead battery. This may be accomplished by executing malicious code in endless loops on the edge devices or by artificially raising the power consumption of the edge devices.

2.1.3 Middleware threats

The middleware layer in IoT provides a connection between the network and application layers, which offers APIs, brokers, database and queuing systems while offering robust processing and storage capabilities [48]. However, the middleware is still vulnerable to a variety of cyber attacks which are addressed here :

Cloud Malware Injection: The attacker may gain control, inject malicious code, or implant a virtual computer into the cloud through cloud malware injection.By attempting to construct a virtual machine instance or a malicious service module, the attacker impersonates a legitimate service. Thus, the attacker has access to the victim's service's service requests and captures sensitive data that may be changed according to the instance.

SQL Injection Attack: SQL Injection (SQLi) attacks pose a threat also to the middleware. The attacker may inject harmful SQL statements into a software in the attempt to tamper with data. The attackers may then access sensitive information about any user and even change database entries [49]. SQLi has been identified as a top threat to online security by the Open Web Application Security Project (OWASP) in their OWASP Top 10 2017 publication [50].

Flooding attack in the Cloud: This attack is similar to a DoS attack in the cloud and degrades service quality (QoS). To exhaust cloud resources, attackers make numerous requests to a service in a continuous loop. By raising the demand on cloud servers, these attacks may have a significant effect on cloud systems.

Signature Wrapping Attack: XML signatures are utilized in the middleware's web services. By exploiting SOAP (Simple Object Access Protocol) flaws, the attacker may circumvent the signature algorithm and perform operations or alter the eavesdropped message in a signature wrapping attack [51].

Man-in-the-Middle Attack (MiM,MitM): One of the most well known attacks in the history of security threats. MiM refers to a malicious third party's cryptographic attack on a communication channel, in which the intruder takes over a confidential/personal communication channel between two or more communicative points or parties (Figure 2.3 [52]). The attacker may manipulate (read, alter, intercept, change, or replace) the communication traffic between victims in this cyber-attack. The thing with the MITM protocol is that the unauthenticated attackers leave no evidence/traces of their interception of this criminality; in other words, the attacker is completely invisible to the victims. To achieve its goal, MiT

attacks utilize numerous penetration tools such as *sniffing, hijacking, injecting* and *filtering* [53].

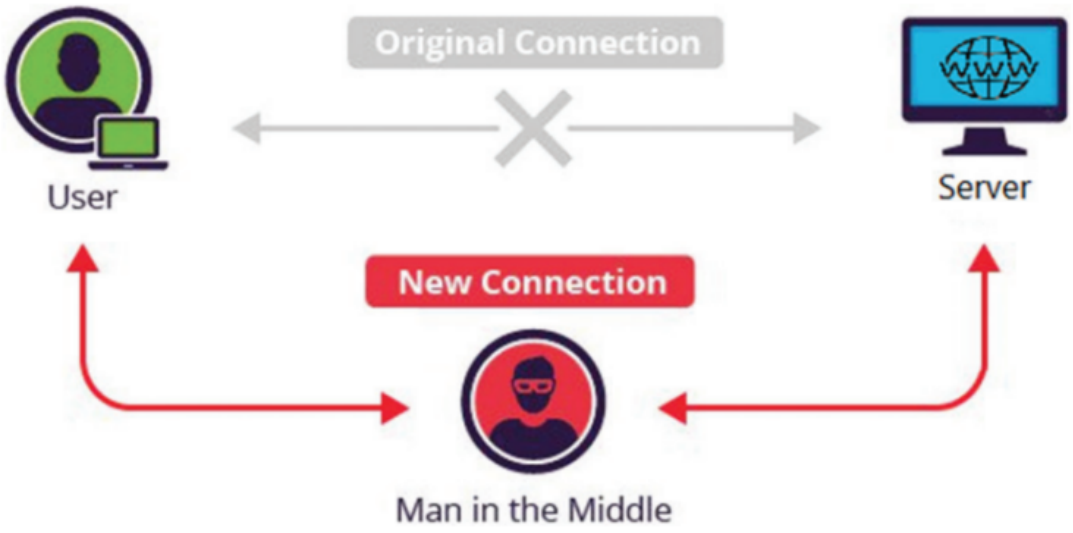


Figure 2.3: Man-in-the-Middle attack [52]

2.1.4 Application layer attacks

Service Interruption attacks: In the current literature, these attacks are sometimes referred to as unlawful interruption attacks or DDoS attacks. Numerous examples of similar attacks against IoT apps have been documented. These attacks prevent genuine users from accessing IoT apps' services by artificially overloading the servers or network.

Sniffing Attacks: Sniffer apps may be used by attackers to monitor network traffic in IoT applications. This may enable the attacker to access sensitive user data if not enough security measures are in place to prevent it [53].

Theft of Data: IoT apps handle a large amount of sensitive and private data. Data in transit is much more susceptible to attacks than data at rest, and data mobility is common in IoT applications. Users will be hesitant to register their personal information on IoT apps that are susceptible to data theft attempts. Encryption of data, isolation of data, user and network authentication, and privacy management are just a few of the methods and protocols used to protect IoT applications against data theft.

Malicious Code Injection Attacks: Attackers often use the simplest or easiest way possible to gain access to a system or network. If the system is susceptible to malicious scripts and misdirection as a result of inadequate code checks, it is the first point of entry for an attacker. Generally, attackers use XSS (cross-site scripting) to insert malicious code into a seemingly trustworthy website. A successful XSS attack may result in the account of an IoT device being hijacked and the IoT system being rendered inoperable.

Access Control Attack: Access control is an authorization method that controls access to data or accounts to only authorized people or processes. Access control attacks are important in IoT applications because

once the access components are compromised, the system becomes very susceptible to attacks.

Reprogram Attacks: If the programming process is not secured, attackers may attempt to remotely reprogram IoT devices. This may result in the IoT network being hijacked [53].

2.1.5 Network attacks

The network is probably the component of an infrastructure that is prone to major attack attempts. This were data is transferred from the devices and sent over to the main system, therefore intruders interested in those data will launch various malicious attacks in order to achieve their goal.

DDoS/DoS Attack: This kind of attack floods the target servers with a high number of unsolicited requests, as shown in Figure 2.4. This disables the target server, thus interfering with legitimate user services. If the attacker uses several sources to overwhelm the target server, this is referred to be a DDoS or distributed denial of service attack. While these attacks are not exclusive to IoT applications, the heterogeneity and complexity of IoT networks make the network layer vulnerable to them. Numerous IoT devices used in IoT applications are not securely configured, making them ideal targets for attackers launching DDoS attacks on target servers [54].

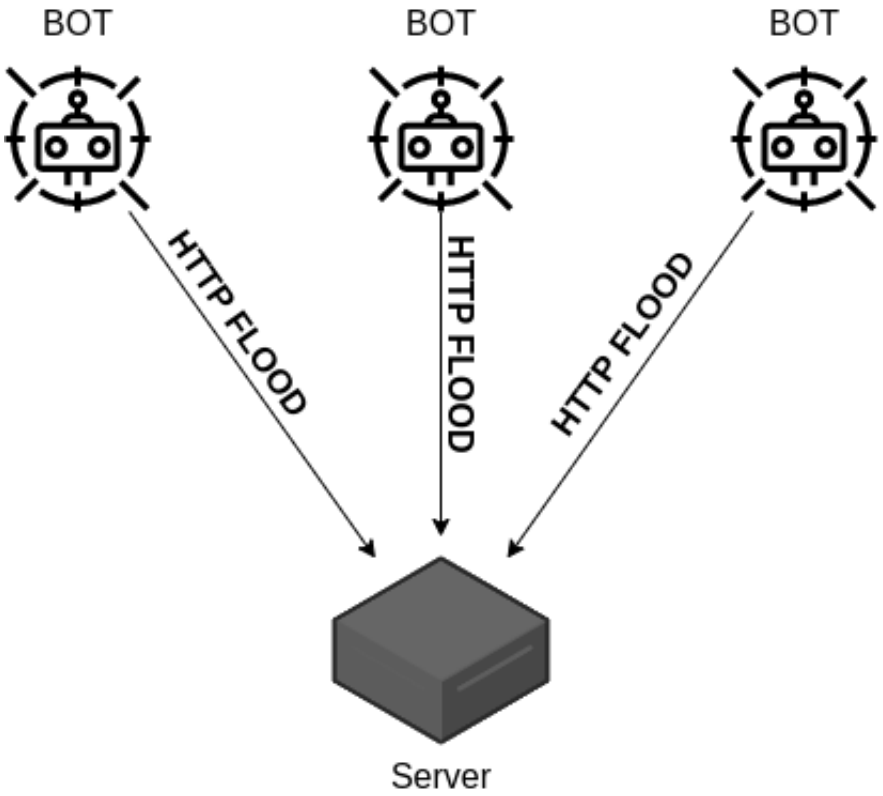


Figure 2.4: DDos attack flooding the server

Routing Attacks: During data transmission, hostile nodes in an IoT application may attempt to alter routing routes. Sinkhole attacks are a kind of routing attack in which an adversary advertises the shortest possible routing path and recruits nodes to use it. A worm-hole attack is another kind of attack that, when coupled with other types of attacks such as sinkhole attacks, may pose a significant security concern. A

warm-hole is an out-of-band link between two nodes with the purpose of transferring packets quickly. Between a hacked node and a device, an attacker may establish a warm-hole.

Phishing Site Attack: Phishing attacks are often used to refer to attacks in which many IoT devices may be targeted with little effort on the part of the attacker. The attackers anticipate that at least a number of the devices will be compromised. Users may come across phishing sites when browsing web pages on the Internet. Once a user's account and password have been stolen, the user's whole IoT ecosystem becomes susceptible to cyber attacks. The network layer of the IoT is very susceptible to phishing site attacks [55].

Access Attack: Also known as advanced persistent threat (APT). This is a kind of attack in which an unauthorized individual or adversary gets access to the IoT network. The attacker may remain undiscovered in the network for an extended period of time. The objective or goal of this kind of attack is to steal important data or information, rather than to create network harm. IoT apps receive and transmit important data on a continual basis, making them very susceptible to such attacks [36].

Data Transit Attacks: IoT apps store and exchange a great quantity of data. Because data is valuable, it is always the target of hackers and other intruders. While data kept on local servers or in the cloud poses a security risk, data in transit or traveling between locations is much more susceptible to cyber attacks. There is a great deal of data flow between sensors, actuators, and the cloud in IoT applications. Due to the fact that such data transfers include a variety of connection protocols, IoT applications are vulnerable to data breaches.

2.2 Encryption

Encryption is an important module of the security in an IoT platform for securing data. Encryption is a technique used by systems for encrypting data so that only authorized parties may decrypt it [56]. In technical words, it is the process of transforming plaintext that is readable by humans to unintelligible text, sometimes referred to as ciphertext. To put it another way, encryption takes readable data and modifies it to seem random as shown in Figure 2.5. Encryption involves the employment of a cryptographic key [57], which is a collection of mathematical values agreed upon by both the sender and receiver of an encrypted communication. Although encrypted data looks random, encryption follows a logical, predictable pattern, enabling a party who gets the encrypted data and has the appropriate key to decode it and restore it to the original form. True safe encryption will use keys that are sufficiently complicated that a third party will be very improbable to decode or break the ciphertext using brute force — in other words, by guessing the key. Encryption may occur when data is being kept or stored, at rest, or while it is being sent to another location, in transit [58].

End-to-end encryption refers to the protection of data throughout the whole route of transportation from one end, supposedly a user, to the other end that is the IoT device and vise versa. When developers or manufacturers use end-to-end encryption, no one other than the sender and intended receiver has access to the data as it travels from device to device, even if it travels over the internet. This level of security is critical for IoT platforms that handle extremely sensitive data and want to keep it secure. Encryption significantly limits the attack vectors available to a thief attempting to steal data transmitted across many

IoT devices, reducing the possibilities of security attacks that we mentioned in Chapter 2.1.1, penetrating and abusing our system. Most IoT devices encrypt data as it travels over the web and to other devices. Few manufacturers, on the other hand, use on-device or centralized encryption, which means that data is only secure for a portion of the time it is in transit.

While cryptography is critical for IoT device security, implementing encryption standards presents some major difficulties. Like I mentioned, due to the fact that these devices have different, less capable hardware requirements than other devices, such as computers or cellphones, conventional encryption techniques may be ineffective. Unfortunately, not all manufacturers believe that well-known encryption methods, such as AES, are appropriate for IoT devices. However, IoT devices are special. Often, they feature customized hardware that offers just enough computing power to accomplish the job at hand. Certain manufacturers have resisted the notion of using tried-and-true encryption technologies such as AES [59]. This is because the AES is not intended to be lightweight, which means that adopting this kind of encryption may be difficult for devices with little computing capacity, such as IoT sensors. Nevertheless, according to major IoT research, instead of working with lighter and less secure encryption models, AES can be configured in order to produce a lightweight result for IoT infrastructures [60].

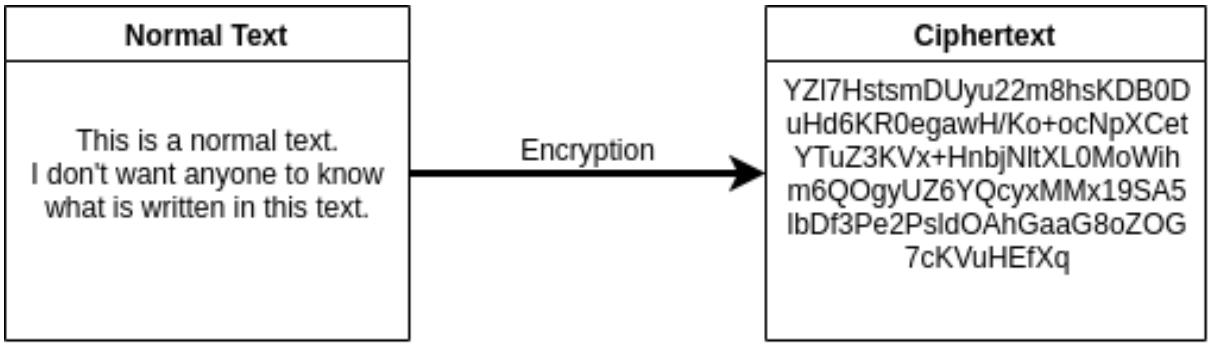


Figure 2.5: Encryption process

2.3 Encryption Diversity

There are plenty of encryption algorithms in the market, one of which is AES (Advanced Encryption Standard), which I mentioned in the previous section. Each of these encryption methods are based one a unique cipher algorithm which performs the cryptography on data that the encryption mechanisms is processing. However, there are only two categories in which the encryption mechanisms are classified, those being the asymmetric and symmetric encryption [61]. On the next sections the encryption methods are analyzed used in symmetric and asymmetric encryption schemes.

2.3.1 Symmetric

Symmetric key encryption is the most common form of encryption. In symmetric key encryption, the same key is used to encrypt and decrypt the transferred data (Figure 2.6 [62]). As a result, the key is preserved and kept private. Symmetric key encryption is performed in two modes: block ciphers and stream ciphers. The block cipher mode divides the data into blocks and generates an encryption key depending on the block length. In the case of stream ciphers, data is split into as few as single bits and

randomized before encryption occurs. Strictly speaking, symmetric key encryption methods are much quicker than asymmetric key cryptosystems while it also has the benefit of using less computer resources and operating at a fast rate of speed during encryption [61]. A block cipher is used as the input together with the key and the output block is the same size as the input block in symmetric key encryption. The following symmetric key encryption methods are evaluated in terms of performance : the **AES** algorithm, the **Blowfish** algorithm, the **Twofish** algorithm, the **DES** algorithm,the **Triple DES** algorithm and the **Rivest Cipher 4 (RC4)** algorithm are some examples of symmetric encryption algorithms.

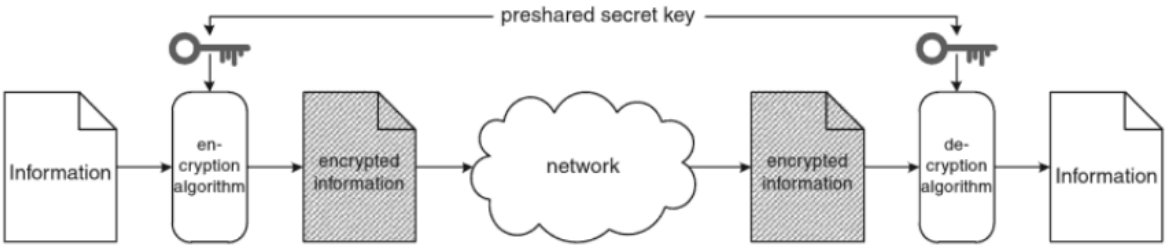


Figure 2.6: Symmetric encryption schema [62]

AES which is also known as Rijndael, named after its its creators Vincent Rijmen and Joan Daemen, was first introduced in 1998 and was created as a replacement for the DES method hence it was approved as an encryption standard by NIST in 2001 [59, 63]. In contrast to DES, AES is a family of block ciphers composed of ciphers with varying key lengths and block sizes and is based on substitution and permutation methods. The raw data is first converted to blocks, and then encrypted using the encryption key. The encryption process is divided into many sub-processes, including sub bytes, row shifting, column mixing, and round key addition, which depending on the key size can be repeated as many as 14 rounds [63]. However, the last round does not include the column mix process which is applied to the other rounds. It is important to notice that an encryption method is a bi-directional procedure that encrypts and decrypts data and therefore should maintain a high level of security for the data they send. Internally, the AES algorithm operates on a two-dimensional array of bytes referred to as the state. The state is composed of four rows of bytes, each of which includes Nb bytes, where Nb is the size of the block length after it's been divided [59].

Blowfish is a significant kind of symmetric key encryption that uses a 64-bit block size and a configurable keys ranging from 32 to 448 bits in length, thus making it suitable for both local and export usage. It is a sixteen-round fiestel cipher that uses big key size. Due to the higher key size, it is more difficult to crack the blowfish algorithm's code and competed in order to replace DES. Bruce Schneier created Blowfish in 1993 as a quick, free alternative to existing encryption methods at the time [64]. It has now been extensively studied, and it is gradually gaining recognition as a robust encryption method. Blowfish is neither copyrighted or subject to a licence, and it is freely accessible for any use. Nowadays a number of products utilize Blowfish and can be seen in the list below [65]:

- **Password Management**
  - ◇ Access Manager
  - ◇ Web Confidential

- Java PasswordSafe

- **Backup Tools**
  - Symantec NetBackup
  - Backup for Workgroups

- **Operating System Examples**
  - Linux
  - OpenBSD

- **Secure Shell (SSH)**
  - OpenSSH
  - PuTTY

- **Email Encryption**
  - A-Lock
  - SecuMail

- **File/Disk Encryption**
  - GnuPG
  - Bcrypt
  - CryptoForge

Twofish is the successor of Blowfish and is also based on symmetric encryption scheme, using just a single 256-bit key. This is one of the quickest encryption algorithms available and is well-suited for use in both hardware and software platforms. Twofish was a finalist in the National Institute of Technology and Science's (NIST) competition to develop a successor for the Data Encryption Standard (DES) encryption algorithm when it was published however it run short to the Rjindael(AES) algorithm encryption which was chosen over Twofish [66]. As with Blowfish, the block cipher technique is also used in Twofish with the addition of distinguishing characteristics that set it appart from its predecessor other major cryptography methods. For starters, it makes use of pre-calculated, key-dependent S-boxes. A substitution-box (S-box) is a fundamental component of any symmetric key method that performs substitution. The S-box obscures the connection between the key and the ciphertext using Twofish's block cipher mechanism. Twofish employs a pre-calculated, key-dependent S-box, which implies that the S-box is already generated but must be decrypted using the cipher key [67].

DES was released in 1977 the National Institute of Standards and Technology (NIST) as a symmetric-key block cipher. DES uses a 56-bit key to encrypt and decode data in 64-bit blocks and is based on the Feistel cipher [64]. It accepts a 64-bit block of plaintext as input and returns a 64-bit block of encrypted text as output. Because the method always works on blocks of equal size and uses both permutation and substitution operations during the process. DES contains sixteen rounds, which means that the primary

process is performed sixteen times to generate the encrypted text as shown in Figure 2.7. The number of rounds is shown to be exponentially related to the time needed to locate a key using a brute-force attack [68]. Thus, as the number of rounds rises, the algorithm's security grows exponentially. DES was obviously no longer impregnable to attacks and has been shown to be susceptible to very strong security attacks, which led to popularity downfall.

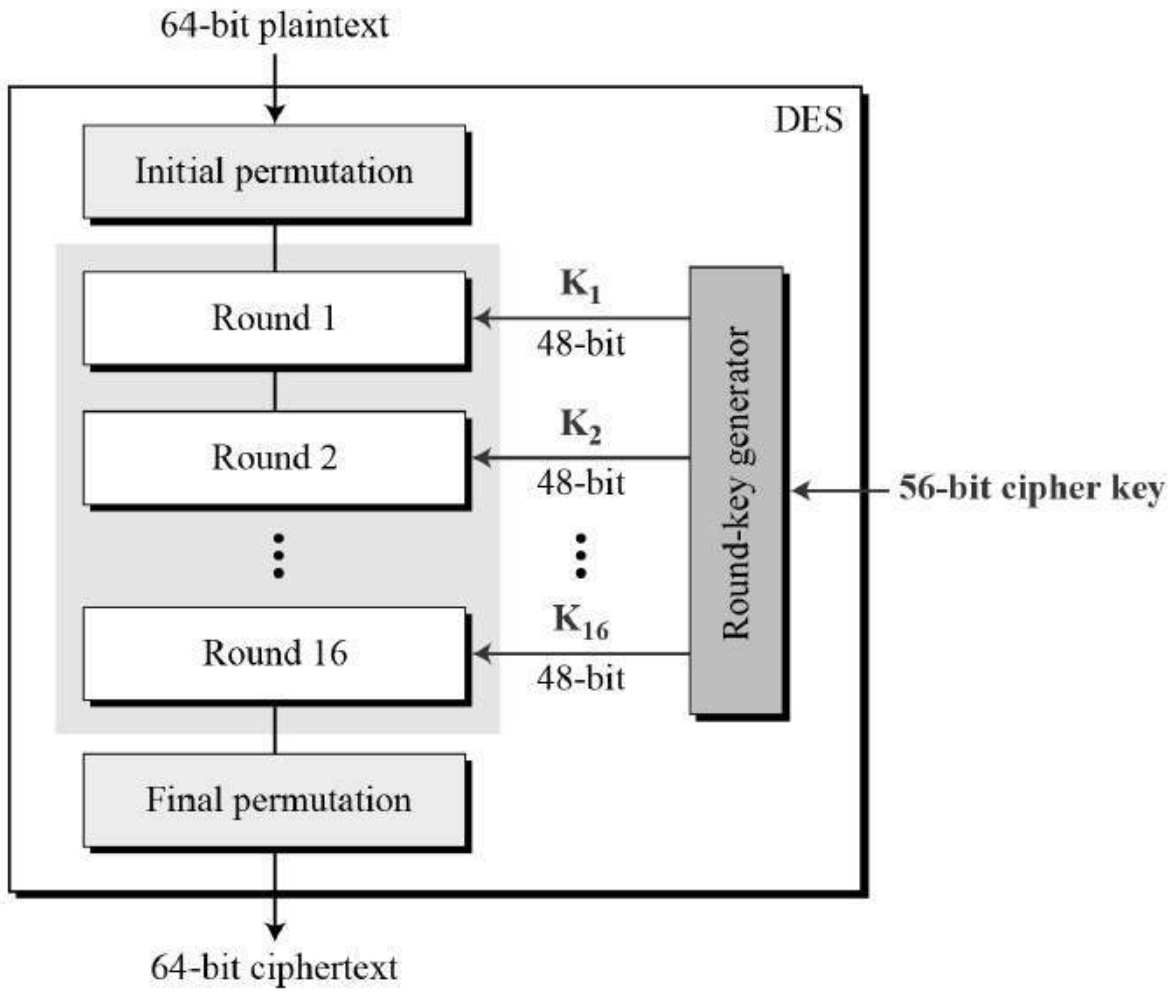


Figure 2.7: General structure of DES [69]

Triple DES is the successor of DES and as the name suggests it applies the same mechanism of 56 bit key and 64-bit blocks just like as DES but applies it three times to the data and utilises a unique key for each phase, including key padding when required, producing a total key size of 112 to 168 bits. This should result in an anticipated strength of at least 112 bits, which is more than sufficient to withstand brute force assaults [68]. Triple DES is much more secure than DES and was set to continue its legacy, however it is deemed to be rather sluggish in comparison to certain modern block ciphers,thus NIST updated the DES with AES(Rijndael) instead of Triple DES [66].

Rivest Cipher 4(RC4) is a symmetric key technique that uses a stream cipher, which in addition to other ciphers, encrypts data in smaller partitions as it would be encrypting one byte at a time. Both encryption and decryption are performed using the same method, since the data stream is simply XOR-ed (Exclusive OR) with the produced key sequence (Figure 2.8. The key stream is fully self-contained

and independent from the plaintext that it's being used to.It initialises a 256-bit state table with a variable-length key ranging from 1 to 256 bits. The state table is used to create successive pseudo-random bits and subsequently a pseudo-random stream that is XORed with the plaintext to produce the ciphertext. The algorithm is componsed into two phases : the initialization and operation process. The initialization step populates the 256-bit state table S with the using the key K as a seed K.Once the state table is configured, it is updated in a consistent pattern as data is encrypted [70].

