Università degli Studi di Padova

Dipartimento di Matematica "Tullio Levi-Civita"

Master Thesis in Computer Science

Study and design of an application for door smart-locks and smart-keys

Supervisor
Prof.ssa Eleonora Losiouk
Università di Padova

Master Candidate
Nicola Salvadore

Don't let us forget that the causes of human actions are usually immeasurably more complex and varied than our subsequent explanations of them.

(Fyodor Dostoevsky — The Idiot)

Abstract

With the increasing use of platforms on which an individual can make available his property for short-term rent, it grows the need of a smart way to manage the access rights to their holding. The common way for the guests to check-in or check-out during the accommodation involves physical key. For a host who lives or is located far away from the rented property, this can be time consuming and sometimes expensive to manage in person. Nowadays, many companies have developed various solutions for this problem, from key-pads, using temporary code numbers, which the guests must insert to have access grants, to properly door smart-locks, unlockable by simply approaching with the smartphone. This leads to the need to generate and manage secure virtual keys, which is a process that can even be automated, given the arrival and departure dates of the guests.
In this thesis, I describe my work at Kuama s.r.l., the company with which I designed and developed Kerbero.
Kerbero is an application that interfaces with smart-locks, in order to generate and manage secure virtual keys. It is designed to communicate with the external APIs of different smart-lock vendors, in order to retrieve and manage the devices available to the host. Moreover, it is able to generate and send temporary virtual keys to guests based on the reservation details provided. Kerbero is composed of a REST API and a Single-Page Application (SPA), which implements part of these features.
As such, in this document, I discuss in detail the analysis, the design, and the technical choices performed during this project.

Contents

Listing of figures

Listing of tables

1 Introduction

The most annoying and time consuming thing of renting a property for short periods is managing the check-in and check-out of guests. Airbnb hosts, for example, are compelled to meet their guests. They are obliged to deliver the keys and, at the end of the accommodation, they have to retrieve them. This turns out to be a problem when the property is located far from the owner, who has two alternatives: to hire a manager or to lose every time a huge amount of time reaching his rented house.

A solution to this problem comes from the Internet of Things (IoT) world. In recent years, smart-locks technology has developed very quickly. Smart-locks are devices that can replace the common door lock, in order to abandon the physical keys using. Smart-locks have different implementations from the simple keypads, to devices that can simply turn the key for you. The market for this technology is growing constantly, is valued at USD 1.64 billion in 2021, with a growth rate of 19.5% and in 2030 it will reach a value of USD 8.13 billion[3].

The advantages of not using physical keys are multiple: from security reasons, thanks to the fact that you cannot lose them, to the possibility of remotely managing everything, from the opening and closing process, to the selection of who is handling the access permissions. Moreover, if you are managing large properties with multiple entrances or areas, you can model a central system that can control all the doors of the aforementioned building.

The use of smart-locks in rented properties is constantly increasing, thanks to the spread of platforms such as Airbnb. There are several reasons, such as the possibility of reducing the

managing time of the listed property, when the owners do not want to manage its houses full-time and even the security that this type of device can provide. As such, they allow one to lock and unlock their rental properties remotely using the smartphone, which, for example, can be especially useful in case of an emergency.
There are also some potential drawbacks and challenges of using smart-locks in the Airbnb context. One concern is the upfront cost of purchasing and installing these devices, which can be significant for hosts. In addition, there is the risk of technical issues or malfunctions, which could cause delays or inconvenience for guests. Finally, there is the potential for privacy concerns, since the smart-locks can record data about who has accessed a property and when.
However, the monitoring feature is not always a drawback, since, from the host's perspective, smart-locks offer an additional security layer, especially for owners who are not present at their properties during the entire duration of a guest's stay. We will explore later that this trust balance between guests and host is important in platform like Airbnb, and how a smart-lock can have an impact on that.
With the collaboration of Kuama s.r.l., I developed Kerbero, an application to manage and interface with multiple smart-locks and generate secured keys. Kerbero is developed modularly, since he can provide support to different smart-locks models and providers, through the simple adding of plugins. The first vendor that we started to provide support was Nuki. Nuki develops smart-locks to open and close the door, which are simple, secure and quite affordable. The peculiarity of these devices is that they do not need to replace the existing door system, since the smart-lock is designed to automatically turn on the existing system. Nuki has several interfaces, but the common way is Bluetooth using the proprietary application. Another way is through the Web application, but the smart-lock must be connected to the Internet via a dedicated bridge.
Kerbero provides a host-oriented user interface, which allows owners to connect their Nuki devices through the internet, linking their existing account. The application can generate temporary virtual keys that can be assigned to upcoming guests. Kerbero is designed to be linked with vacation rental management services, such as Airbnb or Booking, to retrieve the reservation information and automatically create the keys based on these data.
In this thesis, we will explore the context in which the application has been conceived and the technologies involved in it, the analysis, the design and, finally, the resulting evaluation of the project.

1.1 Thesis Outline

The second chapter describes the context in which the project is developed and the scenario covered.

The third chapter provides an insight on the smart-lock technologies, the interfaces and the security vulnerabilities.

The fourth chapter summarizes the Nuki solutions as a case study, with insight into the component and framework with which their devices work.

The fifth chapter describes the analysis, the design and the implementation choices taken during the Kerbero project.

The sixth chapter is an evaluation of the choice made during analysis and design, with a focus on the results and the adopted workflow.

The last chapter is the conclusion of the thesis, in which I analyze personal achievement and present a critical evaluation of my work.

2 The project

In the last twenty years, with the advancement of technology, many Internet-based accommodation reservation systems have been developed, allowing travelers to find a place for the night in a fast and easy way. There are two types of online accommodation booking system[4]:

- property management system, used mainly by hotel or hotel groups to manage their operations;
- vacation rental management system, which is for anyone who wants to offer or book a non-hotel accommodation.

The first type can easily manage the guests check-in and the room lock access, and all the problems derived by having physical keys, with a simple front desk. However, in a vacation rental management system, the previous process can be a problem. The features of the existing platforms grant the property listing, booking, and transactions. The most common way to manage the check-in process is passing the physical keys, and that requires the encounter between the guests and the host, or sometimes a paid collaborator. This is particularly time-consuming for both the locator and the lessee, opening up to the risks of losing keys, or even worse, such as housebreaking and theft. Furthermore, we can identify two more categories in the vacation rental management system, which involve whether the host lives on the

rented property or not, which are, respectively, remote hospitality and on-site hospitality[5]. While in the latter the check-in management requires little effort because it does not involve distance, with remote hospitality it becomes more complicated. A more common scenario is the second type of hosting, in which a properties owner, which lives in a city, has an unused property on the mountain or near the sea, which is listed on a vacation rental management platform. As a consequence, he must go back and forth in order to manage the rented house. The most widely used vacation rental management system is Airbnb. This platform counts 6 million active listings worldwide, more than 4 million hosts and it covers more than 220 countries and regions. The reason Airbnb is growing so much inside the market is the ease-of-use of its services. Another reason is that the platform try to instill a sense of trust between the two parts. Platforms operating inside a *sharing economy* [6] puts a lot of weight on the power of the reviews. Both hosts and guests are encouraged to leave a review on the platform itself. Airbnb has focused a lot on this aspect and the presence of a working review system is part of its success. Another reason for the popularity of Airbnb is the well-designed rent processes, which are fast, easy to use and secure, for all the actors involved in them. For this reason, they are developing more and more features to smooth out existing processes. Airbnb is, for example, pushing hosts to create smart homes to provide a more easy way to access and manage the property[7]. This new feature is called "self check-in", and it is gradually integrated into the platform. Airbnb outline three alternatives to enable self check-in option.

- Lockboxes are the most affordable solution. They are a simple locked storage for physical keys, accessible with temporary codes.
- Smart-locks are the object of this analysis, they provide temporary access to the building and they can be used with the vendor mobile application.
- Keypads allow guests to access by providing a code, previously generated and delivered by the host. They are often inserted into the smart-lock category.

This variety of solutions comes with the differences in cost and features, which can confuse the host. The exponential growth of the market contributes to exacerbate the situation. Many companies are proposing their solution, with different features and as a consequence cost. In the graph 2.1, the variations of prices are shown, in relation to the categories identified before.
Lockboxes are the less expensive alternative for a host but show clear shortcomings. First of

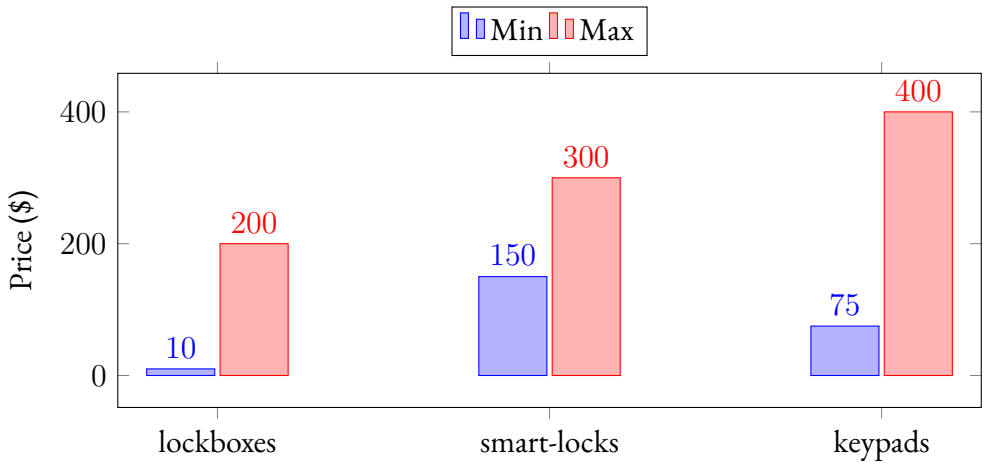


Figure 2.1: Average prices for online purchases of smart door lock solutions.

all, the use of physical keys is involved, which are exposed to the usual security problems. Additionally, the box must be placed externally, making it vulnerable to any kind of tempering. Keypads are not exposed to the lockboxes first problem, because they do not involve keys, but have to be placed externally, making them vulnerable to the public. Lastly, most smart locks have wireless interfaces and as such they have the ability to be designed to not expose any hardware to the exterior of the house. Smart-locks have other shortcomings, like the possibility of jamming, the discharge of batteries and the cybersecurity problems of the wireless protocol in use.