The RC4 processes plain text in units as tiny as one bit. They sort a kind of memory called stat in preparation for replacement. RC4 is a stream encryption that is binary additive. It makes use of variable-length keys ranging from 8 to 2048 bits in multiples of 8 bits and as a result, key length became a critical factor in RC4 Encryption. Although its security is unclear, it does not seem simple to breach. Nevertheless, the algorithm is very quick and it has been used in certain applications and security protocols, such as SSL/TLS,the Wi-Fi Security Protocol WEP and the IEEE 802.11 wireless LAN standard [71, 72]. RC4 is basically a pseudo-random number generator, with the generator's output being unique to the data stream. As a result, it is critical that the same RC4 key is never used to encrypt two different data streams.

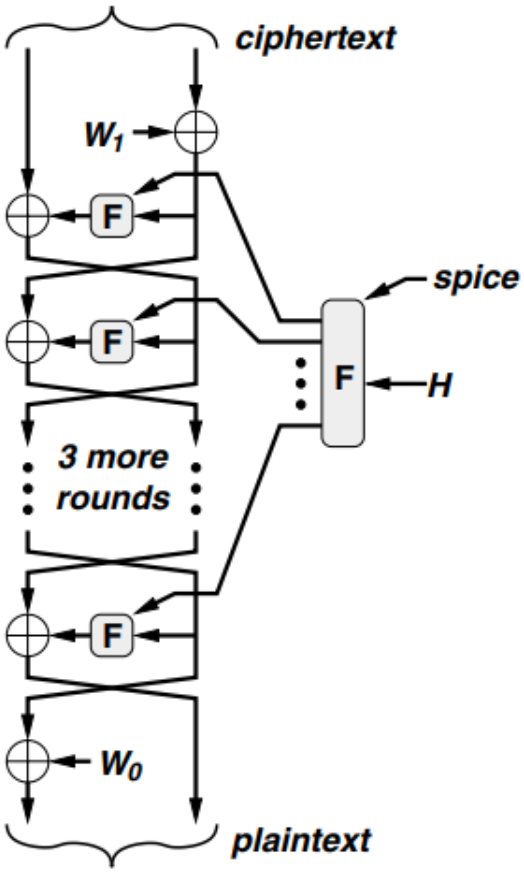


Figure 2.8: RC4 cipher process (decryption) [70]

2.3.2 Asymmetric

In contrast to symmetric encryption, asymmetric encryption utilises multiple keys for the process of encryption and decryption. Asymmetric encryption uses two different encryption keys that are mathematically connected. Additionally, asymmetric cryptography is referred to as public key encryption, as one of these keys is known as the *public key*, while the other is named the *private key*. The first (and most apparent) benefit of this kind of encryption is the added security. The public key, which is publicly available, is used to encrypt the data, while the private key, which must remain hidden and secured, is used to decrypt the data. The whole process of sharing the public key and encrypted/decrypting messages is illustrated in Figure 2.9 . This protects the data against man-in-the-middle (MiM) attacks which I referred to in Chapter2.1.3 [61]. Public key techniques are not only reliable but also convenient because they allow for the safe transmission of encryption keys or other data even when both users haven't shared a secret key in a private communication, which by itself, presents a challenge . Public-key encryption methods often utilise considerably longer keys, which increases the security of the data being sent. To provide the same degree of security as a 128-bit symmetric method, asymmetric encryption algorithms need at least a 3,000-bit key [73]. Lastly, asymmetric cryptography implements the added feature of authentication. As previously stated, data encrypted with a public key can only be decoded with the corresponding private key. As a result, it ensures that the data is only decrypted and read by the intended recipient, therefore it confirming that you are speaking with the person or organisation with whom you believe you are communicating. Some of the most well known asymmetric encryption algorithms are : the **RSA** algorithm, the **Diffie-Hellamn** algorithm, the **ECDSA** algorithm and the **El Gamal** algorithm, which are analyzed in the next few paragraphs.

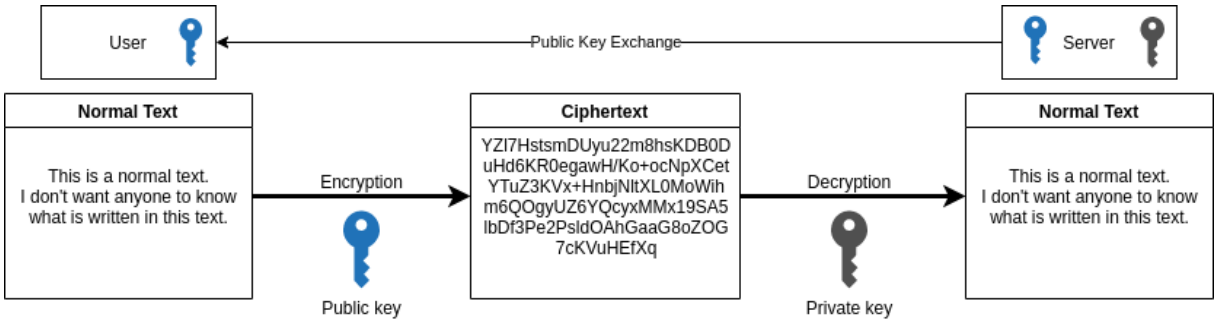


Figure 2.9: Key sharing and data encryption/decreption in asymmetric cryptography

RSA or Rivest-Shamir-Adleman encryption is the most widely used algorithm for assymetric cryptography, as it is resilient and it makes it computationally infeasible, inflexible, and cheap to discover a private key, public key pair in a timely way by attackers [74]. It is suitable for both encryption and digital signatures and is extensively utilised in a broad variety of applications and services. With RSA, the data may be encrypted using either the private or public key, while the other key decrypts it hence the RSA is the most widely used asymmetric encryption method, due to its convenience [73].

The ability to encrypt using either the private or public key offers RSA users with a plethora of benefits. If the public key is used to encrypt the data, it must be decrypted using the private key. This is ideal for transmitting sensitive data over a network or over an Internet connection, where the data receiver gives the data sender their public key. The data sender then encrypts the sensitive information using the recipient's public key and transmits it to them. Because the data is encrypted using the public key, only

the owner of the private key is able to decode it. Thus, even if the data is intercepted in transit, only the intended receiver can decode it.

The second way to use RSA for asymmetric encryption is to encrypt a message using a private key. In this example, the data sender encrypts the data using their private key and transmits the encrypted data along with their public key to the data receiver. The receiver of the data may then decrypt it using the sender's public key, establishing that the sender is really who they claim to be. While data may be stolen and read in transit using this technique, the real goal of this kind of encryption is to establish the sender's identity. If the data is stolen and changed during transit, the public key cannot decode the new message, indicating to the receiver that the data was modified during transit.

The technical specifics of RSA are based on the concept that it is trivial to create a number by multiplying two sufficiently big numbers together, but very difficult to factorise that number back into the original prime numbers. Both numbers are computed using the same two prime integers. RSA keys are typically 1024 or 2048 bits in length and recently been used in 4096 bit forms, which makes factorization very difficult, however, although 2048 bit keys are the most commonly used they are expected to be breakable shortly [75]. The steps for creating an RSA key are described in the study of Dr. Mahajan and Sachdeva [76] which can be seen below :

1. Generate two large distinct primes p and q.
2. Compute $n = p * q$ and $f = (p - 1) * (q - 1)$
3. Select an e, $1 < e < f$, relatively prime to f.
4. Compute the unique integer d, $1 < d < f$ where $e*d \equiv f*1$.
5. Return public key (n, e) and private key d.

Diffie-Hellman (DH) is not exactly a encryption mechanism but rather the first implementations of public-key cryptography (PKC) in practise. Created in 1976 by Whitfield Diffie and Martin Hellman with the assistance of Ralph Merkle and other researchers of UK's intelligence agencies, DH was designed to allow two parties talking over a public channel to create a shared secret without transmitting it over the Internet [77]. This shared channel is then utilized as the secret path for sharing the private keys while keeping it private and secure. It is based on a discrete logarithms problem known as the Diffie-Hellman problem. This problem is considered to be robust and can be enforced using a more complex mathematical algorithm so the security can be enforced [78]. In a schema of two communication ends, supposedly IoT device and the middleware, each of them has some knowledge they want to convey while maintaining its secret. To do this, they agree on a piece of public innocuous information that will be mixed in with their privileged information during its transmission via an unsecured channel. Their secrets are combined with public information, or the public key, and when the secrets are transferred, the information they want to transfer becomes entangled with the shared secret. They may extract public information from the other ends data and by using the information of their own secret, decode the new information that was sent to them. While this technique seems straightforward in its explanation, when lengthy strings are utilised for private and public keys, eavesdropping attempts cannot theoretically decipher the data even with when significantly strong attacks [77].

ECDSA or Elliptic Curve Digital Signature Method or simply Elliptic Curve Cryptograpgy (ECC), is a complicated public key cryptography encryption algorithm, which is used to produce keys that are smaller than the typical keys generated by digital signature methods [79]. ECC is a type of public key cryptography that is motivated by the algebraic structure of elliptic curves over finite fields, where a certain point of the curve is multiplied by another value, resulting in the creation of a new point on the curve (Figure 2.10 [79]). ECC is mostly used to generate pseudo-random numbers, digital signatures, and other cryptographic functions. A digital signature is an authentication technique that utilises a public key pair and a digital certificate to authenticate the identity of the receiver or sender of information, more about this process is included in Chapter 3.2.1.

Altough ECDSA delivers the same result as other encryption methods, it is way more efficient. This is because ECDSA utilises smaller keys in order to provide the same degree of security as other cryptography algorithms. The process starts when the ECDSA certificate is generated, which are a kind of electronic document used to verify the certificate's owner. Certificates include information on the key that was used to generate the certificate, the certificate's owner, and the signature of the certificate's issuer, which is a verified trustworthy entity [80, 81]. This trustworthy issuer is often a certificate authority (CA) that also possesses a signed certificate that can be tracked back to the original issuing certificate authority through the certificate trust chain.

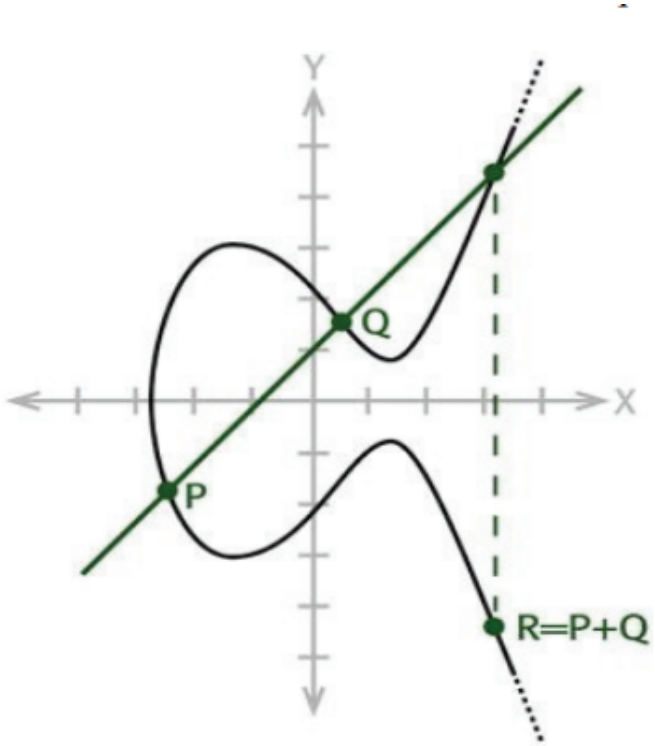


Figure 2.10: Elliptic curve representation of function R=P+Q [79]

The ECC is a relatively new cryptography method, being standardized in 2005, which is based on a complex encryption algorithm which is safer, when compared to other asymmetric encryption methods, to existing cyber attacks [79,82]. Thus, ECC benefits over more frequently used asymmetric key algorithms, such as the RSA, while the use percentage by modern security protocols is increased gradually every

year [82].

El Gamal which similarly to RSA, is an asymmetric encryption method whose security is based upon the difficulty related to computing *discrete logs* in a big prime modulus. First introduced in 1985, it was named by its author and implies that the underlying mathematical issue must be computationally impractical to solve in order to generate the decryption key from the encryption key. El Gamal makes it possible to generate distributed keys quickly and easily, and since encrypted data can be readily exponentiated using collaboratively generated random numbers, it is a good choice for multi party that can be adjusted by the same construction blocks [83]. The following steps drawn by the book of Ryan Ko & Kim K.R Choo [84] represent the algorithmic procedure of El Gamal for the encryption process :

- **El Gamal key generation algorithm**

  1. Select a large prime **p**
  2. Select a primitive value **a** in modulo **p**
  3. Randomly select **d** so that $2 \leq d \leq p-2$
  4. Calculate **b** = $\mathbf{a}^d$ mod **p**
  5. Public Key = **(p,a,b)**
  6. Private Key = **d**

- **El Gamal encryption algorithm**

  1. Select a random integer **k** (which must remain private)
  2. Calculate **r** = $\mathbf{a}^k$ mod **p**
  3. Calculate **t** = $\mathbf{b}^k$ * **M** mod **p**
  4. Discard **k**
  5. Encrypted Message = **(r,t)**

- **El Gamal decryption algorithm**

  1. Calculate **M** = **t** * $\mathbf{r}^{-\mathbf{d}}$ mod **p**

These were some of the most well know encryption methods which are used in both symmetric and asymmetric key cryptography, however, the list is much longer and is beyond the scope of this thesis, hence I am going to summarize in the next table the basic information regarding only those encryption algorithms that I just analyzed.

Table 2.2: Key and block size comparison of encryption mechanisms

	Key Size	**Block Size**
AES	128, 192 or 256 bits	128 bits
Blowfish	32 - 448 bits	64 bits
Twofish	128, 192 or 256 bits	128 bits
DES	56 bits	64 bits
Triple DES	3 x 56 (168) bits	64 bits
RC4 (steam)	40 - 2048 bits	2064 bits
RSA	1024 - 15360 bits	688 - 15024 bits
DH	1024 - 15360 bits	688 - 15024 bits
ECDSA	160 - 512 bits	688 - 15024 bits
ElGamal	768 - 2048 bits	514 bits

2.3.3 Hashing

Hashing is another algorithm which the purpose is also to produce a result that is unreadable and unbreakable by other parties, thus it's been closely related to encryption in the data security area. However these two algorithms differ from each other and are used in divergent occasions. Encryption is a two-way function in which data is sent in as plaintext and comes out as ciphertext, which is unintelligible by the human eye. Given that encryption is a two-way algorithm, the information may be decrypted and made readable once again. Hashing on the other hand, is a one-way algorithm, which means that the plaintext is turned into a unique digest, which is then encrypted and cannot be decrypted [85]. Even though hashing is technically possible to undo, decryption is impractical due to the enormous amount of computing power required to do it. Therefore, a hashing algorithm is used to carry out the operation of hashing that generates a fixed-size bit string value from a file, which in its simplest form is made up by data blocks and what hashing achieves is reducing the length of this data to a much shorter fixed-length value or key that reflects the length of the original data. In some ways, the hash value may be thought of as a condensed version of everything included inside that file [86].

One of the most common applications of hashing is to check for equality between two files. If two

document files are not identical, the computed hash values of these files will let the owner to determine whether or not they are identical without opening them and comparing them word for word. Additionally, hashing is used to validate the integrity of a file after it has been moved from one location to another, which is usually done by a file backup software. A user may check the hash value of both files to verify that the transferred file has not been corrupted before transferring it. If they are the same, the transferred file is an exact duplicate of the original. Hashing can even be used for user authentication processes as Vidya Rao and Prema K.V describe in their article about the IoT authentication mechanism that was based on hashing functions [87]. An example of the authentication process that was performed in their research is presented in Figure 2.11.

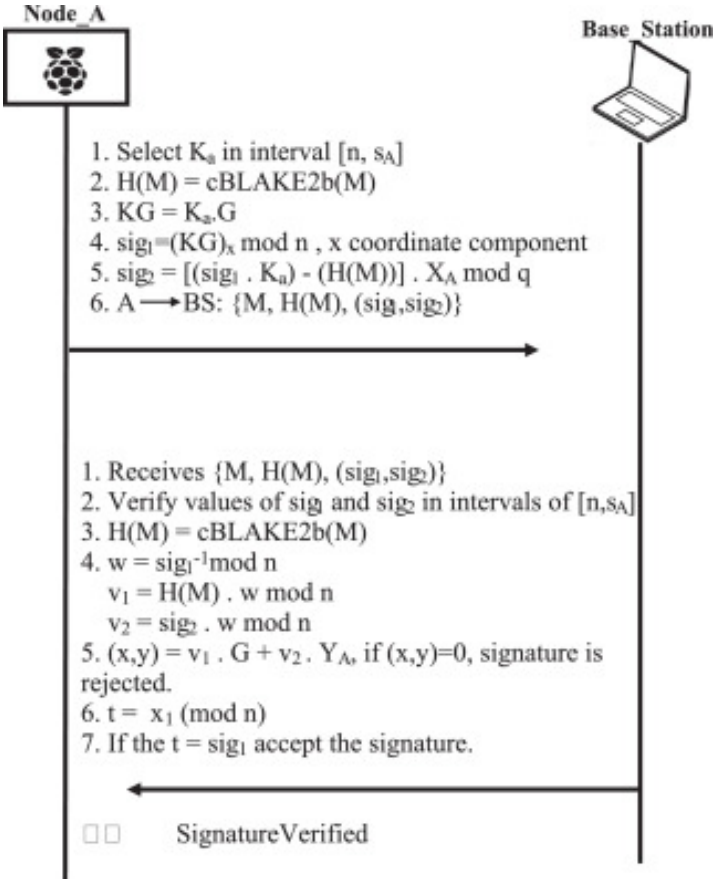


Figure 2.11: Signature and verification process of authentication phase using a hash function [87]

When a single bit or byte of data inside a file is altered, a decent hashing algorithm will show a characteristic known as the *avalanche effect*, which means that the resultant hash output will change substantially or completely [85]. The lack of such randomization is regarded as inadequate randomization, which makes it simple for attackers to breach the hash function's security [88]. Additionally, in certain cases, an encrypted file may be constructed such that neither the file size nor the date and time of last modification are ever changed by the user (for example virtual container files [89, 90]).

While it would be difficult to determine at a glance whether or not two identical files are distinct, the hash values would be able to identify the difference between them if they were different. Therefore in order to avoid producing the same hash result from two distinct inputs, a decent hash algorithm should be complicated enough to avoid this. Hash collisions are a kind of collision that occurs when two hashes

match [86]. In order for a hash function to be good, it must be complicated so the results is unique and robust, avoiding hash collisions, but not so complex that the hash computation would take too much time.The following paragraphs are referred to 2 of the most well known hashing algorithms and their algorithmic structure.

MD5 or Message-Digest algorithm is probably the most well known hashing function, among the MD family and other hash functions, and was created by Ronald Rivest, the same developer of RC4 and contributor to the RSA as we saw earlier, in 1991. MD5 normally generates 128-bit hash values consisting of what looks like random hexadecimal strings (Figure 2.12) that signifies the unique identifier of the file that hash been through the hash process [91]. Despite the fact that MD5 has shown some major vulnerabilities, such as the *birthday attack* exploiting MD5 with the MD5CRK [92], it is still the most commonly used hash function and can be found in protocols such as SSH, SSL and Internet Protocol Security (IPSec). Certain applications even use use salt values, which essentially is random data that guarantees uniqueness when mixed with the hash function which results in more hardened and secure hash value. However, MD5 is typically not recommended for wireless LAN implementations due to the possibility of exposing the user's password as well as the existence of several collision-based weaknesses that have been discovered. It only allows for one-way authentication so there is no mutual authentication between the wireless client and the network in this configuration. In addition, and perhaps most importantly, it does not provide a method for generating dynamic wired equivalent privacy (WEP) keys that are valid for only one session.

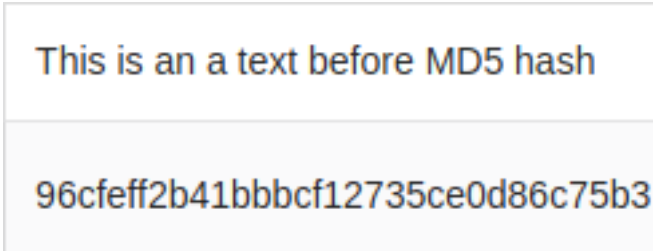


Figure 2.12: Text after MD5 hashing.

On the next list the MD5 algorithm is presented in steps, as described in Rachwati's et al. article [93] :

1. **Append padded bits** : The message if filled with a single *1*(binary) bit at the end of the message followed by enough zeros until the sum of bits are equal to 448 modulo 512
2. **Append length** : The result is appended with a 64-bit representation of the message's length, which is stored in the result. This step is necessary in order to ensure that the message length is an exact multiple of 512 bits in length.
3. **Divide the message** : Blocks of 512 bits are used as the input string which are divided into 16 sub-blocks each being 32 bits in length. The algorithm's result is a collection of four 32-bit blocks that are combined to create a single 128-bit hash value.
4. **Initialize MD Buffer** : Four 32-bit variable are initialized:

   A = 0x01234567

   B = 0x89ABCDEF

C = 0xFEBCDA98

D = 0x76543210

These are called chaining variables.

5. **Process message** : The algorithm is repeated as many times as the message length divided by 512, which is the length in bits of the blocks processed each time. Each of the four 32-bit variables of step 4 are assigned to a certain letter : a for A, b for B, c for C, and d for D. The main loop has four rounds, all very similar, where each of them uses a different operation 16 times. Each operation performs a nonlinear function on three of a, b, c, and d. Then it adds that result to the right a variable number of bits and adds the result to one of a, b, c, and d. Finally, the result replaces one of a, b, c, and d. There are four nonlinear functions:
   F(X,Y,Z) = (X **OR** Y) **AND** ((**NOT** X) **OR** Z)
   G(X,Y,Z) = (X **OR** Z) **AND** (Y **OR** (**NOT** Z))
   H(X,YZ) = X **XOR** Y **XOR** Z
   I(X,Y,Z) = Y **XOR** (X **OR** (**NOT** Z))
   (The letters in **bold** represent the logical operators AND,OR,XOR and NOT)
6. **Output** The message that was produced as output is A, B, C, D containing MD5 digest. Output begins with low-order byte of A, and end with the high-order byte of D.

SHA or Secure Hashing Algorithm, is a family of hashing functions which includes many versions just like MD. It was published by NIST in 1993 starting with SHA-0, with the addition of three more versions, namely SHA-1, SHA-2 and SHA-3. The most well known version is the SHA-2 with its variant SHA256 which indicates the number of input/output size in bits. Some other adaptations of SHA-2 are SHA-224, SHA-384, SHA-512, SHA-512/224, SHA-512/256 [94]. The SHA-256 hashing algorithm is one of the most secure hashing methods available on the market. The United States government mandates its agencies to use the SHA-256 algorithm to safeguard some sensitive information. However, although the precise details of how SHA-256 works remain classified, we do know that it is constructed from a Merkle-Damgård structure derived from a one-way compression function that is based on the Davies-Meyer structure which is a construction for a hash function based on a block cipher [95].

Three characteristics contribute to the security of SHA-256. In the first place, it is almost impossible to recreate the original data from the hash value alone, there would be needed 2^{256} attempts from a brute force attack to reconstruct the original data. Secondly, unlike MD5, data collision is not that keen on SHA hashing functions, as there are 2^{256} possible hash numbers that can be produced by SHA-256, so it very unlikely that two hash values would be the same [93]. Lastly, a small modification to the original data may cause a avalanche effect which I explained earlier, so the change is apparent to the recipient. I am also including the SHA256 algorithm below drawn form the same article as MD5 [93], which includes some steps similarly to MD5.

1. **Append padded bits** : The message if filled with a single *1* (binary) bit at the end of the message followed by enough zeros until the sum of bits are equal to 448 modulo 512
2. **Append length** : The result is appended with a 64-bit representation of the message's length, which

is stored in the result. This step is necessary in order to ensure that the message length is an exact multiple of 512 bits in length.

3. **Parsing the message** : The padded message is parsed into N 512-bit message blocks, M^1, M^2,…, M^n, by appending 64-bit block.

4. **Initialize Hash Value** : The initial hash value, H^0 is set, consist of eight 32-bit words, in a hex-adecimal form.

5. **Process message** : SHA256 uses a message schedule of 64 words, each 32 bits long. The words of the message schedule are labeled W_0, W_1,…, W_{63}. Where :

$$Wt = \begin{cases} M_t^{(t)} & 0 \le t \le 15 \\ \sigma_1^{(256)}(W_{i-2}) + W_{i-7} + \sigma_0^{(256)}(W_{i-15}) + W_{i-16} & 16 \le t \le 63 \end{cases}$$

Figure 2.13: Operation in the SHA256 algorithm [94]

$\sigma_1^{(256)}(W_{i-2})$ = ((W_{i-2}) ROTR 17) **XOR** ((W_{i-2}) ROTR 19) **XOR** ((W_{i-2}) SHR 10)
$\sigma_0^{(256)}(W_{i-15})$ = ((W_{i-15}) ROTR 7) **XOR** ((W_{i-15}) ROTR 18) **XOR** ((W_{i-15}) SHR 3)

6. **Initialize the eight working variables** a, b, c, d, e, f, g, and h, with the(i-1)st hash value For t=0 to 63:

$T1 = h + \Sigma 1^{(256)}(e) + Ch(e,f,g) + K1_{256} + W_t$
$T2 = \Sigma 0^{(256)}(\alpha) + Maj(a,b,c)$
H = G
G = F
F = E
E = d + T1
D = C
C = B
B = A
A = T1+ T2

Where:
$\Sigma_1^{(256)}(e)$ = (e ROTR 6) **XOR** (e ROTR 11) **XOR** (e ROTR 25)
$\Sigma_0^{(256)}(\alpha)$ = (e ROTR 2) **XOR** (e ROTR 13) **XOR** (e ROTR 22)
Ch(e,f,g) = (e **OR** f) **XOR** (**NOT** e **OR** g)
Maj(a,b,c) = (a **OR** b) **XOR** (a **OR** c) **XOR** (b **OR** c)

7. **Output** : After repeating steps one through four a total of N times, the resulting hash function is : $H0_{(N)} \| H1_{(N)} \| H2_{(N)} \| H3_{(N)} \| H4_{(N)} \| H5_{(N)} \| H6_{(N)} \| H7_{(N)}$

2.4 Communication & Security Protocols

Of course the first and foremost well known communication protocol is HTTP that has been around for many years and together with HTML, they are accountable for bringing the World Wide Web or Internet to life. HTTP is designed for distributed, collaborative hypermedia information systems and allows users to interact with online resources such as HTML files by sending hypertext messages between clients and servers using the HTTP request-response protocol, which is a request-response protocol [96]. HTTP makes use of certain request methods in order to accomplish a variety of functions. All HTTP servers support the GET and HEAD methods, but not all of them support the other request methods, which are as follows:

1. **GET** requests a an entire resource with all its components.
2. **HEAD** requests a resource with information regarding only the content of the head.
3. **POST** adds data to a new page under an existing web resource.
4. **PUT** directly changes an existing web resource or, if necessary, generates a new URI.
5. **DELETE** removes a specified resource.
6. **TRACE** shows modification history of a web resource.
7. **OPTIONS** displays the HTTP methods available for the interested web resource/page.
8. **CONNECT** transform the request connection to a transparent TCP/IP tunnel.
9. **PATCH** modifies only specific attributes of a web resource.

HTTP however is not secured by itself, so the implementation of SSL/TLS are required in order to form a secure connection between the sender and the receiver. This is where HTTPS comes into play, which is a protocol that protects data sent over a network, primarily public networks such as Wi-Fi. As I said, due to the fact that HTTP is not encrypted, it is susceptible to attack by eavesdropping attackers who may get access to a website's database and critical information. Because of its bidirectional nature, HTTPS encryption ensures that the data is encrypted on both the client and server ends of the communication channel and only the client has the ability to decode the information received from the server [97].

In terms of communication protocols, HTTP has long been considered a web standard, but as IoT has progressed in many technological aspects, with devices typically having limited computational power and time constraints there has been a demand for more lightweight and responsive mechanisms. As a result, new protocols have been developed, namely, **CoAP** and **MQTT**, which similarly to HTTP both are request-response protocols.

CoAP or Constrained Application Protocol is a document transfer protocol similar to HTTP created by CoRE IETF group, with CoAP sending smaller packets, which are designed for constrained devices, over UDP instead of TCP, meaning that CoAP can tolerate packet loss and recommit the lost packages with ease. On the other hand, data transmission is not reliable due to the absence of QoS in UDP while at the same time being susceptible to packet amplification and IP address spoofing leading to security vulnerabilities such as DDoS attacks as described in Vasques and Gondim's article [98]. CoAP security

is based on DTLS which provides communication encryption and privacy for datagram protocols sharing homogeneous mechanisms to TLS, viz. modified version of TLS handshake, with the exception of few of them being optional in DTLS while fixed in TLS, such as replay detection which ensures QoS [99].

MQTT or Message Queuing Telemetry Transport is a lightweight, open standard communication protocol released by IBM in 2010 which performs an asynchronous connection among the client and the server which renders MQTT applicable for M2M communications and IoT applications [100]. MQTT consolidates in QoS through five header fields that determine the QoS of a message depending on the scope of transmission as Dr.Jaloudi explains in his article [101]. Due to the QoS that MQTT offers there is a huge reliability regarding the packet transmission, although if the network environment is not strong enough, retransmissions may occur which could cause more network congestion.

Despite the fact that the MQTT and CoAP protocols may be beneficial in certain conditions, particularly speed and weightlessness, they do not necessarily imply to the exigencies of IoT security the way that HTTP does. We described about the drawbacks of CoAP being a UDP based protocol thus being vulnerable to common security attacks and concurrently not conveying the impression towards IoT standards of QoS. Although MQTT is already widespread and one of the most commonly used protocols in the IoT industry according to Eclipse's IoT Developer Survey 2019 [102], it goes without saying that limitations are presented in this protocol's aspects as well. In recent research [103], it has been evinced that MQTT security is still underdeveloped whilst applications based on MQTT have been censured for defective implementations in real world scenarios [104, 105] due to the paucity of security [106]. Rodriguez and Batista [107] conducted a full on experiment analyzing the flaws of MQTT's security and which frameworks could possibly mitigate these issues.

2.5 Related Security Surveys in 5G & MEC IoT

There are hundreds of articles, surveys, experiments and analysis upon the topic of security in IoT. Due to IoT's fast growing market and the industry demand for faster processing, developers try to go above and beyond expectations so they can have an end product delivered to the public, without having many security concerns at first. Unfortunately the demand of security still stays relatively high thus many researches have proposed their solution for this problem. In the next paragraphs I am going to analyze the work of other researchers that have developed a solution or idea regarding IoT security and future technologies, which I personally found very interesting.

In the study of GSMA *5G security issues* [108], they analyze the transitioning progress and process from older communication technologies, such as the 4G which most phone carriers support throughout the globe, to renovating and state-of-art technologies such as the 5G combined with IoT. They believe that the rise of 5G will bring an uprising to cyber criminal activity and that the introductions of new and more complicated additions that the 5G carries will open up many back doors and weak spots for intruders due to the number of errors expected that the 5G will have for the first few years at least. Some of these security threats will not be a creation of a new hacking attempt but rather security threats that existed in previous telecommunication generations. They quote *"Security threats associated with 3G and 4G will remain after 5G reaches the public and will heavily influence NR deployments on the horizon of three to five years"*. This study supports that due to the new technological additions that 5G will be consisted

of, such as HTTP and TLS which attackers already are familiar with, hacking into a 5G communication will be as easy as getting into a website. 2G and 3G were build on the SS7 protocol while 4G relies on Diameter and their whole technological structure is shown in Figure 2.14. This paper is very informative when it comes to the drawbacks of mainstream 5G IoT infrastructures and gives a good perspective of what to expect when dealing with a new technology such as 5G. It is really important that the technology of 5G is thoroughly analyzed where the drawbacks and flaws are drawn, so anyone who wants to study and apply the 5G technology for their communication protocol, can understand the risks that come along the innovation of this technology.

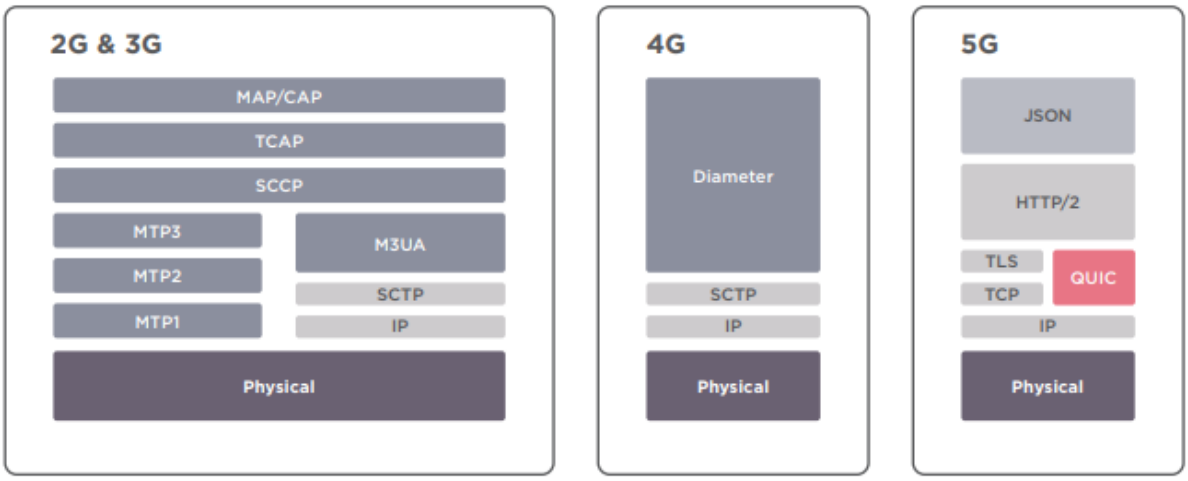


Figure 2.14: Network structure of telecommunication technologies [108].

Singh et al. [6] searched deep into the challenges of cyber threats that could tamper with IoT devices and developed an article dedicated on lightweight encryption solutions which could possibly constrain or prevent several security. They do not focus on a certain encryption model but rather compare the most well known cryptography mechanisms and test the correspondence and performance of each methods to comprehend and determine what is the best encryption scheme when combined with low powered IoT devices.

Although RESTFull implementations are widely used for Web applications, IoT applications greatly benefit from publish subscribe based secure implementations, recently studied by Jurion et.al [109]. This research focuses on developing a secure cloud based system which is based on IoT best practices but also combine new and efficient products so the end result could be perfectly modeled to serve low powered IoT devices in lightweight security while all being cloud connected. Their research is also based on the MQTT protocol and they believe that this technology can be standardized when it comes to generalized 5G platforms (Figure 2.15 [109]). Lastly the analyze, which encryption method can give the best security outcome while also requiring the least power consumption.

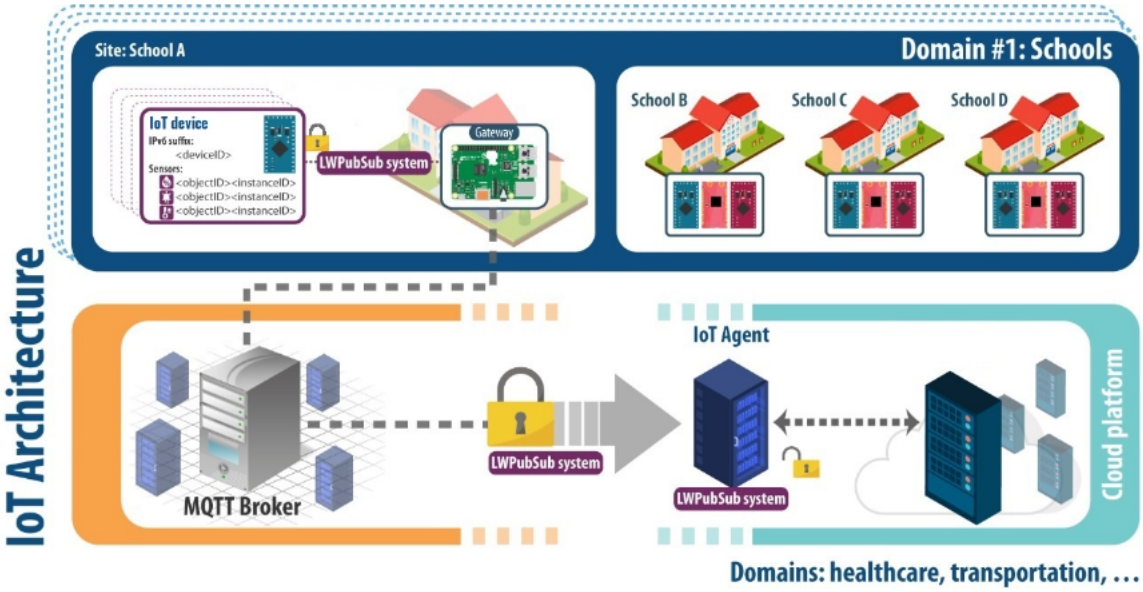


Figure 2.15: Proposed IoT scheme based on MQTT [109].

2.6 Epilogue

The security of an IoT system can be a difficult and frightening task when you are confronted with all the possible threats that could harm your system any time. The threats that I mentioned in Section 2.1.1 could cause a lot of damage to you system, even shutting the whole infrastructure down, if the issue is not addressed properly. Therefore it is very important to understand how the available solutions work and be able to apply them to your own system based on your interests and configuration. In the next chapter I am going to demonstrate a full representation of my work from the base, which is the infrastructure, up to the little details which make the framework and encryption stand out.

Chapter 3: Research design in real 5G MEC environment

Now that we understand the term and usability of the encryption in a system, it is time to test it on a real MEC environment where we check and monitor in depth how and where encryption mechanism is applied, how is this mechanism created and what components or tools are needed. In order to achieve the expected results and explore the novelty of the whole idea of encryption I needed a platform capable to perform 5G communications, together with the team from Smart Internet Lab [17] of Univeristy of Bristol [16] I was working with during my internship there, we developed a IoT based platform which was perfect for the application of encryption as it is based on 5G communications with the convenience of being private. As mentioned on a previous section, private 5G networks are very beneficial and can accelerate the production rate of industries, however it is important to keep this private network secure while maintaining the QoS [110].

3.1 IoT Platform Environment Design

In the next subsections I will be analyzing the structure and the main components of the platform, each of these components is crucial for the development of IoT services and was created from our team. I will be referring how each part is secured and how the encryption protocol is applied in each occasion.

The infrastructure is based to work with a client-server configuration where each client has some IoT devices to their belonging and needs to monitor or retrieve the data from that device in some way. What our platform does is gather the data from each user's device and deliver to the recipient upon their request, while at the same time the communication channels remain secure so the data are delivered exclusively to the client. Security in our system is also enhanced through authentication and authorization using oAuth2, an authorisation protocol that uses a token introspection schema through specific authorization flows so only authorised users can use our platform [111].

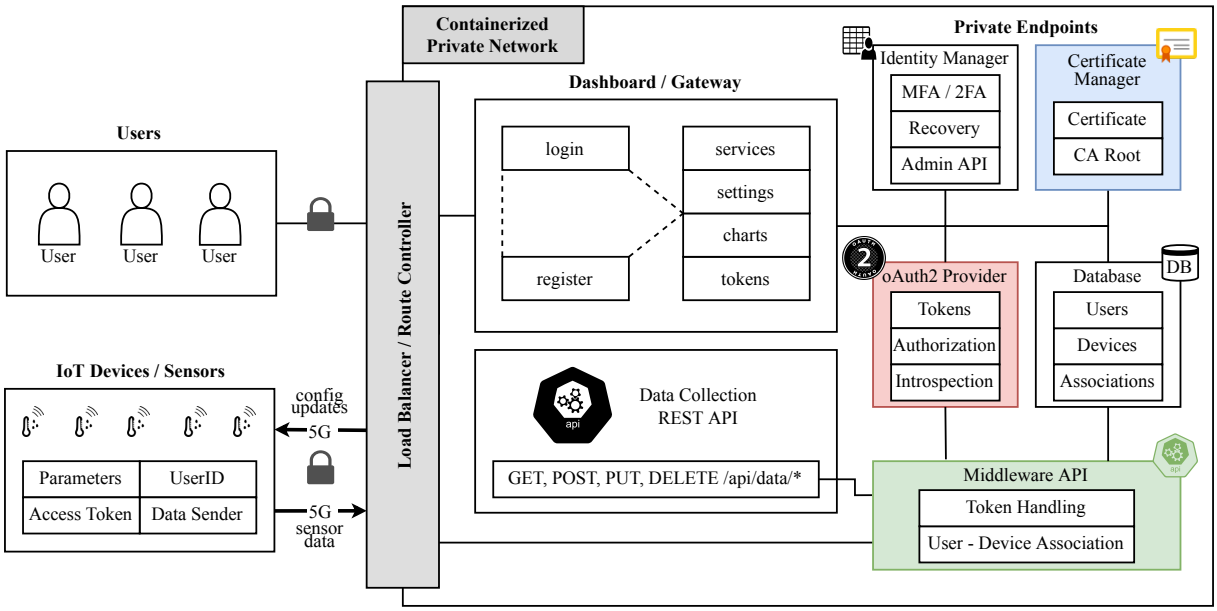


Figure 3.1: Platform Architecture Diagram

3.1.1 Platform Architecture

Figure 3.1 displays the whole structure of our system, the way the information flows through the platform and how the components connect to each other in order to pass the data from the device to its user. Each component is then explained, their functionality and their main role in the infrastructure so you can have a better understanding of the system.

3.1.2 Dashboard

The dashboard is a complete web hub offering users connection to the platform. Here you can also notice the first security act of authentication as it requires login or registration from a user. In the dashboard users can use a number of utilization options including device configuration, such as the device's status and data transmission frequency, as well as monitor these devices and read the collected data. These metrics are also visualized using Angular and WebSockets which makes our dashboard very user friendly and enriches the overall UI, an example is presented in Figure 3.2, where the user selects the device and metric of that device. Additionally, the dashboard serves the system's endpoints services and applications so the user can retrieve and interact with the endpoints in order to retrieve important data and files such as the token needed for authorization and SSL certificate for connection encryption. Typescript and ExpressJS were used for the back-end creation of the dashboard as well as Handlebars for the front-end part.

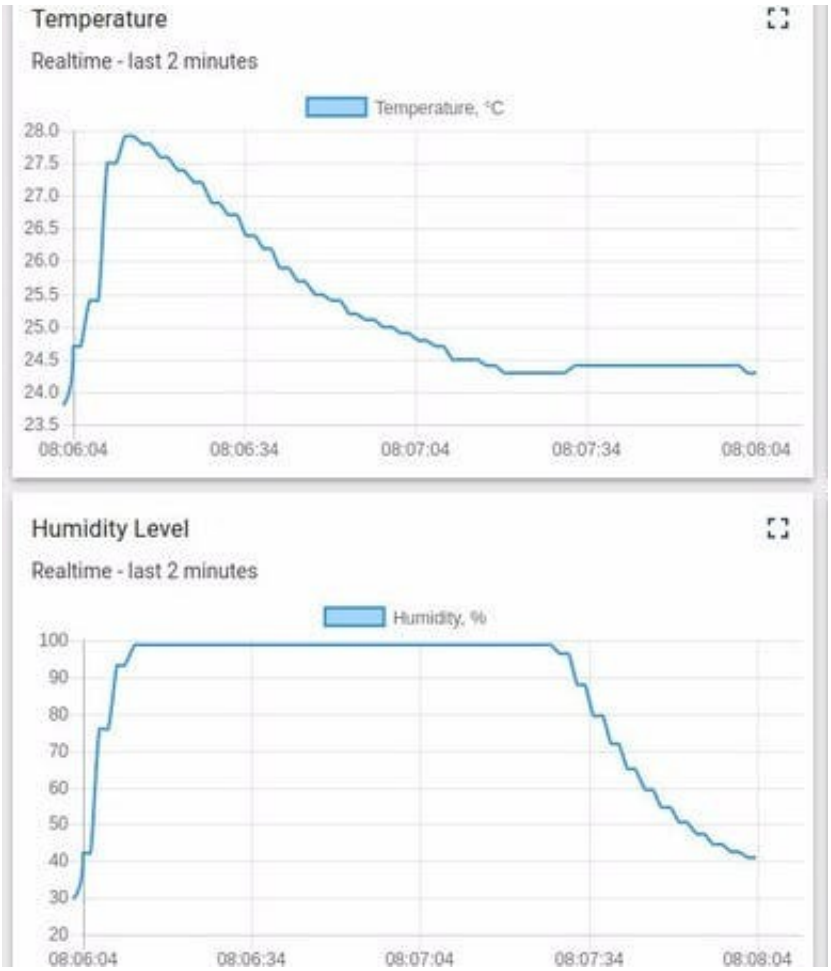


Figure 3.2: Metrics visualization in dashboard

3.1.3 Identity Manager

The identity manager is a policies framework which is responsible for the system's authorization, a crucial role for the security of our platform. In our case we have gone with Ory Kratos, an identity manager that is a cloud native and provides user login and registration, allowing users to create their own account using the username/password and email schema or use a social sign in option (Google,Github,Facebook etc) with account verification so it can verify that the email,physical address, or phone number belong indeed to an identity(user).It also includes multi-factor authentication with third party protocol support such as OTP or Google Authenticator and user information storage with a headless API. Additionally, Ory Kratos takes cares of the user providing account verification in case someone forgot their password and profile or account management so the user can update personal information or passwords. Kratos is also admin friendly providing Admin APIs, such as adding or removing a user and can work with any UI framework using just a few lines of code.

3.1.4 oAuth2 Provider

As the name suggests, oAuth2 provider is in charge of providing authorization to our system, so we had to find a provider that could work swimmingly when combined with our other frameworks, such as the Identify Manager. in order to maintain the system and QoS stable. Luckily, Ory also offers an authorization framework that has oAuth2 already implemented named Ory Hydra and can be operated combined with Kratos without any conflict, besides that Hydra is flexible when it comes to user management as it can support a identity manager of your choice. On top, Hydra offers a cryptographic key storage which is great for our certificate keys safekeeping that are introduced in Chapter 3.2, so we found the combination of Ory Hydra with Ory Kratos to suit our purpose the best. Ory Hydra has low memory and CPU footprint, scales easily and is very responsive making it ideal for IoT devices that tend to have low power hardware and in most cases use a battery as their source of power. oAuth2 protocol uses a token schema where each user has a unique token which is later passed to each of their devices and links the user to the devices. The token expires after a certain amount of time, meaning that the user and the devices are no longer authorized and can no longer send data to the platform. A token introspection method is used to check the status of the token and create a new one if the one it is checking has expired using a refresh token [112].

3.1.5 Database

There are plenty of RDBMS in the market that have been commonly used for years in various IT applications, but since we are evolving to a new era and data transmission rates have changed, we opted for a database with improved flexibility while still having well based security features. The databases we are working on are based on MongoDB, which is a NoSQL solution that delivers flexible-schema data storing non relational data with horizontal scalability. MongoDB stores data in a key-value set as a series of JSON-like documents which is great for our platform, as many of the other components produce and export data in JSON format hence making the saving process way easier. In a journal conducted by Damodaran et al. [113] MongoDB was compared to MySQL, one of the most well known RDBMS, in terms of basic DB functions and you can really observe how much faster and scalable MongoDB is in the Table 3.1 and Figure 3.3 [113].

Table 3.1: Comparison of MongoDB with MySQL [113]

Operations	No.of Records	Execution Time (in ms)	
		MongoDB	MySQL
INSERTION	100	0.01	0.01
	1000	0.5	1.25
	10000	1.2	2.2
	25000	2.25	3
SEARCH	100	0.05	0.152
	1000	0.12	1.52
	10000	0.55	4.47
	25000	1.25	5.21

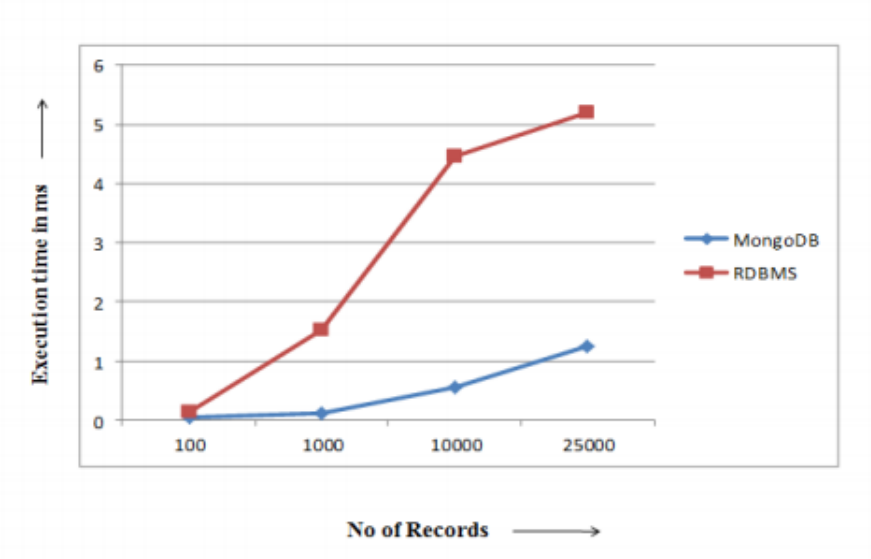


Figure 3.3: MongoDB & MySQL search operation comparison [113]

3.1.6 Certificate Manager

The Certificate Manager is the key component for the encryption of our communication channels, those being from user to platform, from devices to platform and vise versa. The manager produces and stores the certificates and keys needed for the encryption process and can be also be used by other endpoints such as the oAuth2 provider where it was mentioned that it has its own cryptographic key storage, thus it can store the keys created from the certificate manager there so keys don't need to be transferred in every transaction, increasing the data traffic and possibility of key exposure [114]. Further information about the encryption mechanism are presented in Chapter 3.2.

3.1.7 Middleware API

Our system was designed to constantly receive data from the devices, each having different information, different recipient and most importantly different oAuth2 token. We needed an API to which we could insert all these data and be able to redirect each of the information to the corresponding node, while at the same time communicate with our other components and interact with them in order for our system to work swiftly. What we came up with was the Middleware API, which was developed in Typescript and is the heart of our system, forwarding data from the devices to the rest of the components in the right order, so for example data that need to be stored are sent to the Database and the oAuth2 token is sent to

the oAuth2 Provider for introspection.