However, despite the price, a smart-lock seems to be the preferred solution for an host, because offers all the features remotely and let the guest feel safer. As shown in this investigation [6], in a population of Airbnb hosts in the United States who decide to install smart devices in their rented properties, all had installed a smart lock. Their opinion was mostly positive, except for the problems described before.

A final argument involves the advantage of using smart locks, when managing more than one property. Analyzing the data of the population of Airbnb[8] is easy to notice that the number of listings is significantly more than the number of hosts. As such, the host having more than one property listed can be approximately be the majority. Suppose you are a host with more than one property listed on a vacation rental management system, such as Airbnb, and live far from them, the common way he has to manage their guests is to go to their rented house and meet them in person. If the host decides to install a smart-lock or a keypad, then the problem is solved, but he has to face with another one. The number of solutions is dif-

ferent for each type of door and even the vendor can be different. Moreover, after this type of investment, what an host expects is a sort of automation, from the receiving of the book to the generation of the access permission.
The solution proposed in this thesis is an portal for host, in which he can connect all the smart-lock account he owns. This let him to pair the devices, control remotely, generate keys based on the reservation and send that to the guests, without caring about the smart-locks model and provider. The idea is to create a central tool to manage all the properties in one place and give the opportunity of linking all the rental management systems account.

3
The background

3.1 Smart homes

Since they were released, smartphones changed the way people do things and behave. The reason for their success was the addition of the portability feature to the computers. With portability and miniaturization of the hardware, including new sensors, which in a normal portable computer would lack usefulness, we now can fit very powerful devices into our pockets. Moreover, thanks to their sensors and interfaces, they are giving us the possibility to interact with objects in the environment, which before was not commonly defined as "smart". The interfaces we are talking about range from the wired and common USB, to the wireless Bluetooth, ZigBee, Wi-Fi and so on. Since smartphones are changing the market and products, it is common that if we take any object in a house and search on the Internet we should find a smart version of it. There are shutter models, for instance, that are automated and can be connected to the smartphone or a smart home device, all through the Wi-Fi interface. The widespread use of such products has developed a new discipline in computer science called the Internet of Things (IoT). The IoT refers to the interconnected network of physical devices, vehicles, buildings, and other items embedded with electronics, software, sensors, and connectivity, allowing these objects to connect and exchange data. IoT allows everyday objects to be connected to the Internet and to send and receive information concerning their functioning. Moreover, this connection allows for the collection and sharing

of data on the device's usage and status, enabling improvements in efficiency, accuracy, and economic benefit. However, the IoT does not impact only the lives of customers. Alongside robotics, this discipline is changing the industry by directly intervening in processes or collecting useful data.
Nonetheless, the focus of this thesis will be on smart homes, which gathers all the IoT devices that can be installed in our dwellings, and, since they are connected to the Internet, they can provide some automation to everyday actions. They include not only the shutters described above but many others, such as thermostats, TVs, and so on.
A study [9] shows how the pandemic changes the way we live, in particular inside our homes. In fact, people and companies are adopting the work from home, as such people who are not in the office all day, are experiencing changes in their home, in order to make them not only a better place to live, but also to be productive. The reasons why someone wants to buy a smart device can range from economic reasons, such as saving money and energy, to security or health reasons. A study on the benefits and risks of smart home devices [1], conducted on a sample of a hundred people in the UK, shows what customers think are the main benefits of having one of them installed at home (fig. 3.1). Therefore, it is not surprising that we are

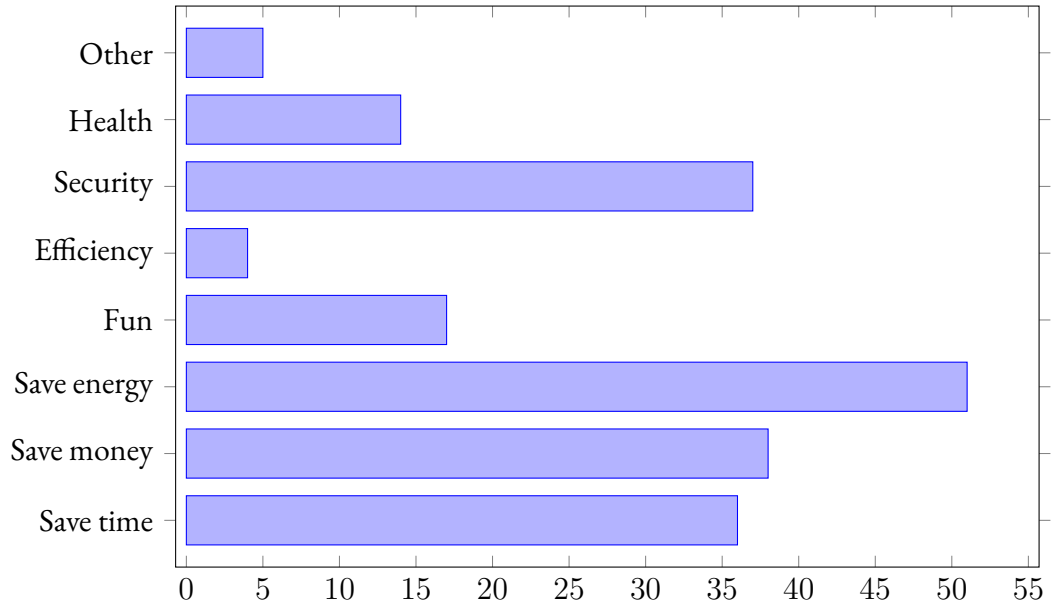


Figure 3.1: Histogram showing the frequency of responses to the question "How do prospective users perceive the specific benefits and risks of smart home technologies?" on a sample of a hundred people in the UK [1].

talking about a large and constantly growing market in which many companies are investing.

3.2 Smart-locks

Locks and keys, as we know them today, have not changed much since they were invented. A lock is installed inside the door profile and a deadbolt can fit a key with a specific shape, which turning can enable or disable the lock. This solution, with a few variations, has remained unchanged through the era, since it actually fits the purpose. However, the advent of IoT raises the question of whether having physical keys is always a benefit in terms of security. Keys until now were the best solution to prevent unwanted people from accessing a private property, but they can be stolen and cloned, even from the deadbolt shape.
As such, door lock manufacturers start to produce new models with a wireless interface, which drives their interest in the IoT market. The proposed solutions are really different from each other, due to the number of possible implementation and hardware involved, such as keypad, deadbolt, handle, fingerprint and so on; and even due to the type of interfaces, such as Bluetooth, ZigBee, NFC, Wi-Fi. A customer who experiences the purchase of this type of product must face a variety of devices, each with different technologies, and has to choose the one that fits his needs. Even the installation method, for example whether the device can operate with the existing lock or not, could be a discriminating factor. As such, this market is fragmented and there are many products that are spreading through it.

3.2.1 Smart-lock interfaces

A smart-lock, as it is intended in this thesis, *is an electromechanical device that performs locking/unlocking operations of a door, after receiving the command through a wireless interface.*
The emphasis on the wireless nature of a smart-lock is mandatory for us, since some definitions do not include this particular characteristic, as a consequence involving devices that are not in our interest.
The following types of smart locks differ from each other by the type of wireless interface used. In particular, what the context requires from the connection technology are the following features.

- High reliability, which can be translated into low power consumption. Smart-locks are usually based on lithium batteries, as such they are not connected to a stable power line. A wireless technology has always to deal with a standby power consumption value, which in this case must be low.
- Low cost: these devices are not usually cheap, but they have to reach a consumer market, in order to spread as much as possible.

- Secure: the implementation of a safe communication channel is a priority when we are considering a device which is designed to protect the home accesses. The focus is on the wireless protocol used to communicate, which must be secured and implemented with encryption.

The following wireless interfaces list are not complete, there are many other interfaces, like NFC, fingerprint and so on. They can be neglected, because they are not more widespread than the one presented or relevant for the purpose of the project. Moreover, it must be specified that smart-locks use not only an interface, but they usually use hybrid wireless connections of the following.

Bluetooth

Bluetooth is the commercial name of the standard IEEE 802.15.1, for short-range wireless technology. The protocol architecture can be divided into different layers.