The API analyzes each request checking the token of the device, with the help of the oAuth2 Provider, before launching the data to the system, if the token is not valid or has expired, the API rejects the request and responds with a status code accompanied by an error message. It then checks in the DB if the token it just checked corresponds to any user and asks the oAuth2 Provider if that user has a new token, the provider responds with the new token to the API which forwards the token to all the devices owned by that user and not only to the device that was rejected, this way the API will not have to repeat this process for every device that had that expired token, saving some important time for our system. Then the device that originally was rejected can re-transmit the API request [115]. However, if the token is not expired the procedure can carry on, meaning that the API should push data to the other components. The DB receives and stores 3 important information from the device, the userID which indicates to which user that device belongs to, the deviceID that is each device's unique identifier and the token that the device had at that certain request. These 3 information are stored and associated with each other so the system can easily perform procedures such as the case I just mentioned where a device has a expired token and tries to send data to our platform. Lastly the data that the device is transmitting, those being metrics a sensor could have such as humidity, temperature, pressure, etc., are passed to the Data Collection REST API. The diagram in Figure 3.4 depicts the flow I just mentioned, the green part is when a token does not have an expired token, while the red part is the procedure the Middleware API has to follow in order to restore the token of the device.

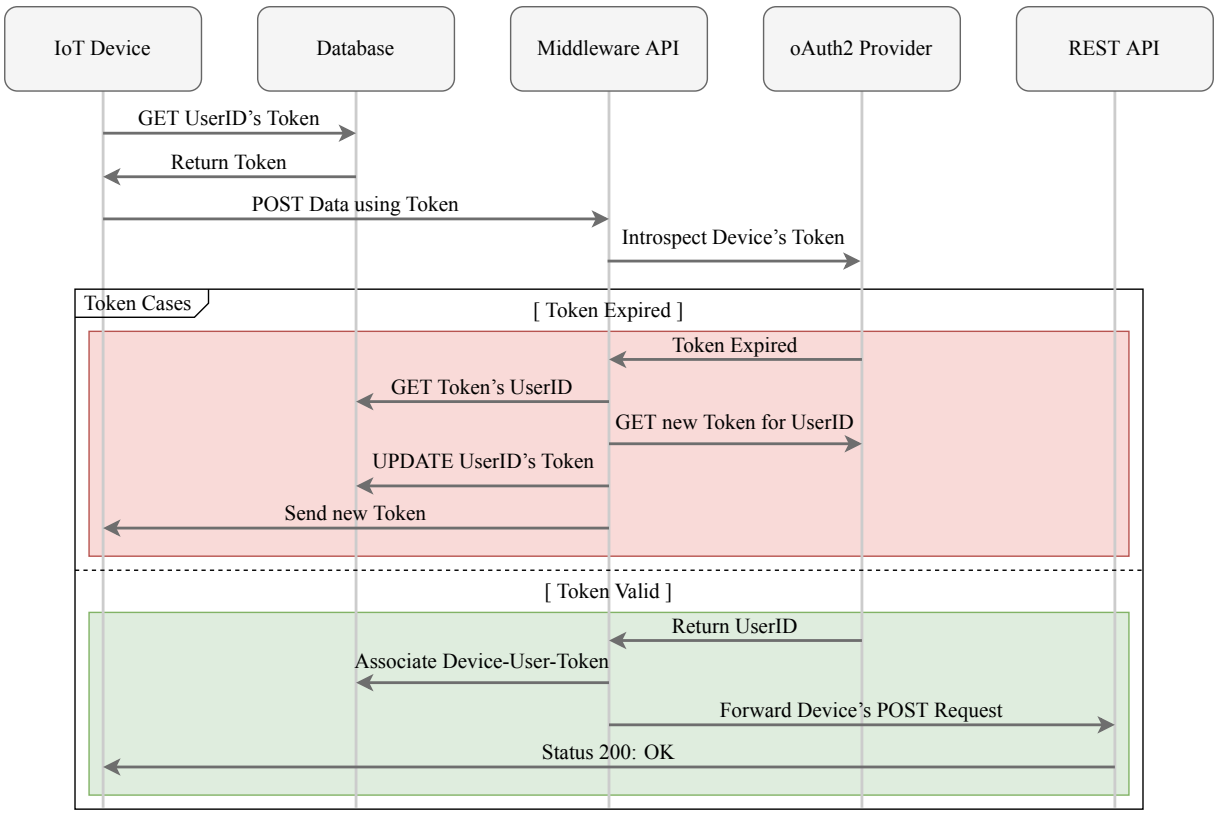


Figure 3.4: Device-Middleware API Flow

3.1.8 Data Collection REST API

Our Data Collection REST API (DCRA) is based on the Orion Context Broker(OCB) data model from Fiware [116], which is a NGSIv2 REST API designed in C++. What the infrastructure needed was a API request handler that would manipulate data according to the action that is required with each request. OCB has been proved to work well with IoT platforms based on an experience on agricultural research of Franco et al. on monitoring the growth of Ocimum basilicum seeds [117]. One event would be gathering metrics from sensor with a POST request header so the data can be stored accordingly to the database. Similar events would be a user updating, removing or configuring devices or data from the database. We have implemented authorization and authentication in the DCRA, meaning that data are distributed to the corresponding user's database record and cannot be seen by others.

3.1.9 Users

Users do not contribute to the structure of the platform but rather serve the purpose of one end when referring to end-to-end communication. The user is first introduced to the Dashboard where we explained the procedure users have to follow regarding the login or registration and must later consent the use of oAuth2 tokens so the devices can be authorized. From that point the user has full access to their personal settings and the devices they own, meaning they can change their password, their devices status and configuration, check devices metrics through visualized charts.

3.1.10 IoT Devices

These devices are usually sensors like those pictured in Figure 3.5,actuators and gadgets of any size located in any desired place, it can be inside a greenhouse for agricultural monitoring where the user needs to check the humidity or the temperature so the crops don't go bad, or it can be simply be attached to a bicycle so the user can always know its location. Whichever the case, all those devices communicate to our platform through 5G connections using 5G antennas as shown in Figure 3.6 which shares similar aspects to the antennas that already in use for mobile communications. The devices have configurable parameters which a user can tweak, such parameters could be the type of metrics the sensor is measuring or the frequency of the measurement, this way the DCRA doesn't need to store excess information for the device that the user will not need, saving up storage and data traffic throughout the communication channels. Each device saves the access token from its owner and is also assigned a unique identifier (deviceID) which is useful for procedures such as the token introspection that happens during the request the device sends over to the Middleware API.

3.1.11 Load Balancer & Containerised Private Network

An IoT platform has to process a lot of I/O data through its system and must do so in an efficient and smart way, so data needs to be spread evenly across the hardware so the work rate remains stable and well optimized. For this reason we used a load balancer performs a distribution of network or application traffic across multiple resources (usually servers). The load balancer is located between client devices and backend servers, receiving and then distributing incoming requests to any available server

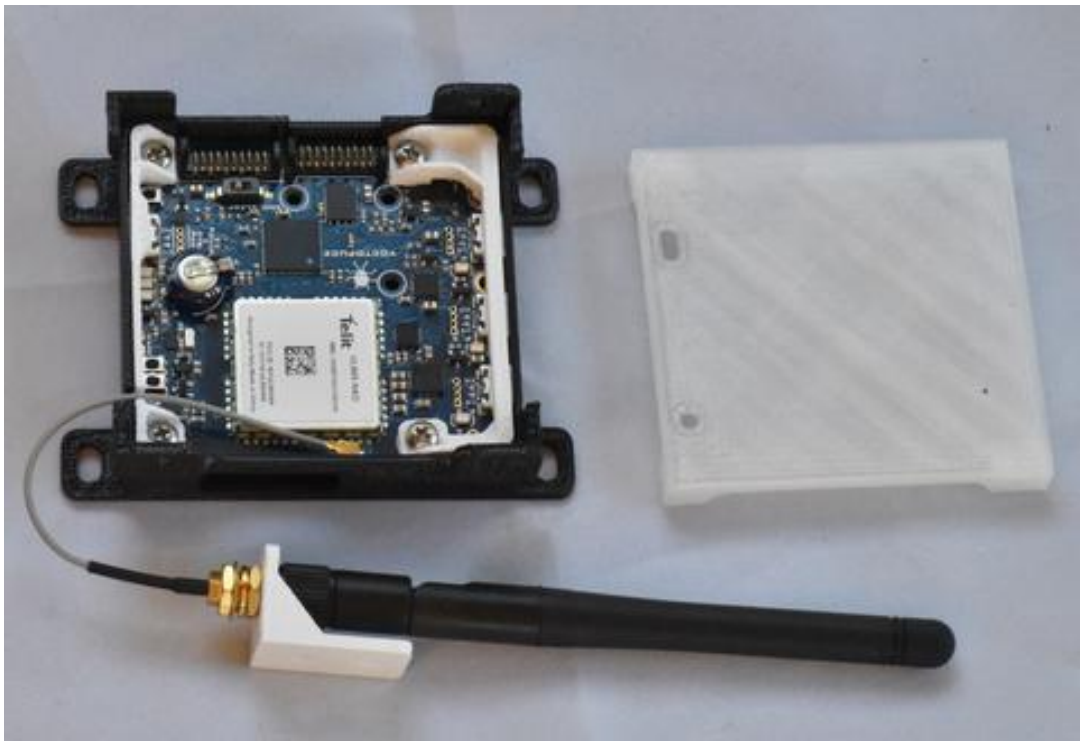


Figure 3.5: IoT sensor with wireless connection antenna

Figure 3.6: 5G antenna tower

capable of fulfilling them. Load balancing is very beneficial for our IoT platforms and has been used for other IoT based projects [2] where in our case it can take advantage of the fully capable servers located at the University of Bristol and have game-changing results when it comes to RTT times of data packages through the system, the RTT times of the load balancer can be observed in the Table 4.1 where we use two heterogeneous units (Physical Switch) through the load balancer. Our choice of load balancing was NGINX [118], a free high-performance HTTP server that uses scalable event-driven (asynchronous)

architecture instead of handle requests threads used by traditional servers and can handle thousands of simultaneous requests, while having a small memory footprint and high-performance at the same time. NGINX can also scale in many directions, from smaller devices all the way up to complex server farms, NGINX has been a choice for numerous commercial sites like Github,Airbnb and Netflix [119]. Additionally, NGINX also serves as a reverse proxy and IMAP/POP3 proxy server, which we used for our links redirection in our Dashboard, packing and organizing our backend even more.

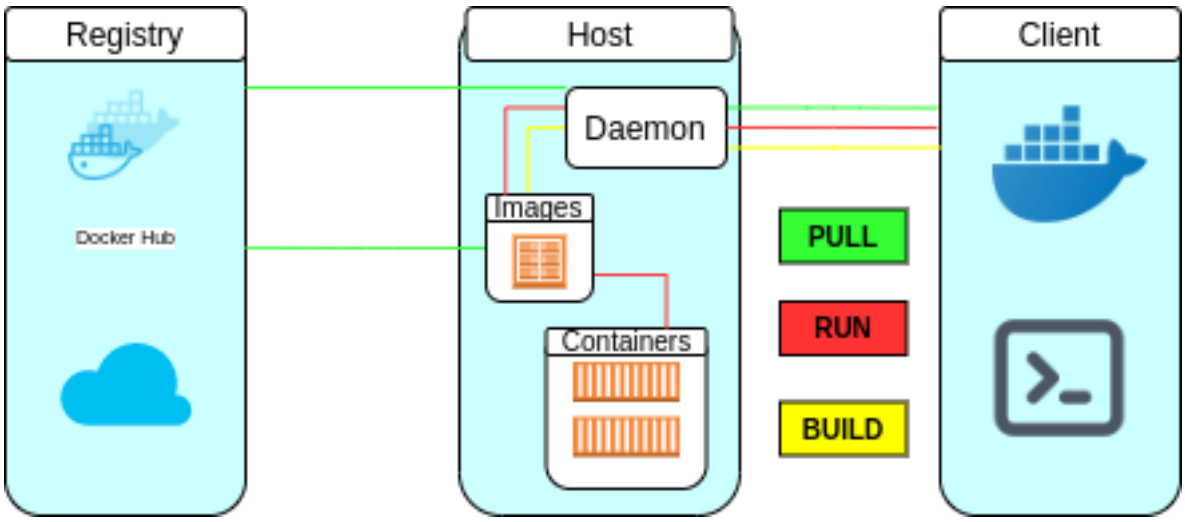


Figure 3.7: Docker platform architecture

As for the private network, we needed a space where each component was organised in their own space due to the diversity of technologies each component uses, so every developing environment has the required libraries, plugins and configurations for the technologies the component is going to user. The best solution for our platform was Docker [120], an open source PaaS software which offers network virtualisation that containerises each of the applications. Each container is isolated from others and can communicate with the rest of the containers through well-defined channels, Docker's architecture is also visualized is Figure 3.7. In our case each of the compartments of our platform has their own container complete with a personalized IP address and a private storage while the communication is done through the network bridges inside a Docker Network. Docker is used in many applications and recently has been a developer's favorite in the IoT industry due to the organising nature of saving the process in the cloud together with the compatibility it offers for many OS, so it has an easy setup procedure if case the project needs to be migrated [121].

3.2 Encryption Method

At this point you understand the whole architecture of our system, how each each components works, how they communicate and their importance in the system. Although one thing that wasn't mentioned is how these components and their communication are secured, which is the important part that we are going to analyse in the experiments that we have conducted in the Smart Internet Labs to check how encryption prevents the data from being stolen and manipulated from eavesdropping attacks and how the encryption overhead affects the communication throughout the system. What I created for our system is a private encryption technique that uses custom made certificates which are configured to our needs and

are up to the latest encryption standards.

3.2.1 Configuration

Our encryption is based on a combination of symmetric and asymmetric encryption taking advantage of the benefits of each method to achieve the best results for our encryption together with a fast processing process of encrypting the data so the whole process does not have a huge impact on our system. The asymmetric encryption method has one major disadvantage, that it is very complex and therefore very time-consuming. No matter how fast a computer is, we wouldn't want to use them to encrypt/decrypt large amounts of data. An encryption software normally uses a symmetric algorithm on the data itself, meaning that a secret key has to be sent to the client, however there is still a possibility that an eavesdropper steals the key and intercepts our data. To avoid situations like these, the system encrypts the private key itself with an asymmetric algorithm, which is also known as session key since it is created when the two ends first connect. The session key is used for all the data transmissions until the session is finished and the connection ended,so for every session a new secret key is generated. Overall, symmetric encryption offers speed and efficiency even when processing large amounts of data while the asymmetric encryption makes sure that the private key is never exposed and is kept well hidden in our system, decreasing the change of cyber attacks discovering our keys during transmission.

The procedure starts as we decide the encryption mechanism and cipher we want to use. There are plenty encryption combinations someone can use the list of combination is never-ending, although I went on with modern solutions which are proved to work great and are contemporary to the security of IoT [56]. The protocol that we used for encryption of the communication and the channels is HTTPS, which is an extension of HTTP that has been the main communication protocol for many years where people have been applying security on HTTP since its creation [97]. For the creation of our key and certificate I used OpenSSL, a commercial-grade all-around cryptography library that offers a full featured application of the TLS and SSL which we explained in Chapter 2.2 on how TLS or SSL works and how it has evolved over the years. OpenSSL is required to create the encryption key and digital certificate for our system which we will explain the procedure later in this section. The key and certificate created by the OpenSSL toolkit are pre-configured to work properly to the system yet, first they must be sent to a CA. The CA authority is entity that signs the OpenSSL certificate and verifies the ownership of a public key by the named subject of the certificate. A CA uses its private key to create a digital signature on the certificate that it issues to validate the certificate's origin (Figure 3.8). Others can use the CA certificate's public key to verify the authenticity of the certificates that the CA issues and signs, meaning that the subject (owner) of the certificate and the party relying upon the certificate trust each other . The format of the certificates is based on the X.509 or EMV standard [81]. Of course it would be easy to just create a certificate and then send it to the CA to sign it, applying this certificate then to our endpoints so they would transfer over HTTPS, but the problem is that our network is private and we wanted to keep things that way, meaning we wanted to have security sourcing within our own system to the other endpoints of our network. I decided that our server should also become our CA, so Certificate Manager's role in the platform (Figure 3.1) is to also sign and certify the certificates besides creating them.

The first step is to create the key that is going to be used for the signing of the certificate. As I mentioned, I used OpenSSL to generate the key that is based on the RSA algorithm to create a 4096 bit key, which

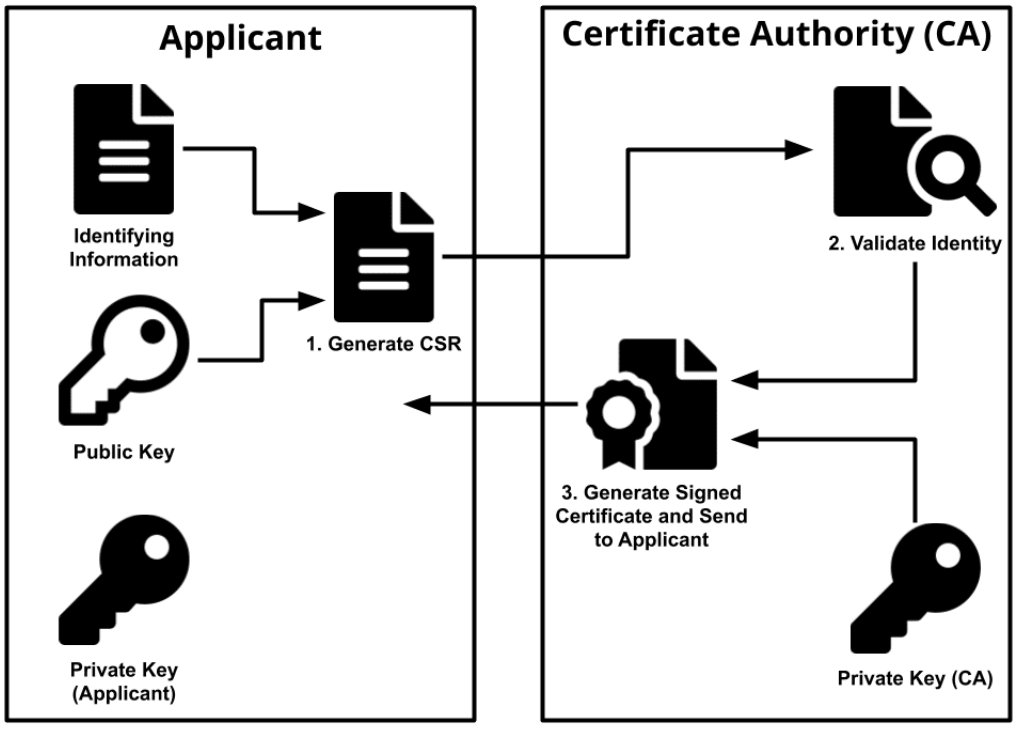


Figure 3.8: Digital certificate process

can observed in Figure 3.9. The RSA encryption method itself is slow so it does not encrypt data itself but rather combined with another encryption ciphers like the Triple-DES (3DES) [68] or AES256 in order to achieve maximum security regarding the key [73]. After the key is created, a CSR should be generated using that key which will later be combined together to create the certificate.The command and parameters for creating a key should look this:

```
$ openssl genrsa -des3 -out server.key 4096
```

For the CSR with pass the key just got created and the name of the CSR file :

```
$ openssl req -new -key server.key -out server.csr
```

I used a 4096 bit key, which might seem like an exaggeration compared to 2048 bits that most RSA keys use. According to NIST's Special Publication 800-57 [75] where they suggest that the transition of cryptohraphic algorithms and key lengths standards will have to change from 2030 as a 2048-bit RSA key is almost equivalent to Security Strength of 112 (Figure 3.2 [75]).

Security Strength corresponds to the amount of work that is required to break a cryptographic algorithm. That means that the higher the number is, the harder it will be for the algorithm to be broken (Table 3.3 [75]). I tried the 4096-bit key since we are working on a hybrid IoT platform with state-of-art equipment and hardware that were provided by the University of Bristol [16], so I didn't have any difficulties regarding the processing of the keys.

Afterwards, the author will have to fill the following X509 attributes of the certificate. It is important that the fields are filled in with the fully qualified domain name and information of the server so it can

Figure 3.9: The encrypted RSA key

Table 3.2: NIST expectations of RSA key lengths [75].

Security Strength	Through 2030	2031 and beyond
<112	Disallowed	Disallowed
112	Acceptable	Disallowed
128	Acceptable	Acceptable
192	Acceptable	Acceptable
256	Acceptable	Acceptable

protected by SSL :

- Country Name (2 letter code) [GB]:
- State or Province Name (full name) [Berkshire]:
- Locality Name (eg, city) [Newbury]:
- Organization Name (eg, company) [My Company Ltd]:

Table 3.3: RSA key length with corresponding Security Strength [75].

Security Strength	RSA key length
<= 80	1024
112	2048
128	3072
192	7680
256	15360

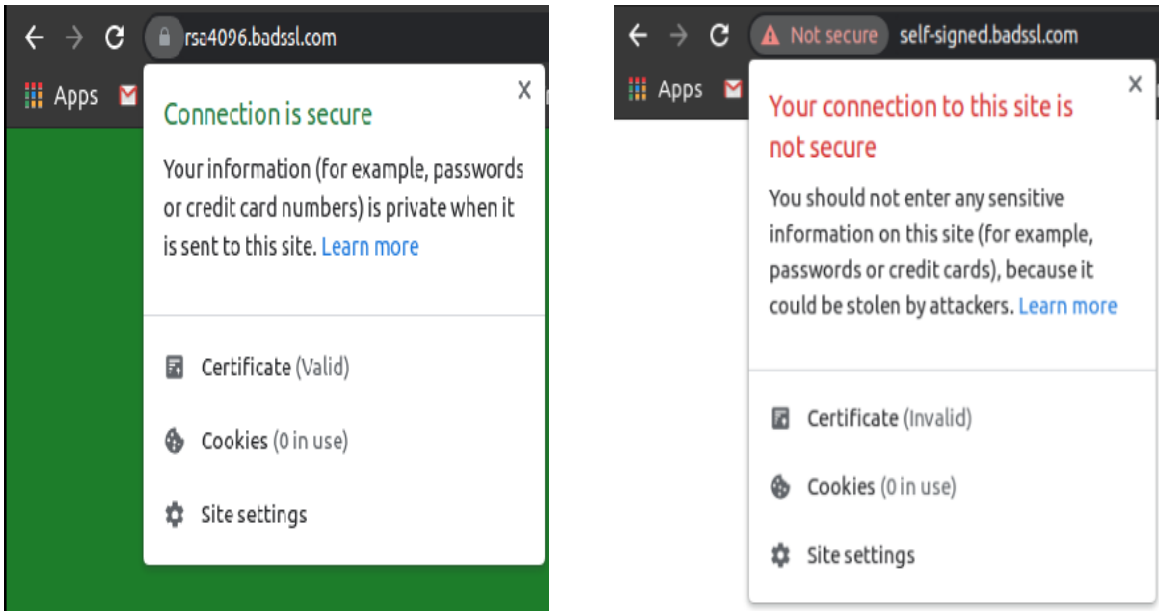
- Organizational Unit Name (eg, section) []:
- Common Name (eg, your name or your server's hostname) []:
- Email Address []:
- Please enter the following 'extra' attributes to be sent with your certificate request
- A challenge password []:
- An optional company name []:

Note that the two last parameters are optional and can be left blank if the publisher wants to. The challenge password is an option that was primarily used by some control servers and panels that used this password for special actions like revoking the certificate, although it is not supported by all modern CAs. The 3DES and AES256 also make use of a pass-phrase as certain servers that use the key, like Apache, which will ask for the pass-phrase each time the web server is started [122]. Even tho, the pass-phrase enhances the protection of the key and the overall system, it is not necessarily convenient, as there will not always be someone around to type in the pass-phrase, in occasions like a reboot or crash. The use of the Triple-DES encryption for the key is additional and can be skipped , thereby no longer needing to type in a pass-phrase. However, if the private key is no longer encrypted, it is important to understand that the key may be readable not only from the root user. Due to our network being private and secured other aspects of our system like authentication and authorization of the user, I didn't have to worry about the key being compromised.