- The core protocols are the low level management system, including, for example, the radio interface, the link manager protocol, which assures authentication, encryption and other security features.
- The cable replacement and telephony control are emulation protocols over the Logical Link Control and Adaptation Protocol (L2CAP) which provides an interface for each of the high level protocols.
- Other existing protocols, such as TCP/IP, which are customized to communicate with L2CAP.

Despite its age, Bluetooth is gaining popularity, especially in the last decade, with the spread of wireless devices, such as headphones, because it provides low latency. The main purpose of this technology is the wire replacement, but in the various updates, many other features have been added, like, for example, the possibility of internet bridging. Another important characteristic is that Bluetooth is designed to be low cost, eventually under $10/unit [10]. It assures a communication range of 10 meters, which is more than sufficient if we do not need remote control on the device. Finally, the telephony control protocol allows connected devices to handle both data and voice transmission. This is the main reason why a Bluetooth module is now installed in most smartphones.
As such, many smart-locks usually implement Bluetooth, because it meets all its requirements. The biggest limitation of this particular approach is the lack of remote control capability. The short-range nature of the protocol does not allow communication over 10 meters,

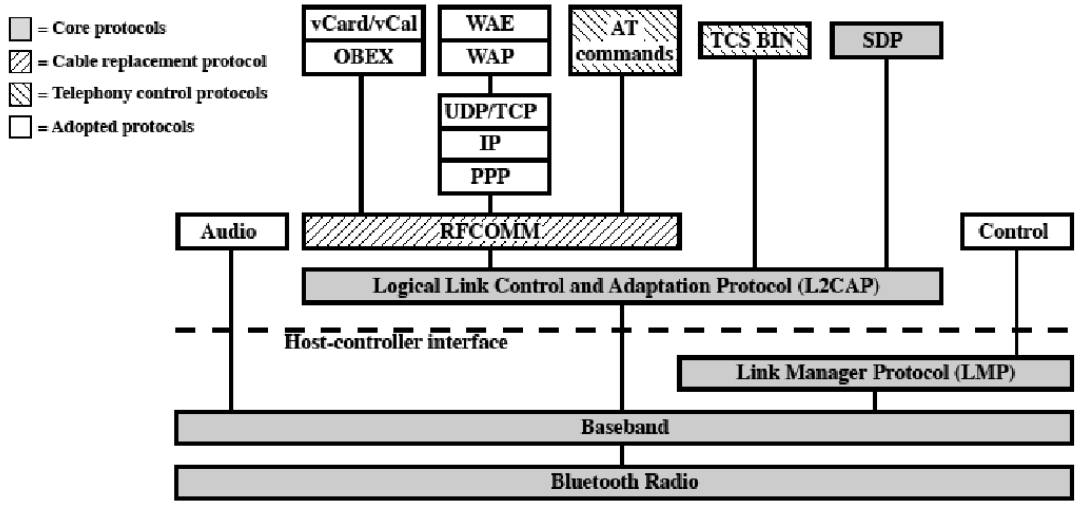


Figure 3.2: Bluetooth Protocol Stack.

and, to patch up this limitation, information must be forwarded by a bridge connected to World Wide Web.

Zigbee

IEEE 802.15.4 is a standard released in 2004 for a low-rate wireless network. The characteristics of this technology could be summarized as follows [11].

- Reliable and self-healing.
- Supports a large number of nodes.
- Secure, with standards based security [AES128].
- Low cost.
- Low power (ability to operate on batteries measured in years).
- Low maintenance (meshing, self organizing).

Part of these goals are reached through the Direct Sequence Spread Spectrum (DSSS) modulation technique, which grants a range up to 150 meters and a low power consumption, compared to the Frequency Hopping Spread Spectrum (FHSS), used by other technologies, including Bluetooth. Zigbee devices are usually organized on a network with a coordinator at the center, who is responsible for the initialization of the channels and security parameters.

Moreover, it is the component in charge of bridging to other networks type.
Then, there is the router, which acts as an intermediate node, accepting connections from the other devices and retransmitting to the receiver. Finally, the end devices are the ones operating actions, with sensors and switches.
Thanks to the technologies in use and the specific radio frequencies chosen, Zigbee devices can rely on batteries and last for years, due to the incredible low power consumption achieved in standby. We must not forget that IoT devices are on standby most of the time and the latent consumption must be lower than possible.
The performance of this technology is overall better than that of Bluetooth; however, Zigbee presents a non-negligible limitation, which is the absence of the dedicated modem inside almost all smartphones. While Bluetooth is widespread in the consumer market, thanks to its integration inside smartphones, ZigBee can rely only on the smart home implementation, when it is supported. A ZigBee smart home configuration usually has various sensors as nodes and a central server which acts as controller and it bridges to another network, which are commonly Wi-Fi or Ethernet. Only then, from the latter network, user applications can interface with the ZigBee devices.
In the case of smart-locks, the ZigBee interface, if implemented, is not used to operate directly with a smartphone, but to share information about access with the smart home application. However, communication can even occur, but must be bridged through another network to which the smartphone can connect, such as Bluetooth or Wi-Fi.

Wi-Fi

Wi-Fi is the name of a family of standards with the official name of IEEE 802.11. Wi-Fi technologies are commonly used to provide a wireless access point for the Internet, since they usually operate in WLAN. It is the most widespread wireless technology in the consumer market and the reasons for that are various. First of all, it supports high bandwidth, up to 600 Mbps with 2.4 Ghz frequency spectrum and 1.3 Gbps with 5 Ghz. Moreover, it is reliable, secure and its signal can reach high distances, for instance some of its extensions can reach 1 km of range.
Most mobile devices that connect to the Internet have a Wi-Fi modem inside, including smartphones. As such, it is always the primary choice when building a smart home that allows devices to communicate through its dedicated modem. This scenario is justified by the fact that Wi-Fi exists, in most houses, before the spread of the IoT, due to its use for Internet connection purposes. As such, most IoT devices and the relative application communicate

over a Wi-Fi network, which provides an eventual internet connection.
Smart-locks connected to Wi-Fi are usually remotely controllable. In fact, Internet access makes these devices available by HTTP requests over WWW.
From a security point of view, Wi-Fi provides different options. Most access points provide an authentication system with the WPA2/PSK protocol, which also allows the network to identify the connected devices. Furthermore, having access to the HTTP protocol allows applications to implement its security protocols, such as Transport Layer Security (TLS).
However, the use of Wi-Fi technologies can have downsides. For example, power consumption is much higher than in previous technologies. This is usually a problem with mobile devices that are powered by batteries and not connected to a power line. In the IoT, this could be bypassed in some ways, like using hybrid approaches. Many smart-locks, for example, use external bridges that can be connected to the power socket and communicate to the hardware with Bluetooth. As such, the smart-locks uses a low power protocol to communicate with the bridge and the latter forwards the data to the home server. Moreover, with the development of these protocols, power consumption has taken many steps forward. Bluetooth, for example, to fill the concurrency gap, has developed a low-power version, called Bluetooth Low Energy (BLE). BLE is able to consume less power approaching the communication with a client-server architecture, using smaller packets, improving the idle time with a sleeping mode and better managing the frequencies in use. Even Wi-Fi has recently introduced new protocols, such as Target Wake Time (TWT), which improves the wake-up time scheduling of devices.
It is important to specify that the values in the below table 3.1 are referring to older, but widespread implementations of the protocols. However, this table is useful for giving an idea of the overall differences between the technologies.

3.2.2 Smart-locks network design

Comparing the various solution proposed by the manufacturer, it is possible to identify a common pattern on how smartphone and smart-lock communicate. The most popular is the Device-Gateway-Cloud model [12] (fig. 3.3), in which the smartphone pairing acts as a gateway to the Internet sending the information to the provider server. This is forced by the lack of a Wi-Fi modem in most smart-lock solutions. However, if it is able to connect to the home network, smart-lock and server can communicate directly, as such the state updates are transmitted through the internet connection.

Standard	Bluetooth	ZigBee	WiFi
Chipset	BlueCore2	ZigBee Chip	CX5311
Range (m)	10	40-100	100
VDD (volt)	1.8	2.4-3.4	3.3
Bit rate (Mbps)	0.72	0.25	52
Battery Life (days)	1 - 7	100 - 1000	0.5 - 5

Table 3.1: Comparison of wireless technologies, focusing on power consumption and performance in the context of smart grid communication. [2]

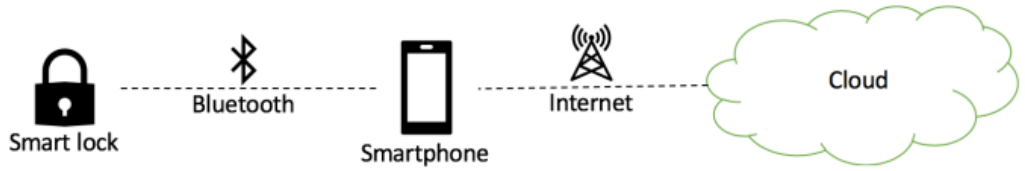


Figure 3.3: The Device-Gateway-Cloud model, the smart-lock use the smartphone as a gateway to the server.

3.2.3 Smart-lock application

The mobile application, installed in the smartphone, plays an essential role in smart-lock management. As we mentioned in the Device-Gateway-Cloud model, the application is responsible to collect data from the device and sent in to the cloud. Moreover, it is able to send request, like the unlock/lock one, to the smart-lock and listen to the answer. Most of the applications provided by the smart-lock manufacturer run on Bluetooth, and as such, the first communication between device and smartphone must be preceded by the pairing of the two. Pairing sets the wireless connection and, most of the time, creates a handshake key that grants communication even offline.
The unlocking process could be slightly different, depending on the smart-lock model. Some of the devices have to be physically touched and if the smartphone paired is in the Bluetooth range, it will unlock. Some others have a virtual button inside the application and do not expose any hardware to the exterior.
Moreover, within the app, the access log list is usually available, which is particularly useful in this context. Through this feature, the application can show the usage history of a key and

see who is entered in the house, as such it can also detect the unexpected access.
Finally, the killer feature is the possibility to generate virtual keys that can abstract different access levels in complex systems, or simply can generate temporary permissions.

3.3 Related work

The aim of the project is the integration of third-party smart-locks in a centralized application, which can easily manage them from a centralized tool. As such, it is easy to notice that the availability of public APIs is a strong precondition in the development of the project.
Fortunately, smart home appliances usually expose any kind of integration due to their intrinsic nature. The IoT world is relatively young and many products are released every year; as such, the market is very fragmented into multiple systems. The integration between these latter is essential when a manufacturer is designing a new device.
Moreover, smart home appliance customers want to build their own custom system, buying products following the philosophy of the cheapest, the most functional or the best integrated. It is common to find IoT devices integrated with Amazon Alexa, implementing the related skill, or with Google Assistant. However, most of the time people who buy smart home appliances have technical skills and usually require the device manufacturer to expose more advanced integration possibilities.
Those are the reasons for the availability of public APIs in mainly smart devices on the market. This is common even in smart-lock products, where the manufacturer ships the object with a well-stocked integration services collection. This brings many others developers to create integrated systems for their own projects and researcher to implement and test various solutions.

3.3.1 The RESTful architecture in a smart home system

Representational State Transfer (REST) APIs are a type of Web interface that uses HTTP requests to manipulate external resources. They are the easiest way to expose features worldwide exploiting the wide spread of the Web. The REST architecture is used in the smart home context to provide a standard and remote way to access information and to perform actions. Many works showed us how to integrate various devices in our home with this type of solution.
There is the possibility of creating, for example, a Smart Home Protocol (SHP)[13]. The solution divides the service into the following five parts.

- There is a setup procedure by which the controller devices (smart home, smart assistance, etc.) access the home network.
- Registration is the phase in which the controller device assists the controlled one, the smart home appliance, register to the cloud server.
- Then, the controller can acquire the information from the smart home device and perform actions based on those specific data.
- After a change occurs in the controlled devices, an event can be broadcast, and other devices react to that.
- Finally, a remote controller, located far from home or not connected to the same network of controlled devices, can perform actions and receive events due to the cloud connection.

The cloud modeling as REST API provides a specific URI for each of the resources provided. As such, the client installed in the controller device sends a request to the server in the cloud, which performs actions or retrieves information from the controlled device. The event handler is usually implemented as a webhook with a subscription-notification mechanism.

3.3.2 Handle a third-party adapter

Services implemented as an SHP can be exposed to a third-party application that can manage IoT devices. They are represented by software not developed by manufacturer of the product, for such third-party, which aims to gather all the functionalities of different devices, or to manage integration between them that is not natively supported. For example, a customer needs to synchronize its smart alarm with the smart light bulb to turning on when the alarm rings. Otherwise, a user owns the same sensor from different manufacturers and wants to manage them in a single application, to interface with another controller. In those cases, a developer can implement an adapter to achieve those goals. In particular, a solution similar to that has already been developed with smart-locks[14]. The solution must be clever, because it has to take into account many actors and phases. SHP can be refined with a specific procedure as follows:

- profile linking manages the user profile and external platform account;
- device inclusion must take place to select the appliance and sync with the application;

- commands processing, requests must be filtered and sorted to call different REST APIs.

Profile linking is a central process for third-party applications. The user must give the access and permission from the application and it is usually performed with an OAuth2 flow. As such, the application must present a link to the authentication provider of the smart-lock manufacturer account. After the authentication the user provides the confirmation on the permission required, then the application can retrieve and access token which can be used to perform the HTTP requests. Inclusion of the device is implemented by QR scanning on the device that provides the serial number[14], but it depends on the procedure implemented by the smart lock vendor. Finally, when an action is initiated in the smart home application by the user, it is transmitted to the gateway. Since the remote device is controlled via an external cloud, the command is sent to the smart lock adapter located in the cloud. The adapter then deciphers the message and sends it to the third-party API, causing the device to change state, such as unlocking the door. For this process to be successful, the request must have the linked user's access token attached. The reverse occurs when someone unlocks the door, with the smart lock indicating the change in state to the cloud, which then sends the change event to the registered webhook. The smart home adapter receives this message and informs the rest of the smart home system of the change. The smart lock supports various commands and states, such as locking/unlocking, notifications when the door is open or closed, low battery alerts, and the possibility to unlock the door using a keypad or pin code (thus requiring CRUD operations for pins, etc.).

3.3.3 Smart-lock vulnerabilities

Because of the importance of the subject, the literature is treating smart-lock device security issues with particular attention. A possible spreading of this technology in everyone's homes could be a new opportunity for new cyber-thieves, which want to break into not them properties.
Sometimes, vulnerabilities are based on device design, so it is very important to discover these defects during their analysis. For example, a basic type of attack could affect the hardware directly. We all know that tech gadgets most of the time are not built to withstand, for various reasons, from maintenance costs to production costs. As such, it is better that a smart-lock does not expose any mechanical part to the outside face of the door. Otherwise, anyone could have access to the device, understand the model and act accordingly.

However, due to the design of a smart-lock, it is not always possible to hide the hardware. As such, we can only analyze software vulnerabilities.

Architectural attacks

This type of attacks can be executed remotely because they rely on architectural vulnerability, in particular in communication between the gateway and the server.

Man-In-The-Middle attacks. This type of attack can be performed by routing the communication flow through a malicious proxy by changing the API URL parameter, which is the server address to which the smartphone application is connecting[15]. As such, the entire data passing through the proxy can be read and the proxy can even fake the response to the application. This can still happen in a system using certificates signing the software involved, because the application does not know that the server to which it is connecting is a proxy. As such, it keeps sending the certificate, which can be easily forwarded to the real server by proxy, leaving no trace.

Overpriviledge attacks. As we have seen before, many smart lock manufacturers expose public API to third-party developers. This is not only a benefit for who is integrating the smart-lock functionality, but even for the provider, can have positive effects, such as a better user experience, thanks to a richer ecosystem.
However, this approach can expose a critical part of the system to software that is not under the control of the manufacturer. In fact, most of the APIs are accessible with OAuth authentication flows, which require a client ID to prove the identity. In the case that a developer lets this client ID be in plain text, accessible with a simple code inspection, this can represent a serious vulnerability.
This category includes attacks that can be performed on bad design APIs by the manufacturer itself. That can happen if a third-party application is requesting a type of privileges, for example, reading the status of the battery, but the bad designed library gives them access to resources which are not requested, or worse, are meant to be hidden.[16]

Eventual consistency. Eventual consistency verifies when data in the smart-lock application and the server must remain continuously synchronized. This is a problem, especially in the Device-Gateway-Cloud model, where the smart-lock state in the server is a projection of the state on the application. In this type of architecture, the two states must be synchronized

or the system can accused vulnerabilities. For example, if the owner creates a one time key, giving permission to a temporary guest, and then he/she revokes the permission on the key, if the guest smartphone is not reachable, because he/she can simply had disabled the internet connection, the server can not revoke the permission. As such, the guest application can still have access until the connection is restored and the synchronization is available again. [12]

3.3.4 Application attacks

Most of the time, the mobile application is the vital part of the system. As such, its vulnerabilities are serious points of failure. In particular, are not uncommon attacks on the communication between the application and the smartphone.
As mentioned in section 3.2.3, most applications use a Handshake Key, in order to establish a connection between the smartphone and the smart-lock. In the mobile operating system, there are many ways to access sensitive information from the application. One can be using a rooted/jailbroken operating system in which the smartphone owner acts as a superuser, so that he/she is able to access the storage of the device and see protected data. As has been shown in other works[17], the Handshake Key can be stolen simply by accessing a XML file from the manufacturer's application.

Denial of Service (DoS)

With the possibility of controlling the communication between the smartphone and the smart-lock, the attacker can disable the functionality of the device. It has been shown that if a smart-lock can handle only a Bluetooth connection at the time and there is no priority system, the attacker can continuously send a link request to the device, preventing the owner/user from connecting with its application[17].

Storage attacks

We have described above how to steal the Handshake Key. However, with the same method, it is possible in some cases to steal even personal information from the application storage. This happens primarily due to the application developers fault, which is letting the data save without encryption.

3.3.5 Preventing the attacks

As we can see in the above sections, most of the fault, of the vulnerability presence, relies on developers' care. Sometimes, a well-designed system is enough to prevent unnecessary information from reaching the wrong hands. However, when this is not possible, especially in mobile application storage, it is always best practice to use a state-of-the-art cryptosystem to protect sensitive data.