Now that the CSR has also been generated, it's time to proceed to the creation of the certificate. As I mentioned, the certificate will not be signed and issued by a verified CA, but rather a self-signed certificate will be generated which our server will create and sign. Most of the verified CAs are commonly found and included in most of the most browsers and OS trust store, such as Let's Encrypt,IdenTrust or GoDaddy just to name a few. That means that any certificate issued by those providers will be ready for HTTPS encryption, although those are not always totally accurate and could have their own issues [80,123]. After generating both the key and CSR files they can be used to create the certificate file which will be used for the encryption transactions. To generate the certificate I used OpenSSL again to pass the CSR and key as that I just created shown in Figure 3.8, while also adding the format of the certificate and the days the certificate is going to be valid for, these usually being X509 format and 365 days of lifetime. The next command shows how the parameters should be used :

```
$ openssl x509 -req -days 365 -in server.csr -signkey server.key -out
server.crt
```

After running the command an "Signature OK" message is presented, which indicates that the process



(a) Secure (b) Not secure

Figure 3.10: Left (a) the connection is secure. Right (b) the connection is not secure because the certificate is invalid

of signing the certificate has been completed and everything worked fine. The problem here is that if a website uses a self-signed certificate for the encryption of the channel, the user will be greeted with a not secure or not private connection message as shown in Figure 3.10(b), indicating that this website does not use encryption for data transmission. This is not the certificate's or CA's fault but rather the client or user that is visiting the website. The CA that issued that the certificate (our server) is probably not in the client's local trust store and hence does not recognize certificates issued by this CA as valid [124]. To resolve this issue, the CA should be manually registered in every browser,OS or platform's local trust store, as most of these trust stores have an option to add a third party CA pretty effortlessly. This process could be very frustrating for users though, especially to those not technologically advanced, as they must repeat this process for every each platform that they are going to use. This of course would discourage users of our platform to use our services, hence I found a solution that would be practical and not so time consuming for the user. MKCERT, developed by Filo Scottile [125], is a zero-config tool automating the process of registering the CA in the local trust store therefore the encryption procedure continues unceasingly for every new entity that is introduced to our system.

The first time a user visits our platform and interacts with our dashboard the browser is going to indicate that the connection is not private (Figure 3.10) and that is because, as I said, the certificate cannot be validated yet from a trusted CA. Users could keep using the dashboard and all its functionalities and would still be protected, but studies have shown users fell uncomfortable having that type of error around [18, 19, 124]. While in the dashboard the user should visit the **/download** redirect in order to fetch and download and store a rootCA file which is going to be used with Mkcert. What I did was use Mkcert as CA for signing the certificate meaning that the author is Mkcert and specifically the one on our server. Users should also download Mkcert but not run it yet, instead they have to attach the environmental global parameter *$CAROOT* to the rootCA certificate file. Following this action the users can proceed to installing Mkcert using the command *$ mkcert -install* and get the following messages :

```
$ mkcert -install
Created a new local CA
The local CA is now installed in the system trust store!
The local CA is now installed in the Firefox trust store (requires
browser restart)!
```

After the installation of mkcert the user is ready to send and receive data through encrypted communication. When a user visits the platform the sign indicating that the communication was not secure has been replaced by the lock icon we are used to see on most websites that have implemented HTTPS encryption, as shown in Figure 3.10(a). Although this process is automated for devices as most of them are autonomous and do not rely on user interaction, the same principle could also be applied to users if they accept to execute the package that automates this whole process. To have a better understanding of the HTTPS and CA flow for implementing the self-signed certificate check the diagram in Figure 3.11.

3.2.2 Alternatives

The HTTP has been a web standard regarding the communication protocols, although due to the evolution of IoT in many technological aspects, these usually being devices of small computational power and time essence (e.g sensors, actuators) there was a need for lightweight and more responsive mechanisms, therefore new protocols were invented, namely, CoAP and MQTT which are both request-response protocols. In 2.4 there is a full analysis of various IoT security protocols. Furthermore, Chapter 2.3 examines the various encryption algorithms that are available in both symmetric and asymmetric encryption, which can be used for the production of public and private keys.

3.3 Enhancing security of the infrastructure

Communication between server and client/device has been established and they can now proceed to transmit valuable data to our system, the problem is that cybercriminal attacks could happen in any department of our network. Of course since we are working on a private 5G network and our platform set inside Docker, our system is already secure to a certain point, however I wanted to take security in our server a step further and attach the encryption mechanism that I just described inside our private network, as it is very important to secure every aspect of our system [58]. In the next few sections I am going to analyze how this mechanism is applied to the components and what security implementations they already use to prevent security attacks and data leaks.

3.3.1 Docker and NGINX Security

One of the most important elements of our system is the virtual network that Docker provides, together with NGINX which is in charge of distributing the workload to machines or servers and the redirection of communication to all the components inside our network. These frameworks both include a instructions to safe use of the applications, which with the correct tweaking could build up the security even more.

Docker implements a set of rules which an administrator should follow to keep the system safe and the

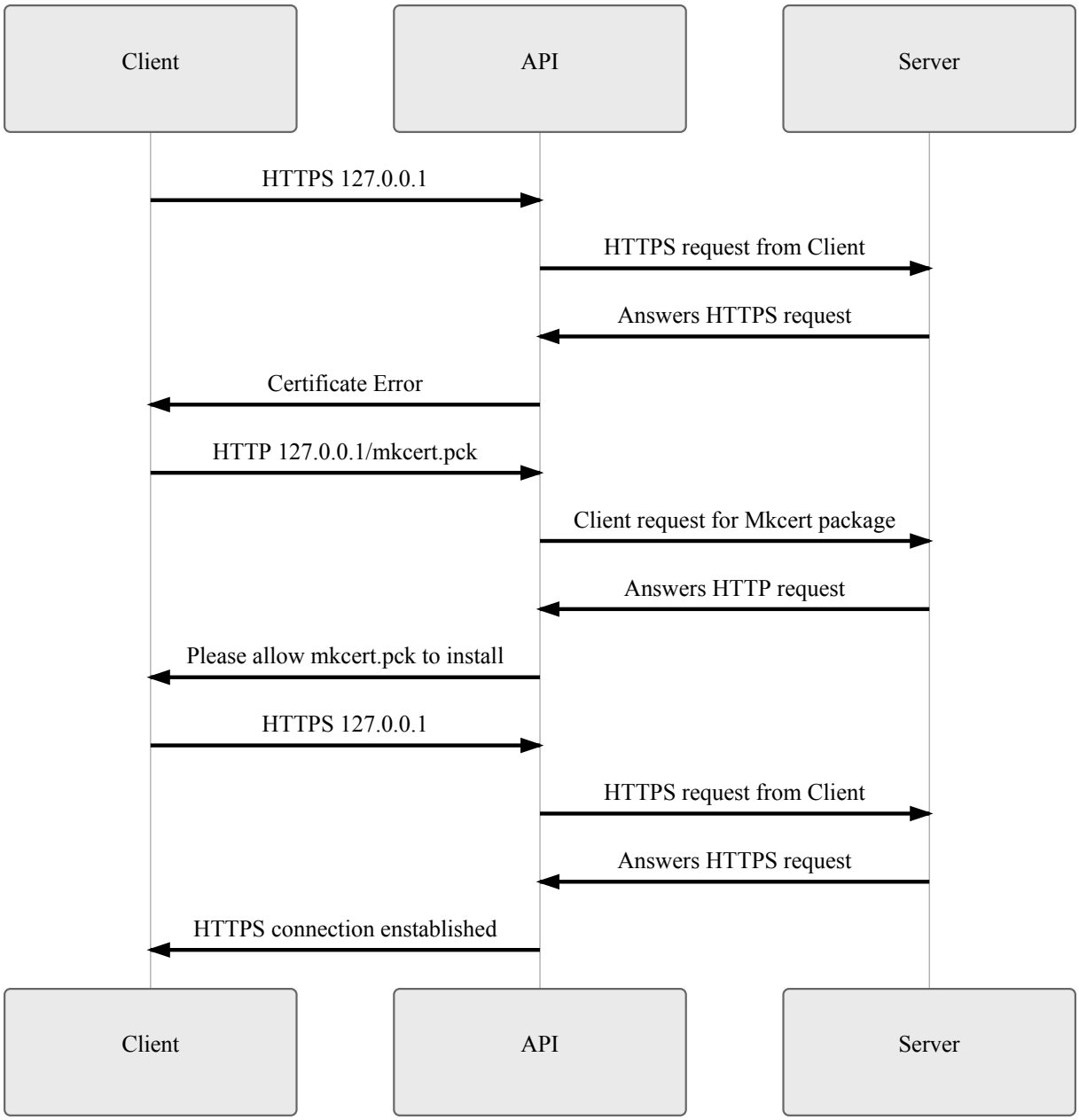


Figure 3.11: HTTPS Establishment

security updated. The first and foremost is keeping the host and Docker up to date, which will prevent container escape vulnerabilities and usually offer the intruder root/administrator privileges. This happens due to the containers sharing the kernel with the host, which means that any exploits that occur inside the container will hit the host kernel, similar to Dirty COW attacks [126] that was developed for Linux kernel exploitation, but has had occurrences in Docker too [90]. Another important rule is to set security policies to the system, where you can also apply Linux security modules like SELinux, AppArmor and seccomp [127]. Upon initializing a Docker network, the nodes or containers are configured for best performance and no restrictions applied for the administrator's convenience therefore letting the user decide for the container's capabilities. This however can lead to security risks and must be changed, starting with the capabilities the container has. By configuring the capabilities or certain privileges of a container that are not essential to the current container, the system can prevent attacks

like the escalation of privileges exploit, by removing the privileges escalation option and is set by the `--security-opt=no-new-privileges` directive. Another option is to disable the inter-container communication which by default is enabled, as not all of the containers are required be linked with in the network, reducing the extensiveness of the penetration attack. Lastly, a good practise is to limit the hardware resources to Docker and set the maximum number of restarts and processes therefore the server will not be prone to DoS attacks.

It is very important to keep certain data secret and secured inside our Docker network such as the certificate and key files that are stored inside our Certificate Manager container (Figure 3.1) and make sure they are not compromised at any time. Docker makes sure that sensitive data such as the TLS certificates/keys, user credentials and SSH keys stay well hidden and safe even during transportation to other containers through their service called Docker secrets. This service provided by Docker uses these secrets to encrypt data during transmission while the secrets are at rest in a Docker swarm [89]. A given secret is only accessible to the services that have been granted access to the certain secret, only while those service are running. Docker can of course be configured to encrypt its containers and images via various mechanisms, with layer encryption being one among them [128], but since our system is based on a server like NGINX, we can configure the security policies inside our server so we don't need to opt in for confusing and time consuming configurations regarding Docker.

NGINX also has its unique defending mechanisms which of course can be configured so the security can be hardened. I have prepared a list of some security configurations that our servers uses and some recommendations for a better experience using NGINX's security modules :

- **Disabling Unwanted NGINX Modules** : When installed, NGINX automatically incorporates many modules just like Docker, which are not always necessary. Disabling the non essential modules can save work load and minimize the possibilities of future attacks due to the limited operations an intruder can perform.
- **SSL and Cipher Suites Configuration** : The TLS protocol has had numerous versions with the latest being TLSv1.3, although older versions can still be used and could be included by default as supported SSL/TLS protocols, which would eventually lead to attacks such as the Beast,Virtual Host Confustion and Certificate related attacks, just to name a few [54]. These kind of attacks have been patched and are no longer considered a threat in latest TLS models therefore it is very important to set the *ssl_protocol* directive in NGINX to use up-to-date TLS protocols. The same application could be used for Cipher Suites too, so the administrator is sure that no vulnerable suites are supported which could lead to data leaks [129], using the *ssl_ciphers* directive and set *ssl_prefer_server_ciphers* directive to on.
- **CSP and X-XSS-Protection** : Setting Content Security Policy (CSP) to the system, can prevent attacks such as data injection and XSS attacks. To enable CSP the *Content-Security-Policy* header should be applied.
- **Include Security Headers** : Just like the CSP header, other security related headers could be applied to enhance security in the system. One of the options is to add the *X-Frame-Options* HTTP header which guides the browser whether to render a page as a <*frame*> or <*iframe*>. This header is very important for clickjacking attacks deterrence. An extra option is to use HTTP Strict Transport Security (HSTS)
- **Monitor Access and Error Logs** : By monitoring the NGINX log files, the administrator can analyze and understand the requests that have been made to the server, including any potential attempt from attacks. The *error_log* directive can be configured to register any desirable types of logs the admin wants to montior. There are plenty of management tools that can organise these log files, one option being the NGINX PLUS version which also provides real-time metrics activity.
- **ModSecurity for NGINX** : There are plenty of Web Application Firewalll (WAF) solutions which can be used with NGINX, but I found out that ModSecurity, an open-source module, worked the

best for our system. The ModSecurity module acts like an application firewall and includes functionalities such as server identity masking, filtering and null-byte attack prevention while performing real-time traffic monitoring.

- **Disable Unwanted HTTP methods** : A suggested policy is to deactivate or remove any unused HTTP methods, that are not going to be utilized by any module. For instance our the server does not use the TRACE or OPTIONS methods so they are not included as a service. This also lowers the possibilities and options for an attacker to penetrate our system.
- **NGINX server tokens** : The server tokens display the version of NGINX the system is using in HTTP responses from the Server header but also when an error page is loaded. To prevent this information disclosure from leaking to unauthorized users the *server_tokens* directive should be set off.
- **Resources and Limits** : Just like Docker, controlling the resources given to a system can lower the chances of DoS attacks. There are several directives which can be configured to set limits to the resources such as *client_body_buffer_size* and *client_header_buffer_size*, aside other CPU and RAM configurations.
- **Update the Server** : As in any other module or application it is very important to keep the version of NGINX up to date, as many security implementations are added with each update which takes measures against potential exploits that the administrator might be unaware of.
- **Automate the process** : The last recommendation is to use third-party applications to automate some of the work. I know this can be a lot to keep in mind and sometimes not knowing the exact modules that the system needs could lead to major repercussions. There are various applications like Gixy [130] or DigitalOcean [131] which help with the process of protecting the NGINX system.

Of course what we needed was to implement the custom certificate and key I had created so they can be composed to work with NGINX. Every server that is created through NGINX has certain security modules which can be configured and accordingly to the administrator's needs. A HTTPS configuration for our server would look similar to this :

```
server {
    listen              443 ssl;
    server_name         www.dashboard.com;
    ssl_certificate     server.crt;
    ssl_certificate_key server.key;
    ssl_protocols       TLSv1.2 TLSv1.3;
    ssl_ciphers         HIGH:!aNULL:!MD5:+SSLv2;
    ...}
```

Initially, the listen socket should contain the ssl parameter besides the port it is listening to, to indicate that this server will be running on the HTTPS protocol. In older versions of NGINX a SSL directive was used to set HTTPS servers, although this did prohibit the use of HTTP servers and thus has been replaced by this socket which is optional. Then we add the address which this server configuration is going to serve, which could include a server name or an internal IP within the network that indicates the location of the server. The next four sockets are referred to the SSL options that are related to the security of the server hence this is where the certificate and key files are added, which are the ones that our server has issued. The *ssl_protocols* and *ssl_ciphers* sockets is where we configure the TSL version and Cipher Suites that were mentioned in the list above about the security modules of NGINX.

3.3.2 Dashboard Security

The Dashboard is the central hub and the only part of the infrastructure that is visible to the users through the web page that visualizes it. Every user interaction is processed through the dashboard first which later communicates with the rest of the components, meaning it can be very susceptible to attempts made by attackers that try to penetrate our system. So to prevent vulnerabilities like clickjacking, cross-site leak and XSS vulnerabilities there are certain restriction that should be applied in order to avoid those kind of attacks :

- **Inline Scripts**
  Restricting a page from executing inline scripts, the system cannot be penetrated by injecting attacks.
- **Unsafe JavaScript**
  Prevent text-to-Java functions like *eval* in the page.
- **Remote Scripts**
  Similar to inline scripts, restricting the page from loading remote scripts from unauthorized servers can prevent injection attacks.
- **Objects**
  To avoid malicious Java/flash or other legacy executable attacks to the system, it is wise to restrict the HTML *object* tag
- **Form submissions**
  Restring the location of the saved data produced by the submission forms can hinder phishing forms injection.

Of course besides these rules the dashboard also has authorization and authentication mechanisms attached, so any unwanted visitors are filtered and kept at bay. The main purpose of the dashboard in the encryption scheme is to serve the rootCA file that is mandatory for the user-to-infrastructure encryption.

3.3.3 Database Security

MongoDB is designed to work with modern application requirements even when it comes to security. This database client has everything to ensure security best practices, such as encryption and authentication modules which are already being used by the rest of the infrastructure. After the user has been authorized and authenticated by the Identity Manager and oAuth2 Provider (Figure 3.1) the data is not sent directly to the user but rather travel through other compartments which also apply their own security to their data as shown in Figure 3.12. MongoDB supports TLS/SSL encryption using both CA signed and self-signed certificates and can be directly implemented to database data through the middleware, which will protect the network traffic and ensure that data is read only from the end the messages were forwarded to. Our system makes sure that data received and sent by the database travel without security risks and efficiency loss, as security overhead tends to increase the overall RTT usually [132, 133].

3.3.4 APIs and Rest of the System

The rest subsystems of the infrastructure also implement the same security modules as the ones I just analyzed, as they also have to process and transfer data and are vital for the the whole data flow. The Middleware API (Figure 3.1) is the first component of the infrastructure that data is fetched to from the devices, this means that protection is a key element as data arrive from outside our network and are

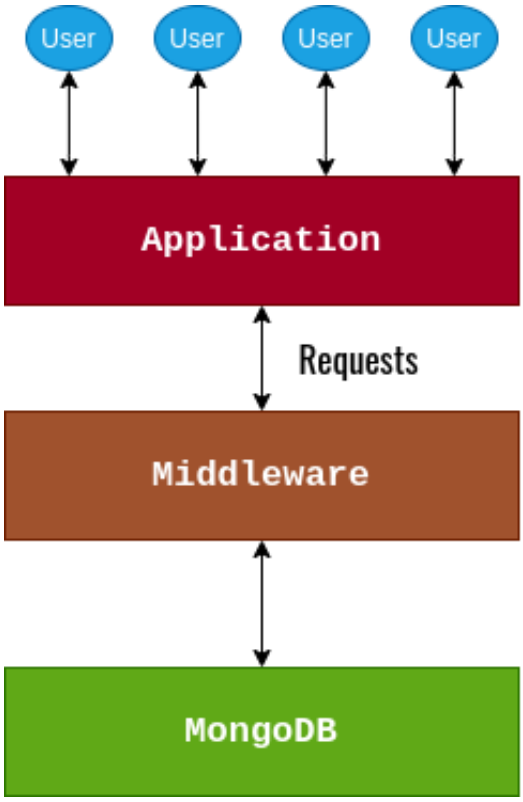


Figure 3.12: Data transmission layers of MongoDB

prone to attacks. The API should constantly check if the device/sensor is authorized and authenticated for certain actions using the token model schema and make sure no activities slip through before the device is examined. Docker and NGINX also make sure that data is protected and secured through their own security modules throughout the infrastructure, which were described earlier in this chapter.

3.3.5 Cohesion with other security elements

One important step for the defense of our system was to ensure that all of the security modules would work together without prompting any issues in the system. Encryption is an important part of the defensing process regardless of other modules a system might have, whether that's authentication, auditing or data integrity mechanisms, as it protects data being leaked outside the private network and makes it more difficult for attackers to compromise data, meaning that potential intruders would have to take extreme alternatives to manage and get into our system. The other security mechanisms are of the same essence, such as the authority and authentication described in Chapter 3.1, so we made sure that a user would be completely isolated, authenticated and authorized to receive and sent data that is private while ensuring quick and reliable feedback from the system. In the next Section I will be analyzing and presenting how these security modules affect the efficiency of the system and what were the solutions to achieve the desirable QoS [134].

Chapter 4: Performance Evaluation & Results

The goal of the project was to develop a lightweight security mechanism that would be able to protect the system with modern solutions that are applicable for IoT cloud based systems using 5G, without hindering the system's performance. This means that we first had to think of how data is going to flow through the system so it is not overloaded by security overhead but be protected at the same time. Once we found the best practice a set of experiments were conducted to get results, but first we have to explain the technological aspects of our experiments, their nature and if these results meet our expectations.

4.1 5G MEC Testbed

For the evaluation of the proposed IoT scheme we used a set of modern hardware tools which the University of Bristol provided us for the purpose of these experiments (Figure 4.1). Moreover, there were certain software applications that were needed for the framework to be completed, so we also designed and deployed a device emulator to serve the purpose of IoT devices and sensors, when actual devices were not available.

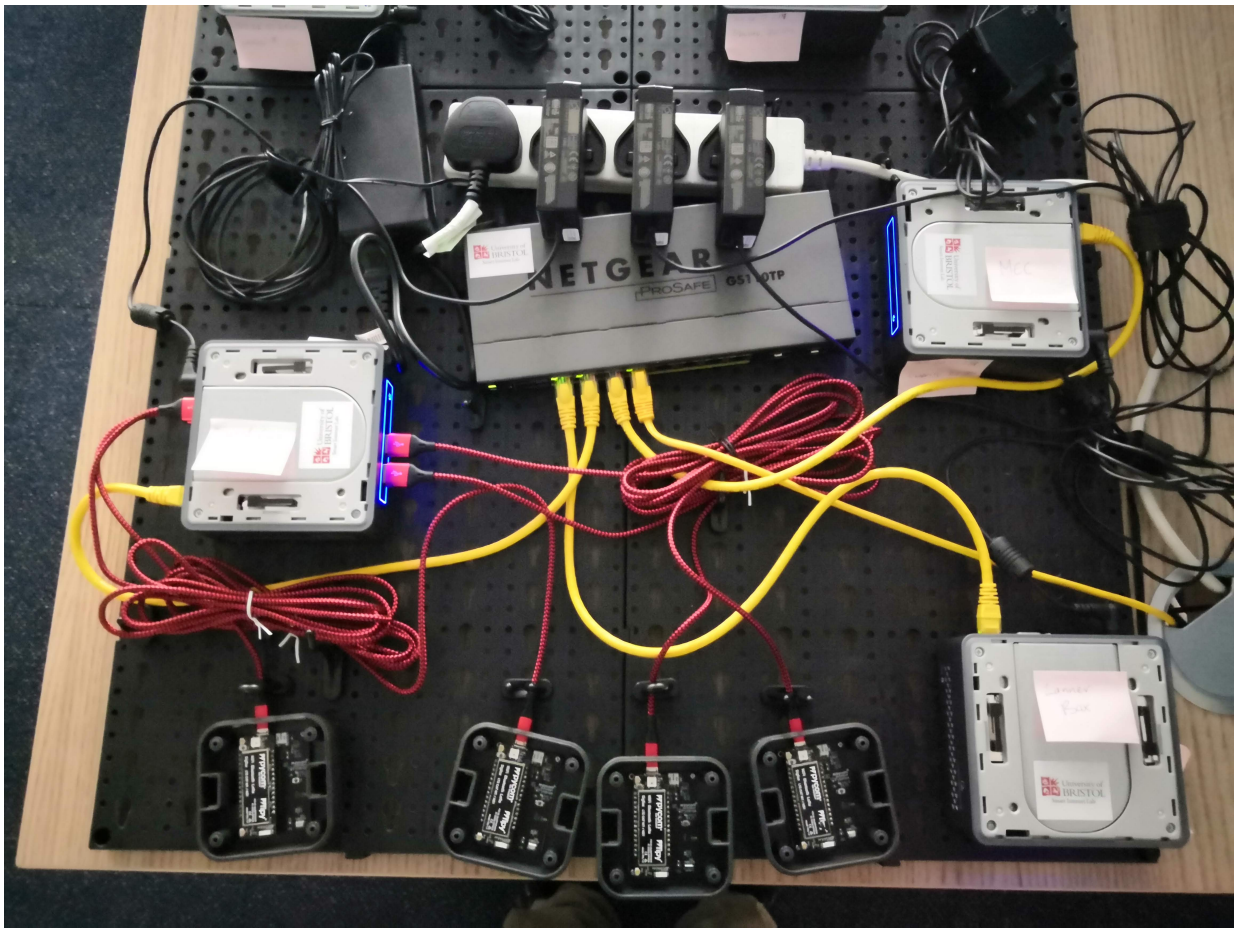
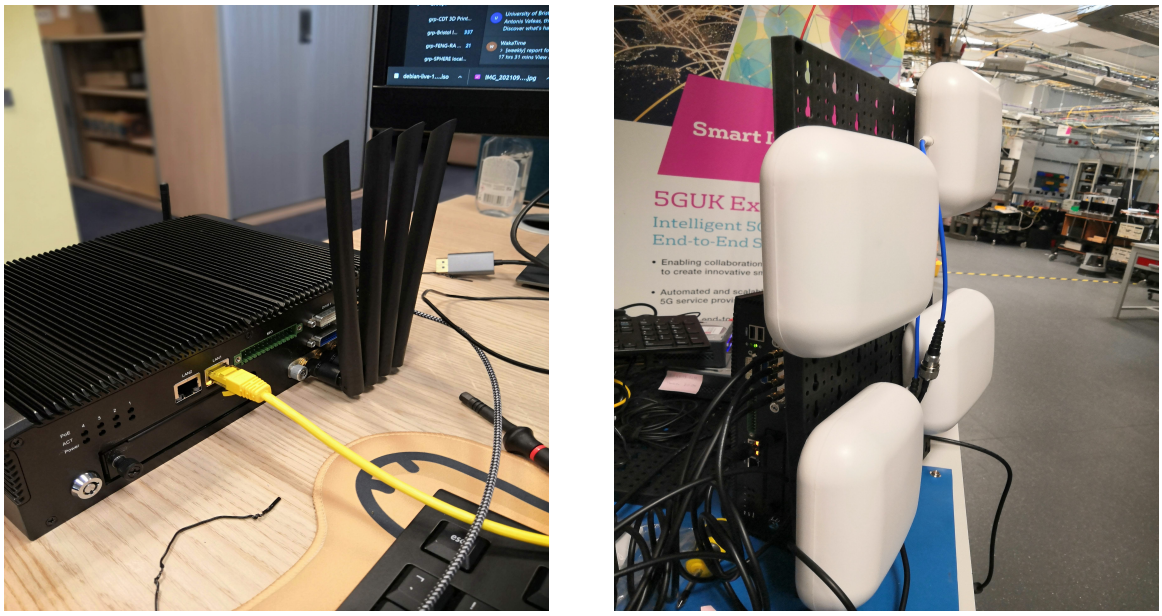


Figure 4.1: Hardware equipment that was used



(a) Indoors router

(b) Outdoors router

Figure 4.2: Left (a) routers used for indoors purposes. Right (b) routers/antennas used for outdoor data transmission

4.1.1 Hardware

As explained in Chapter 3.1.10, first we needed some IoT devices which we would be collecting data from, similar to those in Figure 3.5. These devices of course will be transmitting data through a 5G antenna (Figure 3.6) used for more commercialise and wide area coverage, which our experiments were not intended to do, or through a 5G router system shown in Figure 4.2, since our experiments were conducted inside a restricted zone representing the private 5G network that would be used in an industrial environment.