Thanks to their features, smart locks can even be improved with more security layers by design. Some of the manufacturer applications are able to use the GPS location of the user and determine if he/she is in the near the device (geo-fencing). With this system, it is easy to prevent malicious applications from gaining the unlock challenge too easily.

However, from the perspective of a third-party application developer, the security is based on the correct implementation of the TLS protocol. This most of the time grants that only certificated client can access to the sensitive information on the server. Moreover, appropriate method for the account management and the securing of the authentication are essential in this context to not provide a major vulnerability in the system.

4

Nuki study case

Nuki is an Austrian company, which design and distribute smart-lock solutions for smart homes and offices. We decided to analyze the Nuki smart-locks first, mainly because there was the immediate availability of these devices and, second, for their security and ease of use. Moreover, Nuki follows an open software integration philosophy and has a large community supporting its forum and blog.
The main characteristic of their devices is the lack of replacement parts for your existing "analog" lock. Smart-locks are designed to work with the already installed door key and deadbolt, with only a few steps to mount and unmount the device.
They already offer support for many of the vacation rental management systems, integrating smart-lock functionality with reservation information. Moreover, they grant support to third-party developers to integrate their system, with many options like their REST API.

4.1 Nuki components

Nuki offers different configurations to its customers. For the project, we used two of their products: the smart-lock and the bridge.
The main characteristic of a Nuki smart-lock is the implementation of a small electric drive, which is capable of turning the key inside the deadbolt (fig. 4.1). The device can be mounted inside the house, making it physically inaccessible from the outside. Moreover, its structure

depends only on the shape of the key, for which a few adapters are provided. As such, it is compatible with almost any double cylinder lock.

Figure 4.1: The smart-lock parts and the mounting system.

The user can interact with the button placed on the device to lock and unlock the door, but only from the inside of the house. Moreover, the device still allows for the use of the physical key from outside the building, which is particularly useful in case of emergency.

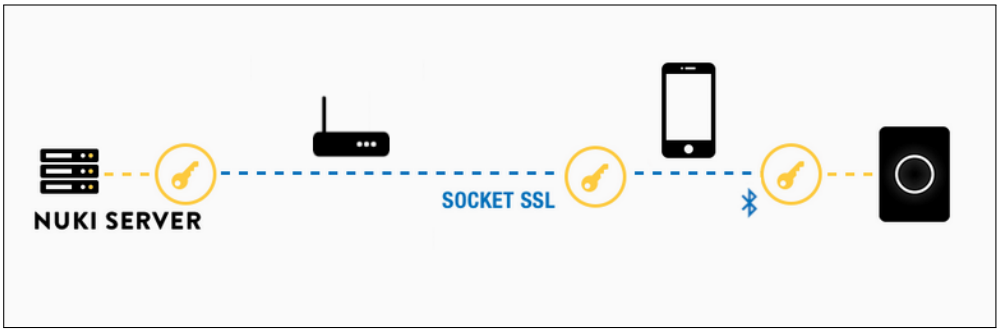
This type of smart-lock can work standalone with the Device-Gateway-Cloud architecture described in section 3.2.2. The device uses Bluetooth Low Energy (BLE 5) to pair and communicate with an ad-hoc mobile application, available for Android and iOS. This allows the user to connect in the limited Bluetooth range. As such, if we want to interact with the smart-lock remotely, we have to buy a bridge.

A Nuki bridge connects the smart-lock to the home Wi-Fi network, allowing the device to access the Internet. The bridge uses two protocols: a BLE module, which interacts with the smart lock, and IEEE 802.11 to communicate with the home modem. The choice of using separate solutions allows the smart lock device to prevent batteries from draining due to the power consumption of the Wi-Fi module. In fact, to work, the bridge must be connected to an electric outlet, within range of the smart-lock.

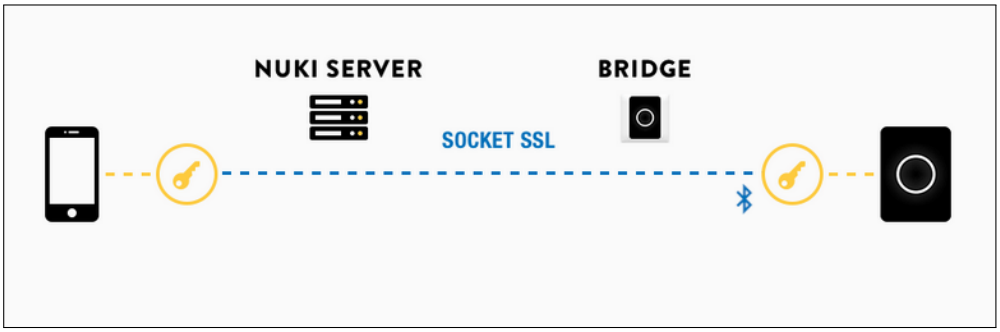
Figure 4.2: A Nuki bridge.

There are other products that allow interaction with the smart-lock, which are the keypad and a clicker, called the fob, to replace the mobile application. Moreover, provides an opener to integrate with the home intercom and door sensors, which are not subject of this thesis.

The out-of-the-box configuration of the smart-lock involves only the device and a smartphone. The application on the smartphone performs a pairing with the smart-lock, creating (but not exchanging) the key used to communicate. Then, the mobile application acts as a gateway, because the smart-lock is not directly connected to the Nuki server (fig. 4.3a). All the information and the state of the smart-lock are gather and sent to the cloud by the smartphone.



(a) Device-gateway-cloud configuration with a Nuki smart-lock.



(b) A Nuki smart-lock connected directly with the server through the bridge.

Figure 4.3: Two different configuration for the Nuki devices.

In the second figure 4.3b, we can see that the bridge replaces the smartphone and connects the smart-lock directly to the server. If the bridge is available, not all the operations, launched from the smartphone, pass through the server. Opening and closing actions, for instance, act as in the Device-Gateway-Cloud configuration if the device is in Bluetooth range. However, immediately after this operation the smart-lock is able to update its state on the server, contributing to maintain a consistent state.
The two wireless technologies involved are Bluetooth and Wi-Fi.

4.2.1 Nuki Bluetooth protocol

As we mentioned in section 4.1, the Nuki smart-lock uses BLE 5 to communicate with the smart-lock and the bridge. To achieve low power data transfer, the technology uses a protocol named the Attribute Profile (ATT)[18]. This protocol is responsible for storing data in the form of tables. The fields of this table are the following:

- a nonce, which identify an handler,
- an attribute type, which is defined by an Universal Unique Identifier (UUID),
- the read/write permission,
- the attribute value.

Along with the ATT protocol, BLE 5 uses the General Attribute Profile (GATT), which is an abstraction layer that allows communication in a client(the smartphone)-server(the smart-lock) architecture. This protocol, combined with advertising and a caching system, permits BLE to reach low power consumption, comparable with the Zigbee's performance.
The protocol developed by Nuki to exchange data uses the following message format (table 4.1).

ADATA			PDATA			
nonce	authorization id	message length	authorization id	command id	payload	CRC
24 Byte	4 Byte	2 Byte	4 Byte	2 Byte	n Byte	2 Byte
unencrypted	unencrypted	unencrypted	encrypted			

Table 4.1: Transfer format for encrypted BLE messages.

As we can see from the table, the protocol information is in the ADATA section, which is not encrypted. Moreover, in the PDATA, we can find the attribute value with a command identifier and the payload, which are encrypted.
Following an example of a read lock state command flow.

Shared key: 217FCB0F18CAF284E9BDEA0B94B83B8D
10867ED706BFDEDBD2381F4CB3B8F730

Authorization-ID: 2

- Client (CL) writes Request Data command with Keyturner States command identifier to USDIO

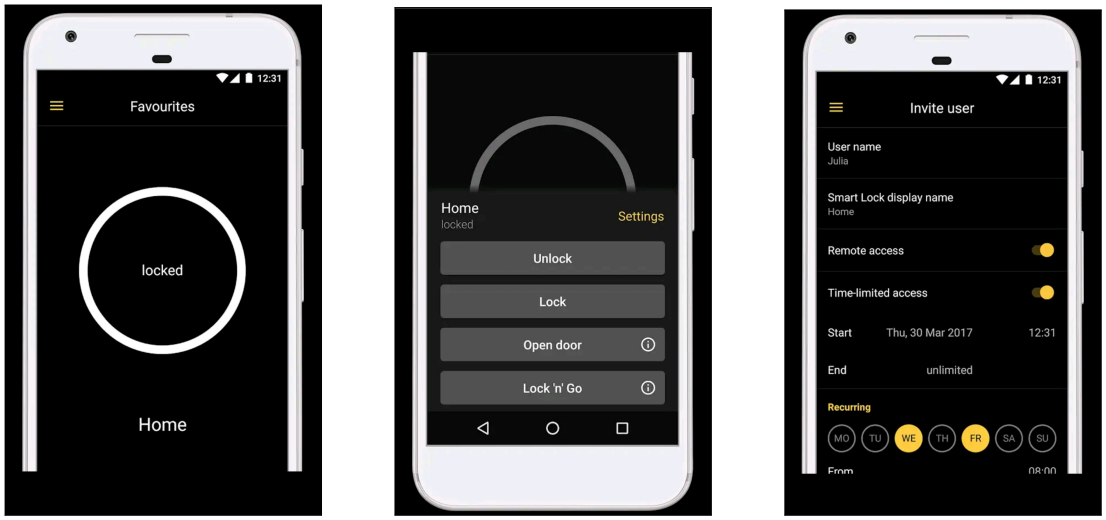
  - Unencrypted: 02000000001000C00418D
  - Encrypted:
    37917F1AF31EC5940705F34D1E5550607D5B2F9FE7D496B602000000
    1A00670D12492600436632E8D927A33FE84E782A9594D39157D065E
  - CL sends encrypted message

- Smart-Lock (SL) sends Keyturner States command via multiple indications on USDIO
  - CL receives 90B0757CFED0243017EAF5E089F8583B9839D61B
  - CL receives 050924D2020000002700B13938B67121B6D528E7
  - CL receives DE206B0D7C5A94587A471B33EBFB012CED8F1261
  - CL receives 135566ED756E3910B5
  - Decrypted: 020100E007030708 0F1E3C0000200A
    * Nuki state: 02
    * Lock state: 01
    * Lock trigger: 00
    * Time: 2016-03-07 08:15:30
    * Offset: 60
    * Battery critical: false

As we can see, from the above example, the communication flow goes through different key exchanges. In fact, Nuki has created a custom encryption protocol to have complete control over the security of its devices. A focus on Bluetooth communication security and an in-depth description of the protocol choices will be carried out in section 4.4.