For the execution of the experiments, two laboratory computers/servers were used with big computational power and plenty of RAM storage so we wouldn't run into any hardware issues while performing the experiments. These two computers could either be working one at a time or in parallel, depending on the type of experiment we are trying to conduct and whether load balancing is utilized [2]. The device emulator module is also a part of our Testbed and is essential for the execution of the experiments, but due to its complexity I separated it into another subsection.

4.2 Device Emulator & Python Threading

In our experiments we need to analyze any possible situation a device can go through and/or stress test our system to check the correspondence at certain levels. A simple sensor is limited to the surrounding environment and cannot be tested for extreme conditions, such as reaching temperature higher than 120 degrees Celsius without specialized materials , before the electronics start failing [135], thus using real devices can sometime be troublesome and may not serve the purposes of tests like these. Another limitation of using real IoT devices is the inflexibility of certain sensors such as RFIDs [136]. To sort this problem out, we developed a Python based applications which simulates the actions of a normal

IoT device, sending hypothetical real time metrics like a sensor would do and forwarding that data over encrypted 5G communication channels. In addition, the emulator can be configured to meet the experiment's requirements and offers features that are hard to be implemented on a real device.

4.2.1 Emulator Python Threading

The device emulator is based on Python Threading [137], which is a python module for parallel execution of python code or functions via procedures called threads. Before explaining how it is applied to our application, let's analyze how threading works and what are it's main characteristics. You might be familiar with the term processing but what you most likely didn't know it is connected to threading . A process is an instance of a computer program that is still running. Each process has its own memory space for storing instructions that are being executed, along any data it needs to store and access to execute. While a thread is a component of a process that can run in parallel with other threads. There can be multiple threads in a process, and they share the same memory space, that is, the memory space of the main process. This means that the code to be executed and all variables declared in the program will be shared by all threads. Running multiple tasks at the same time requires a non-standard Python implementation, writing code in different languages, or using multiprocessing, which will bring in some additional overhead. Due to the way Python's CPython implementation works, threads may not be able to speed up all tasks [138]. This is because interaction with the Global Interpret Lock (GIL) essentially limits the ability to run one Python thread at a time. Tasks that spend a lot of time waiting for external events are often good candidates for threads. Problems that require a lot of CPU calculations and spend little time waiting for external events may not run faster. This applies to code written in Python and running on a standard CPython implementation. If threads are written in C, they can release the GIL and run at the same time. Moreover, each python implementation handles threads in a different manner, so there might be a way of restricting GIL in those. Designing the application to use threads can improve clarity and helps into making the program more concise and easier to reason about. Each device created from our emulator corresponds to a thread and on the next subsection the main components of the emulator are analyzed where python threading is utilized through the devices.

4.2.2 Main Components & Functionalities

The emulator is set into two main classes, the device and supervisor class, which are in charge of creating those devices and setting the emulator to work. The supervisor class carries the functionality of creating a device instance with all the required information about the device. This information is fetched from a pre-configured JSON file which we can alternate before the procedures start, but not while the supervisor is running. The configuration file contains the following information :

- **Users** : The number of users that will be created during the emulation.
- **Devices** : The number of devices that correspond to each user.
- **Prefix** : The name used for the devices followed by a number. The name and number set is unique in the list of devices owned by a certain user.
- **URL** : The relative or full URL path which the devices will be communicating with.
- **Duration** : Duration of the simulation process. The emulator can be set to run for only a few seconds and all the way up to years. This is a good practise to monitor how our system corresponds to different situations of workload.

- **TLS** : Indicates whether TLS is enabled or not.
- **Delay** : The frequency that the devices sent data back to the infrastructure. The number is not stable, as a random number is added to the Delay sourced from a Poisson Distribution algorithm so we can randomize the frequency of each device's transmission.

In Section 3.2.1 I explained the procedure of establishing HTTPS connection successfully between an end and the system. In this occasion the end is the device and in order to establish a secure connection with our infrastructure the rootCA file is needed (Figure 3.11) which is provided by the Certificate Authority. The supervisor is also in charge of retrieving that file so it can be established in each device so the transferred data can be encrypted from the certificate and key that I had created. A part of the supervisor's code can be found in B.1 appendix section, which includes its main functions.

Before using the supervisor, we have created three scripts that are mandatory and crucial for the process of the emulator, which basically serve the emulated user's data that the supervisor is going to use in order to reclaim the infrastructures functionalities, such as authenticating. All of the scripts are written in Python and are based on the Ory Kratos SDK [139] where we have included parts of our system, such as the database or data collector middleware. These three scripts, named *delete_user.py*, *login&access_token.py* and *register_users.py*, have to be executed in a certain order for the emulator to work as depicted in Figure 4.3. The registe_user script creates the users that are needed for the simulation. We have created a list of random names and surnames in a JSON file which are selected and used as the personal information and credential of that user ,such as email, username and password. In appendix B.3.1 you can check the code structure for the whole procedure. The details of the user are then exported to another JSON file named *registeredUsers* that is going to serve the other scripts. Next the login&access_token script imports the information of each user written in the registeredUsers file and completes a login procedure that is required for the user to be authorized to use our application. The script sends HTTPS requests to the correlated authority in order to retrieve essential security fragments, such as the oAuth2 token. Part of the code is displayed in B.3.2 in the appendix section. The information of the user are then stored in the database along the access_token so the user can be authenticated. Although it is lastly referred, the delete_user script is used at the beginning of the procedure and its purpose is to delete any previous record of users that may have been registered in our system. The script scans the MongoDB registries, utilizing the *names.json* file so the name and surname can be used as search parameters in order to delete records with those credentials and make room for new users when the next scripts are used. The code for this script is also included in appendix section B.3.3. The delete_users script does also removes any device registry and the data that these devices have produced so new metrics from newly created devices can be saved.

The last component of the Emulator is the Device class, where the Devices(Threads) are spawned. Its main function is to imitate a real sensor's actions, which would be gathering metrics such as temperature, humidity, weight, pressure, etc. These devices operate accordingly to the parameters that the Supervisor has set to them, like frequency of the transmission and the time they are enabled. If the TLS flag is also set, the devices or threads will establish an encrypted channel which remains active as long as the thread is active, reducing the repetition of newly established connections as I will explain in Section 4.3.4. Part of the code for this class is included in B.2 of appendixes.

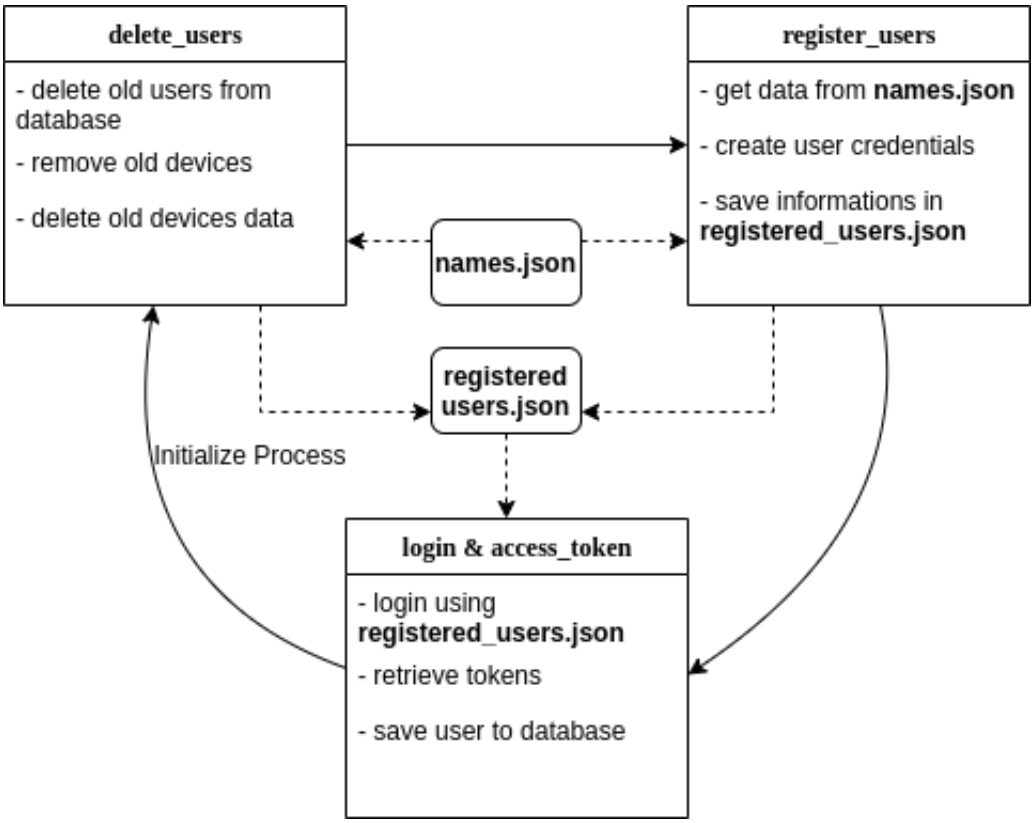


Figure 4.3: Scripts sequence

4.2.3 The infrastructure

The main framework was introduced in chapter 3.1 where each component is analyzed. Since the whole experimentation process is based on the performance of these components and the way they process data, it is a vital key to the whole IoT structure. The benefit of working in cloud based systems is that you don't need to worry about the working environment and how the application is going to migrate to other machines or developers. Our project has been build this way that no optimization is needed when mounted, meaning that it is fully scalable with any software or hardware environments and is ready to be deployed. Another feature of our system is that it's based on a hardened network, this way a future developer can add modification and components without exposing the system [140]. When deployed the system appears seemingly competent and yet simple to the eyes of a the public as they only need to retrieve one package through the terminal, but the thing is it much more than that. The system is based on a complex structure which includes the use of numerous applications collaborating together to achieve the best results. The market's diversity in software and hardware could lead to major compatibility issues had our system been build on an IoT based platform which offers this kind of interoperability and extensibility to our system. The framework is a result of combining the benefits of what modern technological solutions have to offer and deploying it on a cloud based system as seen in Figure 4.4. Everything is included in the host machine which represents a server. The work space that the project was developed is mostly within VSCode where most of our scripts, docker files, log files and the rest of the components are. As explained everything is then containerised and processed through docker which is handled like a private virtual network, including internal IP. The external IP is configurable and can be easily set by using the global variable within NGINX, so the internal IPs are never exposed or used directly in the browser.

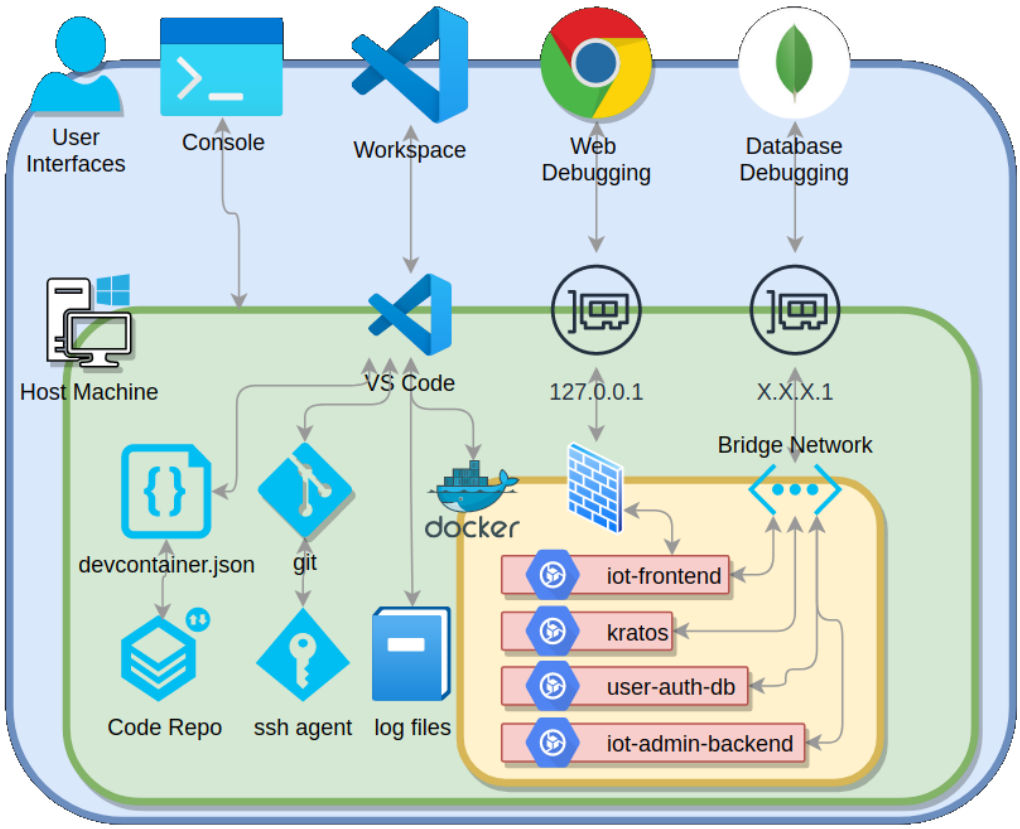


Figure 4.4: Base of the infrastructure

4.3 Experimentation Parameters & Configuration

4.3.1 Configuration

Another feature of the emulator is to write a log of all the messages send by each device together with information including status_code [141],event message, time stamp and most importantly the RTT of a request, an option which we had configured in the Middleware API. This information extracted by the device is the information that is needed for the experiment. As soon as the framework was ready for testing I immediately fired the emulator and was ready to get some results. We had three options or sets to run these experiments,namely a VLAN,Virtual Switch and Physical Switch, each settled in a different environment and configuration so we could get the best results. For VLAN and Virtual Switch experiments, I wanted to establish a baseline of the expected RTT. This makes sense for a smaller number of devices per users, because the presence of an emulator on the same machine will not affect the performance of the system. To understand the large-scale impact of our framework, I am testing the long-term KPI service and the large number of device counts, so that I can understand the deterministic overhead introduced. This compute overhead also becomes the uptime of the load balancing service, which can be managed by orchestrating NFV. Therefore, the NFV resource allocation, more specifically the network splitting and load balancing can reduce overhead to meet specific application needs. The first two configurations are the same, the only difference is the virtual network between the device simulator and the data collector. Unlike VLAN, Virtual Switch uses the Docker network schema. One of the disadvantages of the Docker network structure is the use of VXLANs, Layer 2 virtual bridges, and other network overlays like

Dzogovic et. al [142] explain in their article. The effect of this drawback can be clearly seen in Table 4.1, where the average RTT of test B is significantly higher than that of test A. Finally, tests C and D are run on a Physical Switch configuration that uses a 1 Gbit paravirtualized network adapter for simulation, which makes the two servers appear as if they are separated from each other. As you can see in Figure 4.5 the physical switch had the best performance out of the three environments when it comes to RTT. Thus the next experiments were conducted based on the Physical Switch environment setup on which, as you later will see, we added some optimization options in order to most out of the configuration.

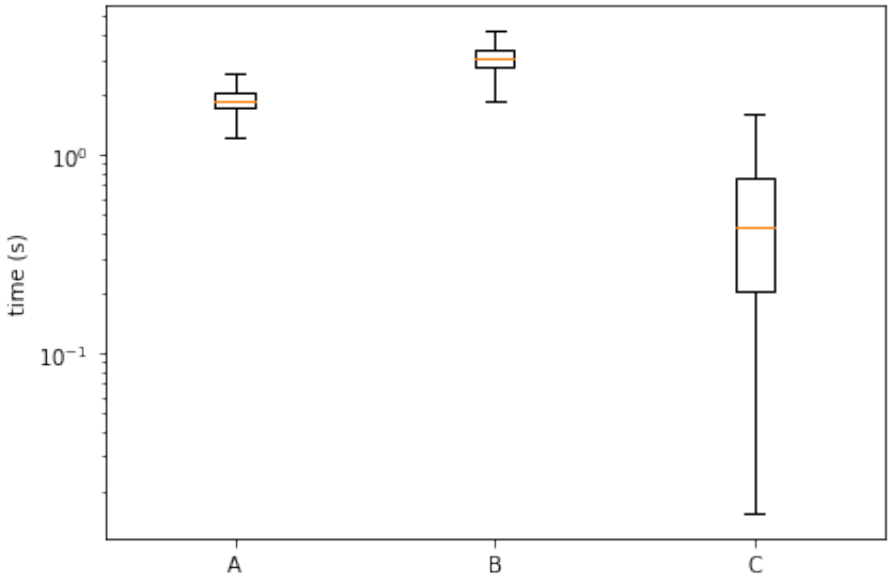


Figure 4.5: Comparison of the RTT in each environmental setup : (A) VLAN, (B) Virtual Switch, (C) Physical Switch

The performance is measured through packets per second (Ps or pps) and its equation is listed below :

$$P_s = P_f \cdot D_u \cdot U \tag{1}$$

where:

D_u is Number of Devices Per user

U Number of Users

P_f Packet frequency $P_x \hookrightarrow \frac{e^{-\lambda}\lambda^x}{x!}$

P_s Packets per second

4.3.2 Optimization

Once the preparation of the experiment environment was established I wasn't quite sure of the results I was expecting. In the first few runs as I was increasing the number of devices per user I noticed that the RTTs kept getting higher in an abnormal matter, which raised suspicions as to whether the procedure

Table 4.1: Table with experiment parameters

	TEST A	TEST B	TEST C	TEST D
API Config	VLAN	Virtual Switch	Physical Switch	Physical Switch
No. of Users	10	10	10	10
Devices per User	50	50	50	50
Authorization	Enabled	Enabled	Enabled	Enabled
Data Rate per Device	Poisson Distribu-tion [6ppm - 24ppm]		~0.2pps	~0.2pps
Test Duration	10 mins	10 mins	10 mins	16 hours
RTT Average w/out TLS	**1,868 sec**	**2,164 sec**	**0,434 sec**	**0,294 sec**
RTT Average with TLS	**0,021 sec**	**3,040 sec**	**0,042 sec**	**0,028 sec**

was executing swimmingly or not. There was noticeable drift of the RTT time in the graph, when we got our results back that wasn't expected, if you check Figure 4.6. That's where our team decided to look further into the data transmission schema to check if there was room for any more optimization to our system. What we found was a tool called Remote Dictionary Server or Redis for short, which is an open source, in-memory data structure store and is as a cache,database,memory broker and queue [143]. Redis provides a response time of less than milliseconds, supports millions of requests per second, and is suitable for real-time applications in industries such as gaming, ad tech, financial services, healthcare, and the IoT.

Today Redis is one of the most popular open source engines and has been rated the "most popular" database by Stack Overflow for five years in a row [144]. Due to its fast performance, Redis is a popular choice for caching, session management, games, leaderboards, real-time analytics, geospatial, travel, chat/messaging, streaming media, and publish/subscribe applications. There is a number of reasons why one would use Redis [145], so I listed some of them to get the idea :

Smart Cache : The real value of Redis is to provide application developers with advanced data structures and operations. "Smart caching" is not just about using these data structures through GET and SET operations, it is the behavior of using their unique characteristics to manipulate data efficiently and optimally. Although a complete overview of Redis' extensive feature is beyond the scope of this thesis, the two main examples of how it enables smart caching are its commands for modifying data on the server and its ability to run Lua scripts.

High availability : The cache is only effective when the application is available. By using Redis's built-in replication mechanism, the data in Redis and the service provided by Redis can be highly available. A Redis cache can be configured with a replica (that is, slave), which will continuously update as the main

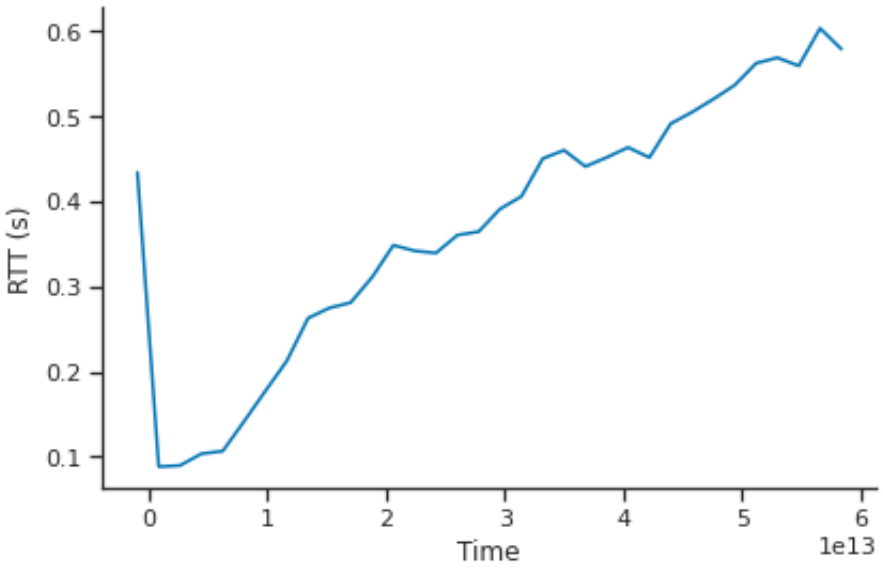


Figure 4.6: RTT escalation without Redis

(that is, master) is modified. To ensure cache availability, management and watchdog processes are used to monitor the cache process, and manages failover when needed.

Data expiration : The keys in Redis can be configured with a time to live (TTL), after which they will expire. Before they expire, these keys are called ”volatile” keys. The TTL of a volatile key is not affected by changes in the data contained in the key, and the TTL can be updated independently. After expires, Redis will automatically delete expired keys without any further action by the application. Expiration is a key concept in the cache, it has two important purposes: First, it is a simple but extremely effective cache invalidation method. Secondly, expiration is an indispensable tool to control the size of the cache, so it is generally recommended to assign an expiration date of to all cached data.

Optimized for speed : Redis is designed and implemented for performance. Because it is written in ANSI C, it compiles into extremely efficient machine code with very little overhead. It uses (primarily) a single threaded event loop model to make optimal use of the CPU cores it runs on. The data structure used internally by Redis is implemented to maximize performance. Most data operations require constant time and space.

Support any data : The data stored in Redis can be of any shape and size. Redis is binary safe, so it can store anything from human readable text to hard-coded binary files. The size of a single data item in Redis can be, from 0 bytes to 0.5 GB, allowing you to cache almost any data.

Memory storage : All data in the Redis is stored in RAM, providing the fastest possible data access time for read and write requests.

Multi-key operation and key-based access Redis is based on a key-value model, in which data is stored and retrieved from Redis through keys. Key-based access allows extremely efficient access time, and the model is naturally cached. Redis provides idiomatic GET and SET semantics for interacting with data while it also has several commands to operate multiple keys. Compared to performing operations one after the other, multi-key operations provide better overall performance because they require much less communication and management.

After applying the Redis module to our system we had some radical changes to the performance of our

system. As you can see in Figure 4.7 the second experiment (B) has a faster RTT time (Y scale). Imagine the performance boost of Redis when combined with other technologies such Docker images based on Alpine [146] and executing those on much more efficient hardware equipment such as future DRR5-6400 RAM memory [147].