4.3 Nuki application

The mobile application is an essential component within the Nuki system, because it allows the user to access the available features and manage the smart-lock configuration. In order to start using the application, it is needed, of course, an identity for the system; as such, the user must log in or sign up with a Nuki account. After that, it is possible to configure the smart-lock for the first time, that gives the configuring user a key with special permissions, which grants the ability to modify all the settings, see the activity log, and all the status information. Moreover, the application gives the possibility to invite new users to only the owner account. It is possible to delegate these permissions to another account.
When an invitation has been created (fig. 4.4c), the invited receives an email asking you to install the app and create an account. Then, in the email, there is a one-time link that redirects to the application and launches the key creation process. The first time the key is used with the smart-lock, the device will be paired to the smartphone. When the new user owns



(a) Overview page. **(b)** Available actions on smart-lock. **(c)** Key creation form.

Figure 4.4: Screenshots from the Nuki mobile application.

the access permission, it is possible to see the current status of the smart-lock (lock, unlock, uncalibrated, etc.). (fig. 4.4a) or, with a swipe up, to use the actions available (fig. 4.4b).

The application is available for both Android and iOS and even for smartwatch OSs. The developers make available an URL scheme to interact with it from other applications. Through the OS intent system, it is possible to:

- jump to a specific smart-lock by name or ID and opens the available actions, as in figure 4.4b;
- open the invite code page with a specific invite code (needed by the invitation link in the email);
- delete a specific smart-lock by id, from the list.

Another important application in the Nuki ecosystem is the Web portal, called Nuki Web. This is available only if the smart-lock is connected to the Internet with a bridge. From there it is possible to manage the device, see the logs, the status, and manage the users. Moreover, from here it is possible to connect with a short-term rental account (a vacation rental management system) and manage the integration with third party options. In the section ”API”, there is the possibility to retrieve the client ID, assigned to the account, or to generate API token, in order to authenticate with the Nuki REST API.

4.4 Security and encryption

One of the strengths of the Nuki smart-lock is the focus on security with which they have designed their products. To ensure a high degree of security, they had designed a protocol over Bluetooth connections. In particular, they ensure strong end-to-end encryption and resistance to ”replay attacks”[19].

4.4.1 End-to-end encryption

Every Nuki application uses its own key to communicate with the smart-lock. Nobody else than your device and your mobile application knows that key. In particular, every message exchanged that runs through the Bluetooth channel is not readable without that key. Encryption is performed with a state-of-the-art algorithm, which is a combination between salsa20[20] and poly1305[21], thanks to the NaCl cryptography utility library. Even in a configuration with a bridge all the data are safe, because they are all encrypted on the source and decrypted on the receiver.
In order to secure the key use for decryption the protocol uses a so called Diffie-Hellman Key Exchange mechanism. This allows the device and the application to create a secret key, known by both sides, without exchanging the key itself.
Finally, the key are saved on both devices; however, as we mentioned in section 3.3.3, the XML of the application, where the key is probably stored, is accessible if the smartphone is rooted or jailbreaked.

4.4.2 The challenge on response

In order to avoid the ”replay attack”, the protocol implements a challenge on the response of every message. The ”replay attack” consists of a channel sniffer that can record all bytes during Bluetooth communication. If a malicious application resends the exact byte stream to the smart-lock, it possibly can retrieve the same result. In particular, the attack can be conducted as follows:

- a sniffer records the exact bytes the owner application sent to unlock a device;
- those bytes are sent again by a malicious entity;
- the smart-lock recognizes those bytes as valid and repeats the action.

However, the Nuki smart-lock implements a challenge to secure this malicious behavior. Before the application can send any requests, it receives from the smart-lock a 32 byte random number, which must be sent again with the request. If another command with the same number is received, the smart-lock recognizes the request as invalid and it answers to the malicious application with an unsuccessful response.[22]

4.5 Nuki web API and Webhooks

Nuki offers REST API to developers who want to integrate their services with the smart-lock features. In order to work, the smart-lock must be connected to internet and accessible remotely. As such, it must be in the bridged configuration.
Moreover, the system needs to activate the Nuki web API from the dedicated portal. Once done, it is possible to use the client ID and apply the client secret for OAuth2 authentication. Furthermore, there is the possibility of creating and deleting API tokens.

4.5.1 Security

The API transmits all commands directly through a permanent HTTPS/TLS connection to the corresponding Nuki bridge, which transmits them via Bluetooth to the smart-lock for execution[23].
When the Nuki Web API is initialized for the first time, the system creates a server-stored Nuki Web Authentication Key, which gives you the ability to execute commands on all devices associated with the account and connected to the Internet.
Thanks to its own Authentication Key, Nuki Web acts independently from the other clients (e.g. Nuki iOS or Android App).
Despite the fact that the communication between the Nuki server and the bridge is encrypted end-to-end, the one between the APIs and the external client relies only on TLS security.

4.5.2 Authentication

The Nuki REST APIs use bearer authentication, which is an HTTP authentication scheme that involves the usage of a security token called bearer. To avoid receiving an unauthorized response, the commands to the API must contain this token to be performed as in the following example:

Listing 4.1: Simple curl command towards the Nuki web API with bearer token.

```
curl -X GET --header 'Accept: application/json'
--header 'Authorization: Bearer c2c0981ffcab78eecd13c8b7ae9fdec4706045bdbb17b1ef06a335
    b832f36641322c5c3357b7fe47'
'https://api.nuki.io/smartlock'
```

As such, the client must obtain the token to perform the request to the REST API. There are different methods to retrieve this information.

API tokens

From the Nuki Web portal it is possible to initialize API integration services. Moreover, it can be generated as a permanent API token, with different levels of permission. This is particularly useful if a smart-lock owner wants to build its own application or if a developer wants to have a fast way to access the API.
Once the key has been generated, it must be saved, because it only appears once. Additionally, from the portal, it is possible to edit permissions or delete a target API token.

OAuth2 authentication

For third-party applications that want to integrate APIs, it is suggested to use the OAuth2 authentication flow. This is particularly useful when there are three actors, the owner of the smart-lock, the third-party client application and the REST API, which must authenticate to each other with a central entity. In particular, the user delegates the authentication to a service that is hosting the user account (Nuki Web), authorizing a third-party application to access the user account permissions.
To achieve that, the external application must be recognized as trustworthy by the REST API. In fact, the developer of the application must apply to receive a client secret from the Nuki web portal, which identify the client. In the Nuki case, this is called "Advanced Nuki API integration", which gives the account the possibility to use the webhooks.
However, in general, there are two different types of OAuth2 authentication flows: the "code flow" and the "implicit". In the "implicit" method, there is no need for the client secret, but the bearer token received at the end of the process lasts only for an hour. Transactions proceed as follows.

- The client of the third-party application starts the authentication request, stating its client ID (a unique identifier), the permissions that want to exploit and finally a redirect URL, which represents an endpoint of the application which can be called.

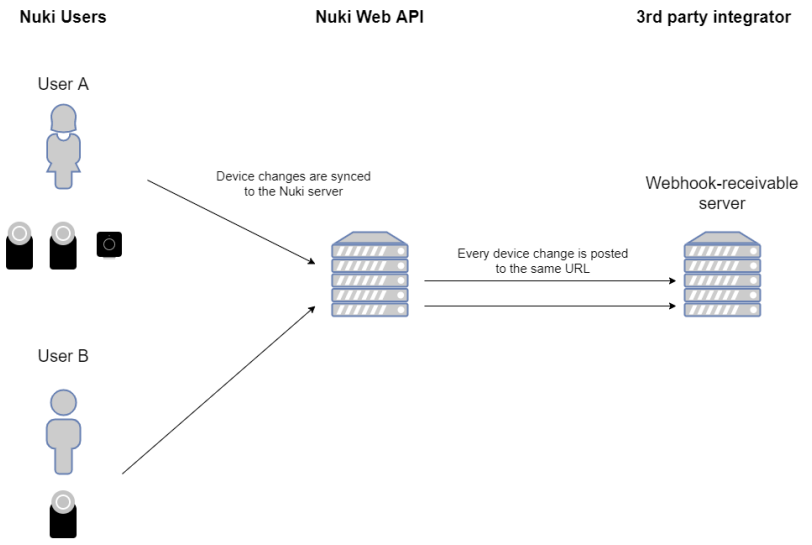
- The OAuth2 service opens the authentication page (in our case, Nuki Web authentication) for the user, declaring the permission he is giving to the external application.
- If the authentication is successful, the server calls the redirect URL provided in the first request with the token as parameter.
- The application then has a bearer token that can be used for one hour.

The ”code flow” method gives, as output, a valid API token and a refresh token, which can be used after the other token is expired in order to retrieve a new valid token. However, the process required a few more steps and the client secret to be completed.

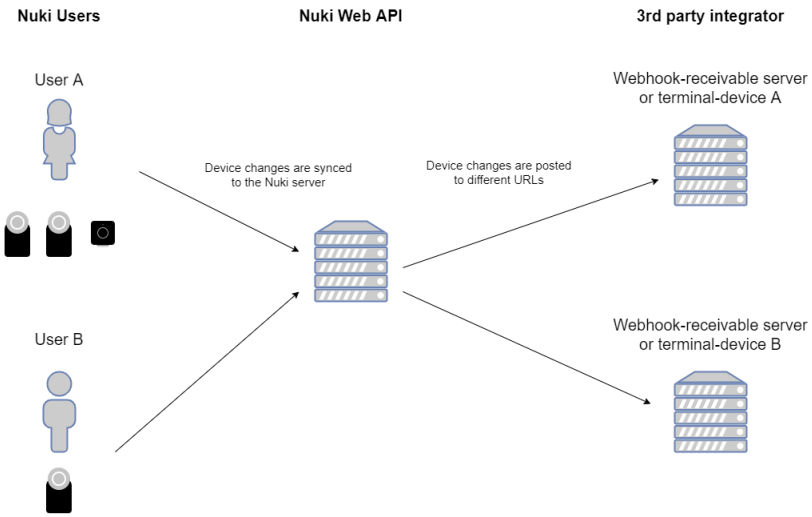
- The first two steps remain the same as in the implicit method, of course, specifying the type of transaction it is requesting.
- In this case, the server returns a code, which is not the API token, and it has to be used to perform another request, adding client secret as a parameter.
- The response to the second request contains the actual bearer token, with its expiration time, and the refresh token.

4.5.3 Advanced API integration and webhooks

The ”advanced API integration” is a Nuki program, to which a user must be applied to retrieve a client secret. Furthermore, it will give access to additional API endpoints to manage webhooks. Webhooks are an event system that is used to asynchronously inform about any changes in the device in a timely manner[24]. Even in this case, there are two possible workflows.
The central webhook workflow forwards all events to a single URL endpoint (fig. 4.5a), whereas the decentralized one can manage more than one URL (fig. 4.5b).



(a) Central webhook workflow.



(b) Decentral webhook workflow.

Figure 4.5: Two possible workflows for correctly manage webhooks.

4.6 Smart vacation rental Nuki solution

Nuki has already created a solution to the problem highlighted in the introduction of this document. A vacation rental management system host can install the smart-lock and the bridge in its property and then link the Nuki web account to whatever platform account they are using as a vacation rental management system. The Nuki is able to check the reservation

and automatically send the invitation code to the guests. After their stay, the invitation code is disabled.
This brings more than one advantage: check-in and check-out can be done 24/7, without human interaction; the service providers and cleaning staff can access and be issued at any time and from anywhere; and finally, the host has remote full control of accesses.
In the project, the goal is to emulate the same behaviour implemented by Nuki, but including more than one provider, giving an application more centered on the host needs.

5 Kerbero

This chapter outlines design and implementation choices about the project in which I had worked during my internship with Kuama Srl. The project is called Kerbero and it is a new application which provides one-point access for multiple smart-lock devices. In addition, it is strongly oriented toward the check-in and check-out of users of the vacation rental system. The application aims to integrate with these platforms, in order to provide the possibility to create a temporary key for the guests, which expires at the end of their stay. Moreover, the application should be compliant with the highest number of open APIs for smart-locks. In order to do that, an architecture must be implemented that can interface with multiple plugins for each of the devices that the goal is to support.
Continuing, we discuss the initial requirement analysis; the feasibility study and the first tests on the technologies chosen; a discussion on the design choices about how to model the concept of keys for the guest, how to manage the identity of an host in the system and how to authenticate.

5.1 Requirements analysis

The initial phase of the project involves many meetings with the committer. Being Kerbero an internal project, the committer was my tutor, with whom we defined all the requirements and the use cases of the application. As such, all the use cases and requirements are retrieved,

updated and documented during the whole project, thanks to the continuous interaction between developers and committer. In this section, the whole requirement analysis is not reported, but only the use cases and scenarios useful to understand the choices we made during the design of the architecture and the choice of technologies are reported. As such, this analysis results being more discursive and less formal.

5.1.1 Actors

First of all, we defined the roles involved in the use of the application. During the analysis, we found that the application is strongly oriented towards the host "actor". The idea composed around the project foresees that the host is the actual user which, once logged in, can have access to the application features. However, we later realized that guests also need a way to interact. In fact, they have to receive an actual virtual key with which they can lock or unlock the smart-lock.
Finally, we realize that only the host needs an identification within the system. As such, we decided to have the ease approach for the guests, which foresees that they should not have to register any account.

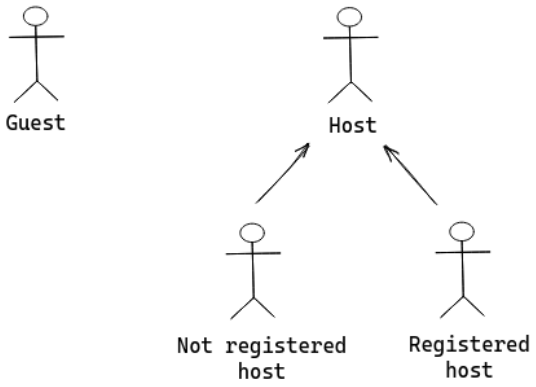


Figure 5.1: Actors scheme from the requirements analysis.

In light of the previous considerations, the identified actors are the following (fig. 5.1):

- the guest, which does only own a key and can open and close the door;
- the not authenticated host, which has the possibility to register an account or log with an already existing one;
- the authenticated host, indeed, has all the permissions, as such he can have access to all the features of the application.

5.1.2 Use cases

In order to analyze the requirements defined by the committer, it is best practice to produce use cases. Use cases are usually represented by UML diagrams and define the interaction of the actors with the system.
The definition of use cases was made top-down and, as such, we first define the main features and then refine them step by step.

General

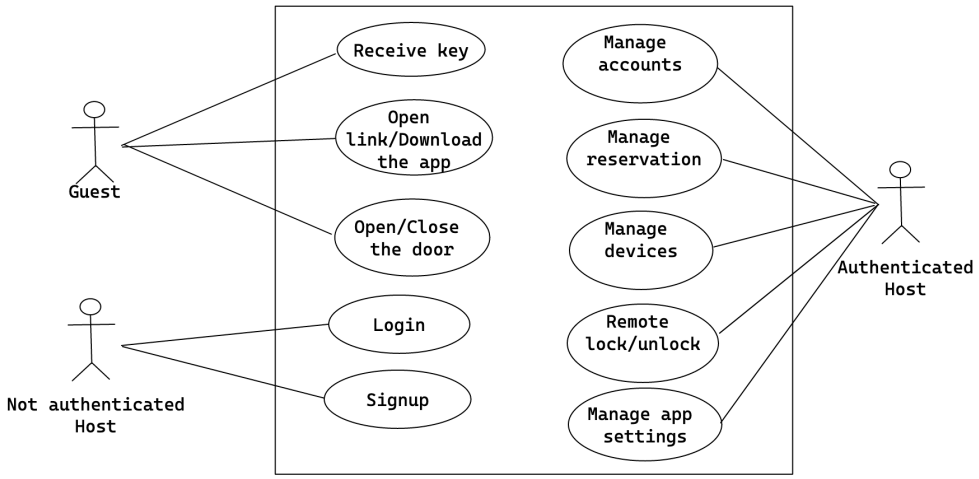


Figure 5.2: General use case.

Starting from the highest level, the general use case defines the main features that we identify to be implemented in the application, giving an abstract idea of what the application must do.
First of all, there are all the features related to authentication and identity. In particular, a not authenticated host can sign up and login, while an authenticated one can sign out.
After that, a guest is able to receive a key and through a link opens a page from which he can lock and unlock the smart-lock.
Furthermore, an authenticated host can access the main feature of the application, which are:

- manage the smart-lock provider accounts (e.g. add/remove a Nuki account);

- manage the reservations, manually add guests details, arrival and departure time, in order to create the correct keys;
- manage the devices, downloaded from the linked accounts;
- remote lock and unlock the smart-locks;
- manage the app settings.