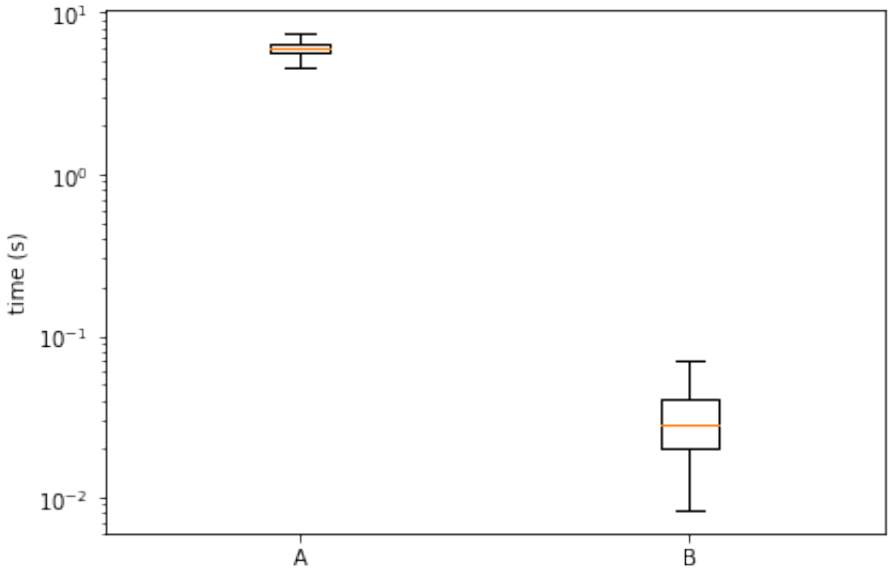


Figure 4.7: Experiment A is without optimization. Experiment B has Redis implemented

4.3.3 Impact of Encryption

The next step is to measure the level of impact that the encryption overhead has over TCP/IP packets and how the RTT is affected. When securing data there are certain procedures and additional information that need to be added to the packet such as encryption and hash padding,IPSec headers and overhead calculations. Therefore it is not only the encryption mechanism that increases the total size of the data but rather a combination of many security overheads which are required and essential for safe data transmission. The security applied to a system might become a major factor of performance degradation and can account for over 80% of traffic increment, if the security modules are not well modified to suit the system capabilities [148]. A quick way to check encryption estimation timing informations is by using OpenSSL's *speed* command. For reference, in Figure 4.8 you can observe the estimated number of 2048bit RSA packets send in a period of 10 seconds. However, there are certain configurations which be applied in our security modules in order to optimise the transportation rates, one of which is SSL Percistancy or TLS Resumption. Due to the fact that most of the components in the infrastructure are based and configured to solely work on HTTPS connection, we didn't have any experiments which were conducted with security disabled. Setting the whole framework to work on HTTP rather than HTTPS requires a lot of work and does not always operate normally. For instance, in some occasions I tried setting the security flag to false so the request methods wouldn't require a SSL certificate for communication, which proved to work fine in mere cases, but soon proved to not cooperate with our system as I was constantly getting security concerned errors. Therefore I decided not to proceed with experimentation on a version that wouldn't have implemented encryption mechanisms.

```
clod@clod:~$ openssl speed rsa2048
Doing 2048 bits private rsa's for 10s: 19773 2048 bits private RSA's in 10.00s
Doing 2048 bits public rsa's for 10s: 685170 2048 bits public RSA's in 10.00s
OpenSSL 1.1.1f  31 Mar 2020
built on: Mon Aug 23 17:02:39 2021 UTC
options:bn(64,64) rc4(8x,int) des(int) aes(partial) blowfish(ptr)
compiler: gcc -fPIC -pthread -m64 -Wa,--noexecstack -Wall -Wa,--noexecstack -g -
O2 -fdebug-prefix-map=/build/openssl-JWge0V/openssl-1.1.1f=. -fstack-protector-s
trong -Wformat -Werror=format-security -DOPENSSL_TLS_SECURITY_LEVEL=2 -DOPENSSL_
USE_NODELETE -DL_ENDIAN -DOPENSSL_PIC -DOPENSSL_CPUID_OBJ -DOPENSSL_IA32_SSE2 -D
OPENSSL_BN_ASM_MONT -DOPENSSL_BN_ASM_MONT5 -DOPENSSL_BN_ASM_GF2m -DSHA1_ASM -DSH
A256_ASM -DSHA512_ASM -DKECCAK1600_ASM -DRC4_ASM -DMD5_ASM -DAESNI_ASM -DVPAES_A
SM -DGHASH_ASM -DECP_NISTZ256_ASM -DX25519_ASM -DPOLY1305_ASM -DNDEBUG -Wdate-ti
me -D_FORTIFY_SOURCE=2
                  sign    verify    sign/s verify/s
rsa 2048 bits 0.000506s 0.000015s   1977.3  68517.0
```

Figure 4.8: Estimated RSA performance using OpenSSL speed command

4.3.4 TLS Resumption

TLS is cryptographic protocol which provides end-to-end data security based on a combination of both symmetric and asymmetric cryptography delivering a well optimized conjunction of performance and security therefore fulfilling the needs of security in IoT [97]. TLS is a layered protocol, meaning that the message may include information when processed through each layer such as length, description and content. Upon transmission, data is fragmented into blocks and compressed while encryption is applied along with a MAC address. When communicating with the server, a TLS session is established when a TLS handshake is completed among the server and the client [57], where the TLS version and cipher suite are specified as well as the authentication of the identities through SSL certificates and generating the session keys. The session is stateful, this signifies that the session can transfer data as long as it is alive without the need of repeating the mentioned steps. One easy way of transferring data securely is establishing a new TLS connection in every client request. Even though this is not erroneous, the proliferation of new sessions in a single interaction can amplify the network traffic and RTT due to excess overhead data. To desist from this situation, we established a single TLS session in o-RTT mode that was presented in TLS 1.3 [149] allowing session resumption. This way the need for further TLS handshakes is expunged and the overall latency reduced [150], promising better response times and performance for devices that require low latency, such us IoT sensors, as shown in Figure 4.9. The parameters and exact results of each experiment can be observed in Table 4.1.

4.3.5 Scalability

In an IoT platform the connected devices to a system could vary from only 2 devices a user could have at their home, to thousands of devices distributed to a complex of industrial areas. I had to make sure that the encryption mechanism would comprehend huge numbers of devices the same way it would with only two or three devices. The only issue was that we were limited in terms of hardware power and were able to get results to a certain amount of devices. One thing we were very satisfied with is the performance and durability of the system over time. In the Figure 4.9 and 4.1 you can see that the average RTT span maintains at the same levels throughout the time, from 10 minutes all the way to 16 hours of uninterrupted data transmission.

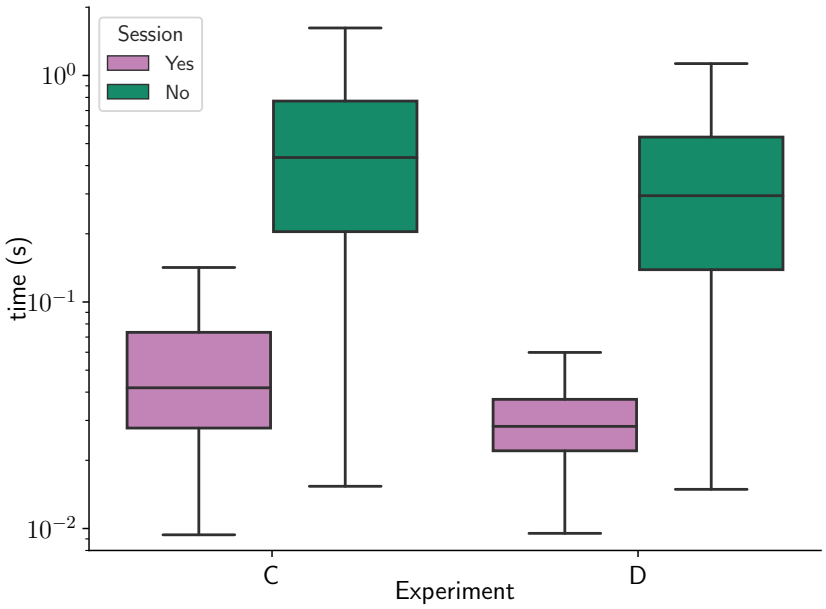


Figure 4.9: RTT using TLS Resumption

4.4 Epilogue

In this chapter I analyzed the experiments that were conducted based on the infrastructure that our team in Smart Internet Labs created. The results vary as certain configurations were tested in order to find the best options that suited the purpose of this whole project. We came up with two main optimisation plans, those being the Redis module and TLS Resumption, which worked exceptionally good in our favor and helped in the enhancement of the overall system and security.

Chapter 5: Conclusions

IoT devices are already a part of our everyday lives in both our homes and workplaces, and statisticians predict that the number of IoT devices will continue to grow in the future based on the present trend. Despite the fact that the IoT paradigm offers amazing features and benefits in a broad variety of sectors such as wearables & smartwatches, smart cities, smart grids, health, and smart farming, this paradigm also has also introduced a number of challenges, particularly in terms of privacy and security. The constrained computing nature of IoT devices often makes it impossible to perform various ciphering algorithms in order to guarantee important security features such as channel protection. Furthermore, because of the tiny size of IoT devices, as well as the cheap cost of their components (or the cost of a single component), large deployments of IoT devices are being integrated into existing infrastructures on a daily basis. Furthermore, aside from the fundamental heterogeneity between the various IoT devices required to perform a variety of functions, IoT deployments are expected to take years to complete, making it quite common for new IoT devices or replacements to run different firmwares in different models from various manufacturers. This heterogeneity has a direct effect on the security of the system. For example, unique behaviours at the time of commissioning make it impossible to guarantee, in a uniform manner, certain security fundamentals from the start. Actually, many devices perform operations from the beginning that the user is not aware of, such as sending information continuously to the manufacturer "only for statistical purposes," which in general also entails privacy concerns, especially when we are talking about sensitive data such as credit card numbers, account passwords or even home security passcodes.

To cope with threats and challenges like these this thesis focuses on development, design and implementation of a secure 5G IoT platform based on the encryption mechanism that I created together with the framework itself. The solution enables the definition and enforcement of proactive and reactive orchestration security paradigms that are based on capabilities and take into account the current knowledge about the infrastructure. To achieve peak robustness and threat resistance a broad variety of security features, including data and communication encryption, authentication, authorization, network administration, IoT management, channel protection, privacy, network slicing and orchestration, database encryption, and others, are provided by both mine and my team's extension. For modelling, refining, and translating such security models, as well as for coordinating and enforcing them throughout the MEC IoT infrastructure, our team developed and deployed the necessary components and processes via various security enablers that were specifically intended for IoT.

Overall, this thesis aimed to research into the encryption mechanisms that has been developed over the years, analyze them, justify their use cases and then apply them into a real 5G testbed. The infrastructure itself included a combination of new and subtle technologies and protocols applied to production now level 5G IoT Cloud system and experiment on top of that. Security of a IoT system does not conclude only to the encryption part but rather focuses on the combination of various different security elements such as authentication and authorization, in order to achieve state-of-art security architecture in our system. The experiments that were conducted in the laboratory are based on real advanced IoT network using 5G spectrum and MEC servers which I wouldn't have the chance to work on without the assistance of the Smart Internet Labs [17]. During the experimentation phase there were attributes still being added to the system so I would gather more information regarding the optimization of an IoT system, therefore

developing this research even further. We do get plenty of details about the impact that encryption has on a hybrid MEC platform from the recorded results, which hopefully can assist the readers of this thesis to have a better understanding of the overall MEC security and encryption algorithms, primarily the end-to-end encryption procedure for 5G IoT platforms.

5.1 Analysis & Contributions

In the first chapters the structure of this thesis was presented together with a broad and full explanatory literature review which studies the principles of security, its meaning and definition as well as the application in IoT system through its many forms, either that is encryption, authentication, authorization or even a simple hash function. Next the various and malicious challenges that threaten general and IoT systems with some of them being as old as the security modules themselves. The weak spot of a system cannot be determined just by one point of view, because as I explained in Section 2.1.1, the system can be penetrated from multiple layers of the system, either that is application, middleware, network or sensor level, each of them is exposed to unique threats depending on the information that the layer holds and the amount of effort and time it takes for an intruder or impostor to get access. Some of the layers are targets of well known attacks with SQL Injection, Man-in-the-Middle and DDoS attacks being among them.

To minimize the risk of getting attacked by all these threats one important security module is encryption. Encryption is the process of protecting data, when they are stored or even in transmission, by converting them to a preposterously written file called ciphertext, which can only be decrypted by entities that have acquired a certain key which decodes that file. However, in order for the interested end to be able to encrypt/decrypt the data the key should be shared with them beforehand. For this reason encryption is divided into two main methods called symmetric and asymmetric key encryption. The first one requires the same set of keys that the source used to encrypt the data, in order for the recipient to decrypt the ciphered data. Hence, the key should be transferred privately to the other end as it could fall victim to an eavesdropping attack therefore the attacker would be able to read all the transferred data. On the second occasion of asymmetric encryption, there are two separate keys used for the same process, one called the private key and the other private key, thus this technique is also called public key encryption. Finally the numerous encryption algorithms are presented. While all of them have the same scope of encrypting data, the result and techniques vary and so do the amount of security they provide to the system. The Rijndael encryption was found to be the best performing variant when competing in the NIST competition and was crowned the AES title. At the end of the literature review I have included some interesting related, to my thesis, work while I also analyze some additional communication protocols that have picked the public's interest in the IoT community.

Since the framework I was working on was developed by me and my team, I had to give a full explanation of the whole infrastructure and the tools I was going to work with. Starting from the platform of the framework, that is also its base, we have created a well thought and tightly connected virtual network which was build with the aim of NGINX and Docker using containers for each of our components (Figure 3.1). The dashboard is the representing model which the users can interact with and get their services from. Next the two APIs, data collection API and middleware API are in charge of communicating with the IoT devices and receive or send HTTP requests, as well as, rerouting the data to the right destination for them to be saved, configured or deleted. The other private endpoints each have a unique job in the

system. The identity manager is in charge of the authorization, the oAuth2 provider for authentication, the certificate manager for encryption and the database is self explanatory. After you get an idea of the infrastructure it is time to analyze the encryption method that supported the security of our system. I worked with the HTTPS protocol which is based on the SSL and I explain in details on Section 2.4 why I went with this option. Although I was a bit sceptic about this approach in the beginning, I decided to get through this research with method and see the results by myself, instead of trying to find answers online. However, I was able to modify the encryption mechanism and key the way I wanted to. So based on what I had studied about encryption algorithms in Section 2.2 I was confident in finding a combination that would be feasible to work on our IoT platform. In Section 3.2.1 I have a full in-depth analysis of how the encryption mechanism works and how I automated the process of key sharing with IoT devices.

Last but not least, are the experiment results. Before jumping to the numbers I had to first introduce the testbed and environment that the experimentation was going to happen. Since I was working on a real 5G based system, it was really thrilling to try new and innovative technologies in person, which many people don't have access to, while also testing the capabilities and be able to compare that to more traditional means. Except the 5G IoT equipment we also created a device emulator which was set to mimic the the device functionalities and from the testing that I did, the emulator could represent a simple IoT device exceptionally good. I tested the performance of our system in three different environments, each having their configuration and capabilities in which I could determine the impact that encryption has on the system. Later I discovered that the TLS protocol has a feature called TLS resumption in which multiple HTTP request can be completed without the need of setting up a new TLS connection for each request, which has a huge impact on the overall performance based on the results we what in Figure 4.3.4.

5.1.1 Recommendations & Future work

At the time I am writing this thesis the development of the infrastructure and the encryption application has nearly been finalized. However, there are still some configuration elements that we want to add in the future and perform further experiments to our system. One of the experiments I am really looking forward to is a penetration testing against the infrastructure. This means that I will be attempting to get inside the framework, as an intruder, and steal valuable information from the system using various techniques, like the attacks that are mentioned in Section 2.1.1 and see how strong the system really is in terms of general security and encryption-wise. Another interesting attempt would be using the DDoS attack and check how well will the middleware, which is responsible for retrieving multiple device requests, cope with that.

Another good practise would be to try other encryption combinations on the system. There were some great ideas that were suggested in other publications while I was gathering information about the literature review, which I am very keen on trying. For instance, I came upon Sing's et al. [6] research, where they propose an encryption method that is very power-friendly and is based on an algorithm they produced themselves called HLA. The last step is then emigrating the whole infrastructure, together with its security modules, to a MQTT or CoAP embedded IoT platform and be able to see the difference they may have compared to the HTTP based platform that I am currently using.

One of my certain goals is to soon publish the international journal that I am currently co-working on

with the team at UOB which is based on the research and development of the secure IoT infrastructure that was also used for the purpose of this thesis.

Bibliography

[1] P. Gokhale, O. Bhat, and S. Bhat, "Introduction to iot," *International Advanced Research Journal in Science, Engineering and Technology*, vol. 5, no. 1, pp. 41–44, 2018.

[2] W.-Z. Zhang, I. A. Elgendy, M. Hammad, A. M. Iliyasu, X. Du, M. Guizani, and A. A. A. El-Latif, "Secure and optimized load balancing for multitier iot and edge-cloud computing systems," *IEEE Internet of Things Journal*, vol. 8, no. 10, pp. 8119–8132, 2021.

[3] H. R. Ghorbani and M. H. Ahmadzadegan, "Security challenges in internet of things: survey," in *2017 IEEE Conference on Wireless Sensors (ICWiSe)*, pp. 1–6, 2017.

[4] M. H. Khan and M. Ali Shah, "Survey on security threats of smartphones in internet of things," in *2016 22nd International Conference on Automation and Computing (ICAC)*, pp. 560–566, 2016.

[5] M. Vajaranta, J. Kannisto, and J. Harju, "Implementation experiences and design challenges for resilient sdn based secure wan overlays," in *2016 11th Asia Joint Conference on Information Security (AsiaJCIS)*, pp. 17–23, 2016.

[6] S. Singh, P. K. Sharma, S. Y. Moon, and J. H. Park, "Advanced lightweight encryption algorithms for iot devices: survey, challenges and solutions," *Journal of Ambient Intelligence and Humanized Computing*, pp. 1–18, 2017.

[7] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of things (iot): A vision, architectural elements, and future directions," *Future generation computer systems*, vol. 29, no. 7, pp. 1645–1660, 2013.

[8] S. Wang, Z. Zhang, Z. Ye, X. Wang, X. Lin, and S. Chen, "Application of environmental internet of things on water quality management of urban scenic river," *International Journal of Sustainable Development & World Ecology*, vol. 20, no. 3, pp. 216–222, 2013.

[9] M. A. Khan and K. Salah, "Iot security: Review, blockchain solutions, and open challenges," *Future generation computer systems*, vol. 82, pp. 395–411, 2018.

[10] M. Cremonini and P. Martini, "Evaluating information security investments from attackers perspective: the return-on-attack (roa)," in *WEIS*, 2005.

[11] Z. Ling, K. Liu, Y. Xu, C. Gao, Y. Jin, C. Zou, X. Fu, and W. Zhao, "Iot security: An end-to-end view and case study," *arXiv preprint arXiv:1805.05853*, 2018.

[12] M. Katagi, S. Moriai, *et al.*, "Lightweight cryptography for the internet of things," *Sony Corporation*, vol. 2008, pp. 7–10, 2008.

[13] M. Ebrahim, S. Khan, and S. S. U. H. Mohani, "Peer-to-peer network simulators: an analytical review," *arXiv preprint arXiv:1405.0400*, 2014.

[14] M. Ebrahim, S. Khan, and U. B. Khalid, "Symmetric algorithm survey: a comparative analysis," *arXiv preprint arXiv:1405.0398*, 2014.

[15] S. Al Salami, J. Baek, K. Salah, and E. Damiani, "Lightweight encryption for smart home," in *2016 11th International conference on availability, reliability and security (ARES)*, pp. 382–388, IEEE, 2016.

[16] "University of bristol official website." `https://www.bristol.ac.uk/`.

[17] "Smart internet labs official website." `http://www.bristol.ac.uk/engineering/research/smart/`.

[18] S. Furnell and N. Clarke, "Power to the people? the evolving recognition of human aspects of security," *computers & security*, vol. 31, no. 8, pp. 983–988, 2012.

[19] L. F. Cranor and S. Garfinkel, *Security and usability: designing secure systems that people can use*. " O'Reilly Media, Inc.", 2005.

[20] M. E. Whitman and H. J. Mattord, *Principles of information security*. Cengage learning, 2011.

[21] G. Disterer, "Iso/iec 27000, 27001 and 27002 for information security management," 2013.

[22] C. Canongia and R. Mandarino, "Cybersecurity: The new challenge of the information society," in *Handbook of Research on Business Social Networking: Organizational, Managerial, and Technological Dimensions*, pp. 165–184, IGI Global, 2012.

[23] N. I. for Cybersecurity Careers and S. D. of Homeland Security, "A glossary of common cybersecurity terminology," 2014.

[24] O. O. U. Press, "Oxford university press : Oxford online dictionary," 2014.

[25] J. A. Lewis, "Cybersecurity and critical infrastructure protection," *Center for Strategic and International Studies*, vol. 9, 2006.

[26] S. SECTOR and O. ITU, "Itu-tx. 1205," *Interfaces*, vol. 10, no. 20-X, p. 49.

[27] G. o. C. Translation Bureau, "Public safety canada : Terminology bulletin 281:emergency management vocabulary," 2014.

[28] C. C. Wood, "Why information security is now multi-disciplinary, multi-departmental, and multi-organizational in nature," *Computer Fraud & Security*, vol. 2004, no. 1, pp. 16–17, 2004.

[29] K. D. Mitnick and W. L. Simon, *The art of deception: Controlling the human element of security*. John Wiley & Sons, 2003.

[30] E. Kimura, T. Norihiko, and K. Ishihara, "The design of virtual private network topology for migrating to ipv6 world," in *2004 International Symposium on Applications and the Internet. Proceedings.*, pp. 313–316, IEEE, 2004.

[31] D. Miorandi, S. Sicari, F. De Pellegrini, and I. Chlamtac, "Internet of things: Vision, applications and research challenges," *Ad hoc networks*, vol. 10, no. 7, pp. 1497–1516, 2012.

[32] R. Roman, J. Zhou, and J. Lopez, "On the features and challenges of security and privacy in distributed internet of things," *Computer Networks*, vol. 57, no. 10, pp. 2266–2279, 2013.

[33] V. Gazis, K. Sasloglou, N. Frangiadakis, and P. Kikiras, "Wireless sensor networking, automation technologies and machine to machine developments on the path to the internet of things," in *2012 16th Panhellenic Conference on Informatics*, pp. 276–282, IEEE, 2012.

[34] R. Wainwright and B. Waites, "The changing face of organised crime: Can europol keep up?," in *Organized Crime, Corruption and Crime Prevention*, pp. 327–334, Springer, 2014.

[35] M.-H. Maras, "Internet of things: security and privacy implications," *International Data Privacy Law*, vol. 5, no. 2, p. 99, 2015.

[36] M. Abomhara and G. M. Køien, "Security and privacy in the internet of things: Current status and open issues," in *2014 international conference on privacy and security in mobile systems (PRISMS)*, pp. 1–8, IEEE, 2014.

[37] N. A. M. YOUSEFI, M. Bagheri, and J. E. NEISHABOURI, "A decision making model for outsourcing of manufacturing activities by anp and dematel under fuzzy environment," 2012.

[38] S. Alam, M. M. Chowdhury, and J. Noll, "Interoperability of security-enabled internet of things," *Wireless Personal Communications*, vol. 61, no. 3, pp. 567–586, 2011.

[39] F. T. Commission *et al.*, "Internet of things: Privacy & security in a connected world," *Washington, DC: Federal Trade Commission*, 2015.

[40] D. Hodges, "Cyber-enabled burglary of smart homes," *Computers & Security*, vol. 110, p. 102418, 2021.

[41] L. Li and A. Haans, "An investigation on the risk perception of residents in iot smart home environments," *Eindhoven University of Technology*, 2017.

[42] V. Hassija, V. Chamola, V. Saxena, D. Jain, P. Goyal, and B. Sikdar, "A survey on iot security: application areas, security threats, and solution architectures," *IEEE Access*, vol. 7, pp. 82721–82743, 2019.

[43] J. Wurm, K. Hoang, O. Arias, A.-R. Sadeghi, and Y. Jin, "Security analysis on consumer and industrial iot devices," in *2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 519–524, IEEE, 2016.

[44] F. Meneghello, M. Calore, D. Zucchetto, M. Polese, and A. Zanella, "Iot: Internet of threats? a survey of practical security vulnerabilities in real iot devices," *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 8182–8201, 2019.

[45] C.-H. Liao, H.-H. Shuai, and L.-C. Wang, "Eavesdropping prevention for heterogeneous internet of things systems," in *2018 15th IEEE Annual Consumer Communications & Networking Conference (CCNC)*, pp. 1–2, IEEE, 2018.

[46] M. Ahmed and A.-S. K. Pathan, "False data injection attack (fdia): an overview and new metrics for fair evaluation of its countermeasure," *Complex Adaptive Systems Modeling*, vol. 8, no. 1, pp. 1–14, 2020.

[47] S. Kumar, S. Sahoo, A. Mahapatra, A. K. Swain, and K. K. Mahapatra, "Security enhancements to system on chip devices for iot perception layer," in *2017 IEEE International Symposium on Nanoelectronic and Information Systems (iNIS)*, pp. 151–156, IEEE, 2017.

[48] S. Bandyopadhyay, M. Sengupta, S. Maiti, and S. Dutta, "A survey of middleware for internet of things," in *Recent trends in wireless and mobile networks*, pp. 288–296, Springer, 2011.

[49] M. A. Razzaque, M. Milojevic-Jevric, A. Palade, and S. Clarke, "Middleware for internet of things: a survey," *IEEE Internet of things journal*, vol. 3, no. 1, pp. 70–95, 2015.

[50] D. Wichers and J. Williams, "Owasp top-10 2017," *OWASP Foundation*, 2017.

[51] M. Jensen, C. Meyer, J. Somorovsky, and J. Schwenk, "On the effectiveness of xml schema validation for countering xml signature wrapping attacks," in *2011 1st International Workshop on Securing Services on the Cloud (IWSSC)*, pp. 7–13, IEEE, 2011.

[52] T. Melamed, "An active man-in-the-middle attack on bluetooth smart devices," *Safety and Security Studies (2018)*, vol. 15, p. 2018, 2018.

[53] A. Ornaghi and M. Valleri, "Man in the middle attacks," in *Blackhat Conference Europe*, vol. 1045, 2003.

[54] Y. Sheffer, R. Holz, and P. Saint-Andre, "Summarizing known attacks on transport layer security (tls) and datagram tls (dtls)," *Internet Engineering Task Force Request for Comments*, vol. 7457, 2015.

[55] J. Hong, "The state of phishing attacks," *Communications of the ACM*, vol. 55, no. 1, pp. 74–81, 2012.

[56] S. Sanap and V. More, "Analysis of encryption techniques for secure communication," in *2021 International Conference on Emerging Smart Computing and Informatics (ESCI)*, pp. 290–294, IEEE, 2021.

[57] P. Morrissey, N. P. Smart, and B. Warinschi, "A modular security analysis of the tls handshake protocol," in *Advances in Cryptology - ASIACRYPT 2008* (J. Pieprzyk, ed.), (Berlin, Heidelberg), pp. 55–73, Springer Berlin Heidelberg, 2008.

[58] H. Kaur, R. Chauhan, M. A. Alam, N. A. Razaqi, and V. Chang, "Accessing sensor data via hybrid virtual private network using multiple layer encryption method in big data," in *Cognitive Informatics and Soft Computing*, pp. 187–195, Springer, 2020.

[59] J. Daemen and V. Rijmen, "Aes proposal: Rijndael," 1999.

[60] A. Bogdanov, F. Mendel, F. Regazzoni, V. Rijmen, and E. Tischhauser, "Ale: Aes-based lightweight authenticated encryption," in *International Workshop on Fast Software Encryption*, pp. 447–466, Springer, 2013.

[61] M. B. Yassein, S. Aljawarneh, E. Qawasmeh, W. Mardini, and Y. Khamayseh, "Comprehensive study of symmetric key and asymmetric key encryption algorithms," in *2017 international conference on engineering and technology (ICET)*, pp. 1–7, IEEE, 2017.

[62] D. Serpanos and T. Wolf, "Quality of service and security," *Architecture of Network Systems*, 2011.

[63] C. Paar and J. Pelzl, "The advanced encryption standard (aes)," in *Understanding cryptography*, pp. 87–121, Springer, 2010.

[64] T. Nie and T. Zhang, "A study of des and blowfish encryption algorithm," in *Tencon 2009-2009 IEEE Region 10 Conference*, pp. 1–4, IEEE, 2009.

[65] E. Consulting, "What is blowfish? who uses blowfish?." `https://www.tomshardware.com/news/ddr5-6400-ram-benchmarks-major-performance-gains-ddr4`, 2020.

[66] B. Schneier and D. Whiting, "A performance comparison of the five aes finalists.," in *AES Candidate Conference*, pp. 123–135, 2000.

[67] B. Schneier, "The twofish encryption algorithm.," *Dr. Dobb's Journal: Software Tools for the Professional Programmer*, vol. 23, no. 12, pp. 30–34, 1998.

[68] R. P. Adhie, Y. Hutama, A. S. Ahmar, M. Setiawan, *et al.*, "Implementation cryptography data encryption standard (des) and triple data encryption standard (3des) method in communication system based near field communication (nfc)," in *Journal of Physics: Conference Series*, vol. 954, p. 012009, IOP Publishing, 2018.

[69] B. A. Forouzan and D. Mukhopadhyay, *Cryptography and network security*. Mc Graw Hill Education (India) Private Limited New York, NY, 2015.

[70] S. R. Fluhrer and D. A. McGrew, "Statistical analysis of the alleged rc4 keystream generator," in *International Workshop on Fast Software Encryption*, pp. 19–30, Springer, 2000.

[71] N. AlFardan, D. J. Bernstein, K. G. Paterson, B. Poettering, and J. C. Schuldt, "On the security of rc4 in {TLS}," in *22nd {USENIX} Security Symposium ({USENIX} Security 13)*, pp. 305–320, 2013.

[72] A. Mousa and A. Hamad, "Evaluation of the rc4 algorithm for data encryption.," *Int. J. Comput. Sci. Appl.*, vol. 3, no. 2, pp. 44–56, 2006.

[73] G. Ye, K. Jiao, H. Wu, C. Pan, and X. Huang, "An asymmetric image encryption algorithm based on a fractional-order chaotic system and the rsa public-key cryptosystem," *International Journal of Bifurcation and Chaos*, vol. 30, no. 15, p. 2050233, 2020.

[74] U. Somani, K. Lakhani, and M. Mundra, "Implementing digital signature with rsa encryption algorithm to enhance the data security of cloud in cloud computing," in *2010 First International Conference On Parallel, Distributed and Grid Computing (PDGC 2010)*, pp. 211–216, IEEE, 2010.

[75] E. Barker and A. Roginsky, "Transitioning the use of cryptographic algorithms and key lengths," tech. rep., National Institute of Standards and Technology, 2018.

[76] P. Mahajan and A. Sachdeva, "A study of encryption algorithms aes, des and rsa for security," *Global Journal of Computer Science and Technology*, 2013.

[77] U. M. Maurer and S. Wolf, "The diffie–hellman protocol," *Designs, Codes and Cryptography*, vol. 19, no. 2, pp. 147–171, 2000.

[78] D. Boneh, "The decision diffie-hellman problem," in *International Algorithmic Number Theory Symposium*, pp. 48–63, Springer, 1998.

[79] V. Kapoor, V. S. Abraham, and R. Singh, "Elliptic curve cryptography," *Ubiquity*, vol. 2008, no. May, pp. 1–8, 2008.

[80] J. A. Berkowsky and T. Hayajneh, "Security issues with certificate authorities," in *2017 IEEE 8th Annual Ubiquitous Computing, Electronics and Mobile Communication Conference (UEMCON)*, pp. 449–455, IEEE, 2017.

[81] F. Forsby, M. Furuhed, P. Papadimitratos, and S. Raza, "Lightweight x. 509 digital certificates for the internet of things," in *Interoperability, Safety and Security in IoT*, pp. 123–133, Springer, 2017.

[82] K. Mahmood, S. A. Chaudhry, H. Naqvi, S. Kumari, X. Li, and A. K. Sangaiah, "An elliptic curve cryptography based lightweight authentication scheme for smart grid communication," *Future Generation Computer Systems*, vol. 81, pp. 557–565, 2018.

[83] F. Brandt, "Efficient cryptographic protocol design based on distributed el gamal encryption," in *International Conference on Information Security and Cryptology*, pp. 32–47, Springer, 2005.

[84] R. Ko and R. Choo, *The cloud security ecosystem: technical, legal, business and management issues*. Syngress, 2015.

[85] Y. Yang, X. Zhang, J. Yu, P. Zhang, *et al.*, "Research on the hash function structures and its application," *Wireless Personal Communications*, vol. 94, no. 4, pp. 2969–2985, 2017.

[86] R. Sobti and G. Geetha, "Cryptographic hash functions: a review," *International Journal of Computer Science Issues (IJCSI)*, vol. 9, no. 2, p. 461, 2012.

[87] V. Rao and P. K.V., "Light-weight hashing method for user authentication in internet-of-things," *Ad Hoc Networks*, vol. 89, pp. 97–106, 2019.

[88] S. Krendelev and P. Sazonova, "Parametric hash function resistant to attack by quantum computer," in *2018 Federated Conference on Computer Science and Information Systems (FedCSIS)*, pp. 387–390, IEEE, 2018.

[89] S. Chamoli *et al.*, "Docker security: Architecture, threat model, and best practices," in *Soft Computing: Theories and Applications*, pp. 253–263, Springer, 2021.

[90] C. Binkowski, S. Appel, and A. Aßmuth, "Securing 3rd party app integration in docker-based cloud software ecosystems," *CLOUD COMPUTING 2018*, p. 87, 2018.

[91] "Chapter 3 - an introduction to cryptography," in *Next Generation SSH2 Implementation* (D. Liu, M. Caceres, T. Robichaux, D. V. Forte, E. S. Seagren, D. L. Ganger, B. Smith, W. Jayawickrama, C. Stokes, and J. Kanclirz, eds.), pp. 41–64, Burlington: Syngress, 2009.

[92] M. Stevens *et al.*, "On collisions for md5," 2007.

[93] D. Rachmawati, J. Tarigan, and A. Ginting, "A comparative study of message digest 5 (md5) and sha256 algorithm," in *Journal of Physics: Conference Series*, vol. 978, p. 012116, IOP Publishing, 2018.

[94] R. Roshdy, M. Fouad, and M. Aboul-Dahab, "Design and implementation a new security hash algorithm based on md5 and sha-256," *International Journal of Engineering Sciences & Emerging Technologies*, vol. 6, no. 1, pp. 29–36, 2013.

[95] B. Preneel, *Davies–Meyer Hash Function*, pp. 136–136. Boston, MA: Springer US, 2005.

[96] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "Hypertext transfer protocol–http/1.1," 1999.

[97] E. Rescorla *et al.*, "Http over tls," 2000.

[98] A. T. Vasques and J. J. Gondim, "Amplified reflection ddos attacks over iot reflector running coap," in *2020 15th Iberian Conference on Information Systems and Technologies (CISTI)*, pp. 1–6, IEEE, 2020.

[99] E. Rescorla, H. Tschofenig, and N. Modadugu, "The datagram transport layer security (dtls) protocol version 1.3 draft-ietf-tls-dtls13-32," 2019.

[100] D. Locke, "Mq telemetry transport (mqtt) v3. 1 protocol specification," *IBM developerWorks Technical Library*, vol. 15, 2010.

[101] S. Jaloudi, "Mqtt for iot-based applications in smart cities," *Palestinian Journal of Technology and Applied Sciences (PJTAS)*, no. 2, 2019.

[102] E. Foundation, "Iot developer survey," *IoT Developer Survey 2019*, June 2019.

[103] L. R. Hernandez Ramos Santiago, Villalba M. Teresa, "Mqtt security: A novel fuzzing approach," *Wireless Communications and Mobile Computing*, vol. 2018, p. 8261746, Feb 2018.

[104] K. Alghamdi, A. Alqazzaz, A. Liu, and H. Ming, "Iotverif: An automated tool to verify ssl/tls certificate validation in android mqtt client applications," in *Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy*, pp. 95–102, 2018.

[105] P. Anantharaman, M. Locasto, G. F. Ciocarlie, and U. Lindqvist, "Building hardened internet-of-things clients with language-theoretic security," in *2017 IEEE Security and Privacy Workshops (SPW)*, pp. 120–126, IEEE, 2017.

[106] A. Palmieri, P. Prem, S. Ranise, U. Morelli, and T. Ahmad, "Mqttsa: A tool for automatically assisting the secure deployments of mqtt brokers," in *2019 IEEE World Congress on Services (SERVICES)*, vol. 2642, pp. 47–53, IEEE, 2019.

[107] L. G. Araujo Rodriguez and D. Macêdo Batista, "Program-aware fuzzing for mqtt applications," in *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis*, pp. 582–586, 2020.

[108] G. A. Member, "Positive technologies: 5g security issues." `https://www.gsma.com/membership/resources/positive-technologies-5g-security-issues/`, 2019.

[109] N. Junior, A. Silva, A. Guelfi, M. de Azevedo, and S. Kofuji, "Lightweight and secure publish-subscribe system for cloud-connected ultra low power iot devices," *Journal of Communication and Information Systems*, vol. 36, pp. 110–113, 01 2021.

[110] A. Aijaz, "Private 5g: The future of industrial wireless," *IEEE Industrial Electronics Magazine*, vol. 14, no. 4, pp. 136–145, 2020.

[111] O. Sievierinov and O. Kholosha, "Securing bearer token in oauth2. 0," *COMPUTER AND INFORMATION SYSTEMS AND TECHNOLOGIES*, 2021.

[112] Y. A. Gao, J. Basney, and A. Withers, "Scitokens ssh: Token-based authentication for remote login to scientific computing environments," in *Practice and Experience in Advanced Research Computing*, pp. 465–468, 2020.

[113] B. Dipina Damodaran, S. Salim, and S. M. Vargese, "Performance evaluation of mysql and mongodb databases," *Int. J. Cybern. Inform.(IJCI)*, vol. 5, 2016.

[114] A. Guru and A. Ambhikar, "A study of cryptography to protect data from cyber-crimes," *Research Journal of Engineering and Technology*, vol. 11, no. 2, pp. 45–48, 2020.

[115] M. Wyatt, "What is an api? a digestible definition with api examples for ecommerce owners."

[116] "Fiware orion official website." `https://fiware-orion.readthedocs.io/en/master/index.html`.

[117] J. D. Franco, T. A. Ramirez-delReal, D. Villanueva, A. Gárate-García, and D. Armenta-Medina, "Monitoring of ocimum basilicum seeds growth with image processing and fuzzy logic techniques based on cloudino-iot and fiware platforms," *Computers and Electronics in Agriculture*, vol. 173, p. 105389, 2020.

[118] "Nginx official website." `https://www.nginx.com/`.

[119] "Why netflix chose nginx as the heart of its cdn." `https://www.nginx.com/blog/why-netflix-chose-nginx-as-the-heart-of-its-cdn/`.

[120] "Docker official website." `https://www.docker.com/`.

[121] N. Tansangworn, "Development of iot edge hub for wireless sensor networks based on docker container," in *2020 IEEE International Conference on Smart Internet of Things (SmartIoT)*, pp. 356–357, IEEE, 2020.

[122] M. O'Leary, "Apache apache and modsecurity," in *Cyber Operations*, pp. 721–788, Springer, 2019.

[123] D. Kumar, Z. Wang, M. Hyder, J. Dickinson, G. Beck, D. Adrian, J. Mason, Z. Durumeric, J. A. Halderman, and M. Bailey, "Tracking certificate misissuance in the wild," in *2018 IEEE Symposium on Security and Privacy (SP)*, pp. 785–798, IEEE, 2018.

[124] M. Ukrop, L. Kraus, V. Matyas, and H. A. M. Wahsheh, "Will you trust this tls certificate? perceptions of people working in it," in *Proceedings of the 35th Annual Computer Security Applications Conference*, pp. 718–731, 2019.

[125] "Filo scottile's mkcert." https://github.com/FiloSottile/mkcert.

[126] A. Saleel, M. Nazeer, and B. D. Beheshti, "Linux kernel os local root exploit," in *2017 IEEE Long Island Systems, Applications and Technology Conference (LISAT)*, pp. 1–5, IEEE, 2017.

[127] S. Kumaran S and S. Kumaran S, "Containers and security," *Practical LXC and LXD: Linux Containers for Virtualization and Orchestration*, pp. 145–155, 2017.

[128] I. Giannakopoulos, K. Papazafeiropoulos, K. Doka, and N. Koziris, "Isolation in docker through layer encryption," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pp. 2529–2532, IEEE, 2017.

[129] D. Aivaliotis, *Mastering NGINX*. Packt Publishing Ltd, 2016.

[130] "Yandex's gixy repository." https://github.com/yandex/gixy.

[131] "Digital ocean's official website." https://www.digitalocean.com/.

[132] S. Chellappan and D. Ganesan, "Mongodb security," in *MongoDB Recipes*, pp. 215–241, Springer, 2020.

[133] H. Hasija and D. Kumar, "Compression & security in mongodb without affecting efficiency," in *Proceedings of the Second International Conference on Information and Communication Technology for Competitive Strategies*, pp. 1–6, 2016.

[134] J. He, J. Liu, Y. Shen, X. Jiang, and N. Shiratori, "Link selection for security-qos tradeoffs in buffer-aided relaying networks," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 1347–1362, 2019.

[135] "Heat has a negative affect on electronics." https://element5digital.com/heat-has-a-negative-affect-on-electronics/.

[136] D. Dinculeană and X. Cheng, "Vulnerabilities and limitations of mqtt protocol used between iot devices," *Applied Sciences*, vol. 9, no. 5, p. 848, 2019.

[137] A. Marowka, "Python accelerators for high-performance computing," *The Journal of Supercomputing*, vol. 74, no. 4, pp. 1449–1460, 2018.

[138] F. He, X. Hu, S. Liu, T. Li, K. Zhu, X. Bao, and C. Jiang, "Algorithm for improving processor utilization in multi-core processor environment by python language," in *2021 IEEE 4th Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC)*, vol. 4, pp. 775–779, IEEE, 2021.

[139] "ory kratos client v0alpha1api." https://github.com/ory/sdk/blob/master/clients/kratos/python/docs/V0alpha1Api.md#initialize_self_service_registration_flow_without_browser.

[140] S. Longari, A. Cannizzo, M. Carminati, and S. Zanero, "A secure-by-design framework for automotive on-board network risk analysis," in *2019 IEEE Vehicular Networking Conference (VNC)*, pp. 1–8, IEEE, 2019.

[141] M. Nottingham and R. Fielding, "Additional http status codes," *Internet Engineering Task Force (IETF)*, 2012.

[142] B. Dzogovic, B. Santos, V. T. Do, B. Feng, N. Jacot, and T. Van Do, "Connecting remote enodeb with containerized 5g c-rans in openstack cloud," in *2019 6th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/ 2019 5th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom)*, pp. 14–19, June 2019.

[143] M. D. Da Silva and H. L. Tavares, *Redis Essentials*. Packt Publishing Ltd, 2015.

[144] "Stack overflow survey 2020." `https://insights.stackoverflow.com/survey/2020#technology-most-loved-dreaded-and-wanted-databases`.

[145] S. Chen, X. Tang, H. Wang, H. Zhao, and M. Guo, "Towards scalable and reliable in-memory storage system: A case study with redis," in *2016 IEEE Trustcom/BigDataSE/ISPA*, pp. 1660–1667, IEEE, 2016.

[146] K. Jangla, "Docker compose," in *Accelerating Development Velocity Using Docker*, pp. 77–98, Springer, 2018.

[147] Z. Liu, "Ddr5-6400 ram benchmarks show major performance gains over ddr4." `https://www.tomshardware.com/news/ddr5-6400-ram-benchmarks-major-performance-gains-ddr4`, 2021.

[148] F. Kerschbaum and M. Härterich, "Searchable encryption to reduce encryption degradation in adjustably encrypted databases," in *IFIP Annual Conference on Data and Applications Security and Privacy*, pp. 325–336, Springer, 2017.

[149] E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.3." RFC 8446, Aug. 2018.

[150] N. Aviram, K. Gellert, and T. Jager, "Session resumption protocols and efficient forward security for tls 1.3 0-rtt," *Journal of Cryptology*, vol. 34, no. 3, pp. 1–57, 2021.

[151] N. Naik, "Choice of effective messaging protocols for iot systems: Mqtt, coap, amqp and http," in *2017 IEEE international systems engineering symposium (ISSE)*, pp. 1–7, IEEE, 2017.

[152] M. Hayden, "National information assurance glossary, committee on national security systems," 2010.

Appendices: Appendices

Appendices: Device Emulator Code

B.1 Supervisor

```
class Supervisor(object):
    def __init__(self, nusers, ndevices, url, device_name, delay, minutes_duration,
    ↪  tls):
        self._nusers = nusers
        self._ndevices = ndevices
        self._device_kwargs = {
            'url': url,
            'access_token': None,
            'device_name': device_name,
            'supervisor': self,
            'delay': delay,
            'tls': tls }
        self._is_running = False
        self._setup_complete= False
        self._devices = []
        self.minutes_duration = minutes_duration
    @property
    def is_running(self):
        return self._is_running
    @property
    def is_setup_complete(self):
        return self._setup_complete

    def _stop(self):
        self._is_running = False
        [d.join() for d in self._devices]
```

B.2 Device

```
class Device(threading.Thread):
    def __init__(self, supervisor, url, device_name, thread_index, access_token =
    ↪  None, delay = 1., tls = True):
        threading.Thread.__init__(self, name='Device_%s' % thread_index)
        # Configuration Parameters
        self._supervisor = supervisor
        self._url = url
        self._device_name = device_name + str(thread_index)
        self._access_token = access_token
        self._delay = delay
        self._downtime_start = None
        self._registered = False
```

```
    self._tls = tls

% Send data over SSL connection
    def _send_data(self, req_session:Session):
if not self._registered:
    body = {"id": self._device_name,
            "type": "Sensor Type",
            "temp": {"type": "Value Type",
                     "value": "Value"}}
    req_endpoint = "/v2/entities"
    req_method = "POST"
else:
    body = {"id": self._device_name,
            "temp": {"value": "Value"}}
    req_endpoint = "/v2/entities/{}/attrs".format(self._device_name)
    req_method = "PATCH"

headers = {'Authorization': 'Bearer {}'.format(self._access_token)}
try:
    if self._tls:
        response = req_session(req_method, self._url, req_endpoint, body, headers)
    else:
        response = request(req_method, self._url, req_endpoint, body, headers,
         ↪  verify="cert.pem")
    if response.status_code in [200, 201, 204, 422]:
        # Message sent successfully / Device already registered
        self._registered = True
    elif (response.status_code == 403):
        # Token invalid or expired
        if self._tls:
            response = req_session.POST(self._url, "/getNewToken", body, headers)
        else:
            response = requests.POST(self._url, "/getNewToken", body, headers,
             ↪  verify="cert.pem")
        if (response.status_code == 200):
            # Replace current token with new one
            old_access_token = self._access_token
            self._access_token = response.text
            headers = {'Authorization': 'Bearer {}'.format(self._access_token)}
            if self._tls:
                response = req_session(req_method,self._url, req_endpoint, body,
                 ↪  headers)
            else:
                response = request(req_method, self._url, req_endpoint, body,
                 ↪  headers)
        # Packet Loss Cases
        else:
            pass
    else:
```

```
            pass
```

B.3 Scripts

B.3.1 User Registration

```
# Load Users' Info
with open('config/names.json', 'r') as infile:
    user_names = json.load(infile)

# Set variables
desired_users=emulator_settings['users']
userCredentials = []
user_count=0

for name in user_names['names']:

    # Check for desired users limit
    user_count=user_count+1
    if (desired_users<user_count):
        break

    # Construct User
    [first_name, last_name] = name.split(" ")
    user_email = "{}.{}@iot.crew".format(first_name, last_name)
    user_password = secrets.token_urlsafe(16)
    user_name = "{}_{}".format(first_name, last_name)

    json_res = json.loads(response.text)
    action_url = json_res['ui']['action']

    # Add registered user in list
    userCredentials.append({
        "first_name": first_name,
        "last_name": last_name,
        "user_password": user_password,
        "user_email": user_email,
        "user_name": user_name,
        "user_id":json_res['identity']['id']
    })

# End of registration. Store users in file
with open('config/registeredUsers.json', 'w') as outfile:
    json.dump(userCredentials,outfile, indent=2, sort_keys=True)
```

B.3.2 Login & Access Token

```
# Set Hydra Client Info
client_id = clientinfo["confClient"]["client_id"]
client_secret = clientinfo["confSecret"]["client_secret"]
redirect_uri = 'https://{}/emulator/callback'.format(NGINX_URL)
scope = ["offline_access"]

# Set URLs
code_url =
 ↪  "https://{}/hydra/oauth2/auth?client_id={}&redirect_uri={}&response_type=code&scope={}&state=aW9
 ↪  client_id, redirect_uri, scope[0])
login_url = "https://{}/kratos/self-service/login/api".format(NGINX_URL)
token_url = "https://{}/hydra/oauth2/token".format(NGINX_URL)

# Load registered users' info in userCredentials list
userCredentials=[]
with open('config/registeredUsers.json', 'r') as infile:
    userCredentials = json.load(infile)

localClient = MongoClient(
    "mongodatabasehost",
    27017,
    serverSelectionTimeoutMS=10,
    connectTimeoutMS=20000,
)
# Check connectivity
try:
  localClient.server_info()
except ServerSelectionTimeoutError:
  print("local server is down.")
  exit()
dbCollection = localClient["iotUsers"]["users"]

storeCredentials = []

for user_identity in userCredentials:

  # Get Login Flow from Kratos
  response = requests.request("GET", login_url, verify="cert.pem")
  json_res = json.loads(response.text)
  action_url = json_res['ui']['action']

  # Submit Login form to Kratos
  payload = json.dumps({
    "method": "password",
    "password": user_identity["user_password"],
    "password_identifier": user_identity["user_email"]
  })
```

```
  headers = {
    'Content-Type': 'application/json'
  }
  response = requests.request("POST", action_url, headers=headers, data=payload,
   ↪  verify="ssl_certificate.pem")

  json_res = json.loads(response.text)
  session_token = json_res["session_token"]

  token = json.loads(response.text)

  # Add registered user in list
  storeCredentials.append({
    "user_token":token["access_token"],
    "user_id":user_identity["user_id"]
  })
```

B.3.3 Delete Users

```
if (os.path.exists('metrics')):
    shutil.rmtree('metrics')

if (os.path.exists('config/registeredUsers.json')):
    localClient = MongoClient(
        "mongodatabasehost",
        27017,
        serverSelectionTimeoutMS=10,
        connectTimeoutMS=20000,
    )
    # Check connectivity
    try:
        localClient.server_info()
    except ServerSelectionTimeoutError:
        print("local server is down.")
        exit()

# delete orion
    localClient.drop_database("orion")

    # Load registered users' info
    userCredentials=[]
    with open('config/registeredUsers.json', 'r') as infile:
        userCredentials = json.load(infile)

    # Enter a context with an instance of the API client
    with ory_kratos_client.ApiClient(configuration) as api_client:
        # Delete generated user identities
```

```
api_instance = v0alpha1_api.V0alpha1Api(api_client)
for user_identity in userCredentials:
    user_id=user_identity["user_id"]
    try:
        # Delete
        api_response = api_instance.admin_delete_identity(id=user_id)
        userCollection.delete_one({"id":user_id})
        deviceCollection.delete_many({"userID":user_id})
```