Authentication

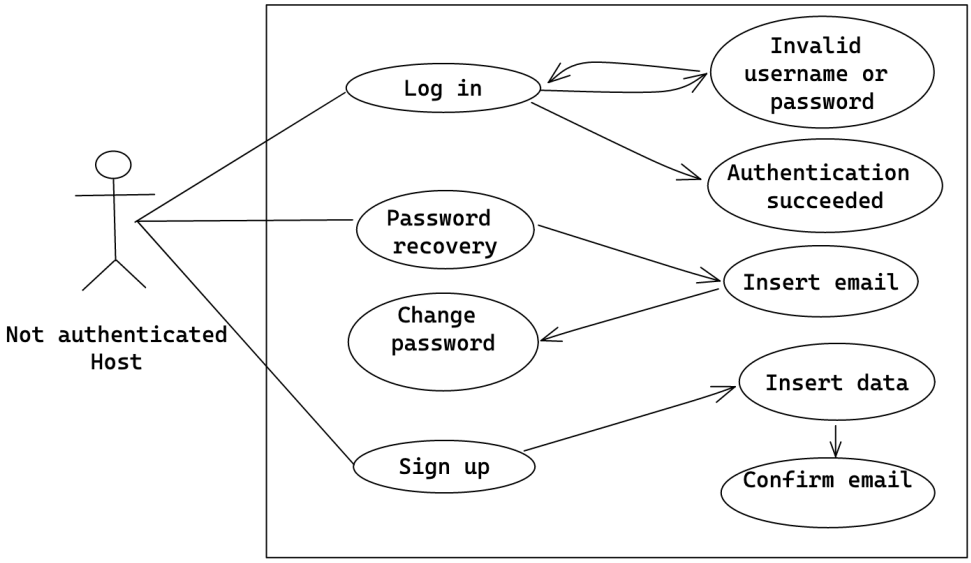


Figure 5.3: Authentication use case.

The authentication is managed with a common log in and sign up system. Moreover, since account management (known as sign-up) is entrusted to the user, there is the possibility to recover the account if the user has forgotten the credentials.

Smart-lock accounts management

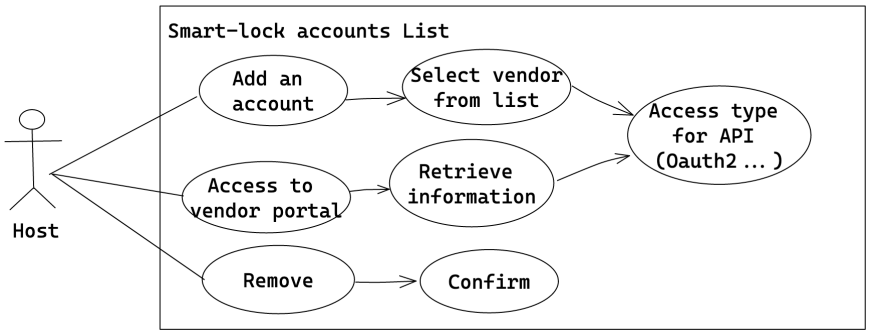


Figure 5.4: Smart-lock accounts management use case.

The authenticated host must be able to add a new smart-lock provider account. This integration with the external account should synchronize the information. He can remove a selected account as well.

Devices management

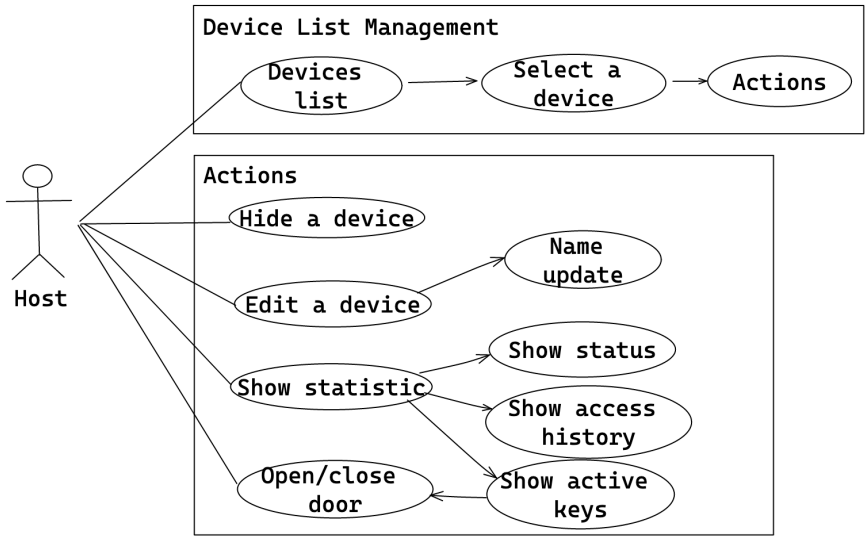


Figure 5.5: User devices use case.

We decided to give the host the ability to see the list of devices synchronized with the provider accounts. Moreover, it is possible to lock and unlock, edit and show a status page for each of

them.

Keys management

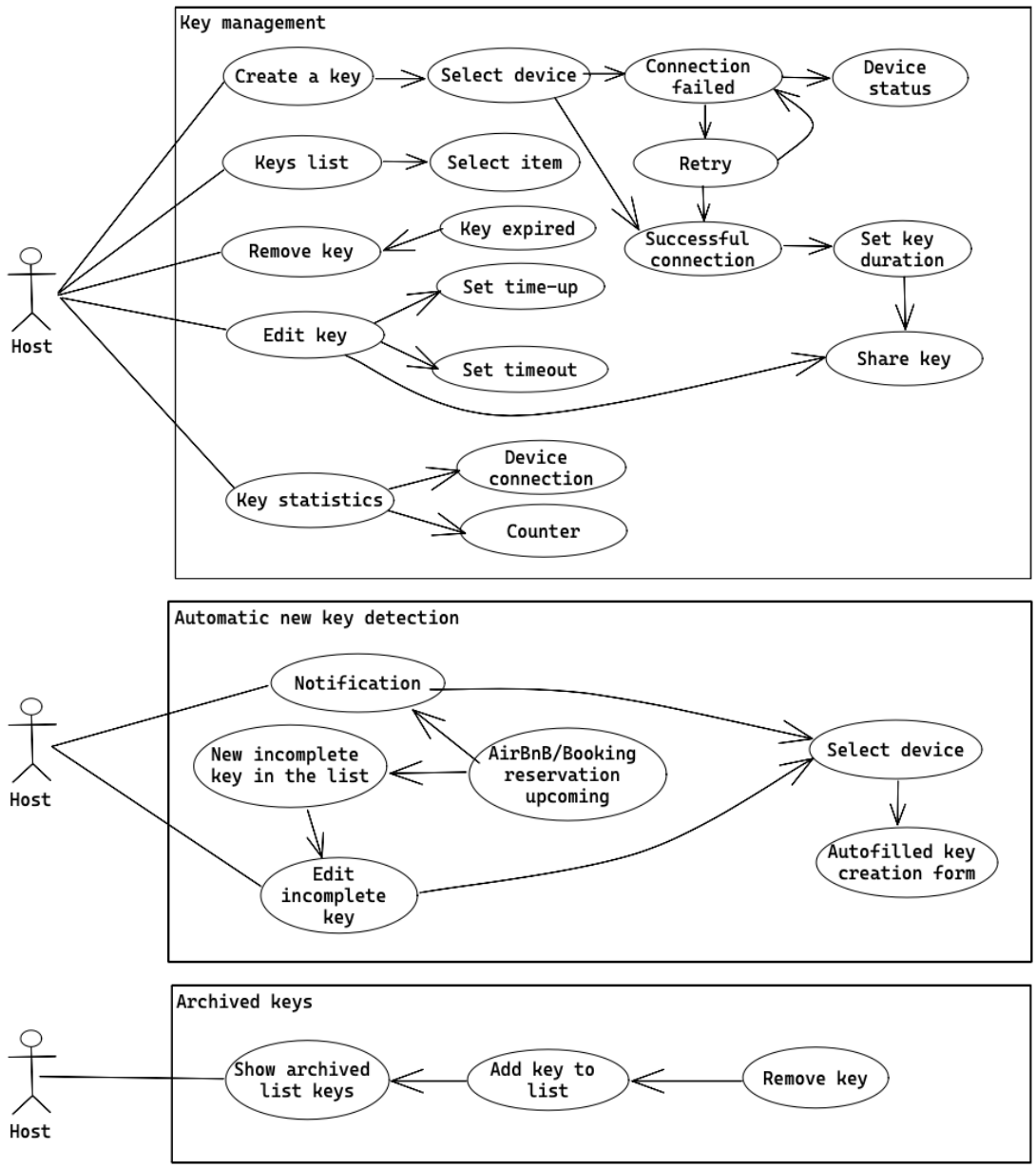


Figure 5.6: Key use case, with automatic creation from reservation and key archive.

The management of virtual keys is the core feature of the application. The host should be able to create a key from scratch following this process:

- select the smart-lock for which the key will be created;
- if the connection with the device can be established the process continues, otherwise it can be retry or abort;
- then the host can set the starting and expiring validity date;
- finally, it is possible to insert the guest emails and send the invitation with the key link.

From the list of keys, it is possible to select an item and have access to the actions, which are the editing and the status display. Automatic new key detection is the system that allows the host to automatically create keys starting from a confirmed reservation. The reservation can be detected by a webhook and is reported to the host.

Vacation rental management system links

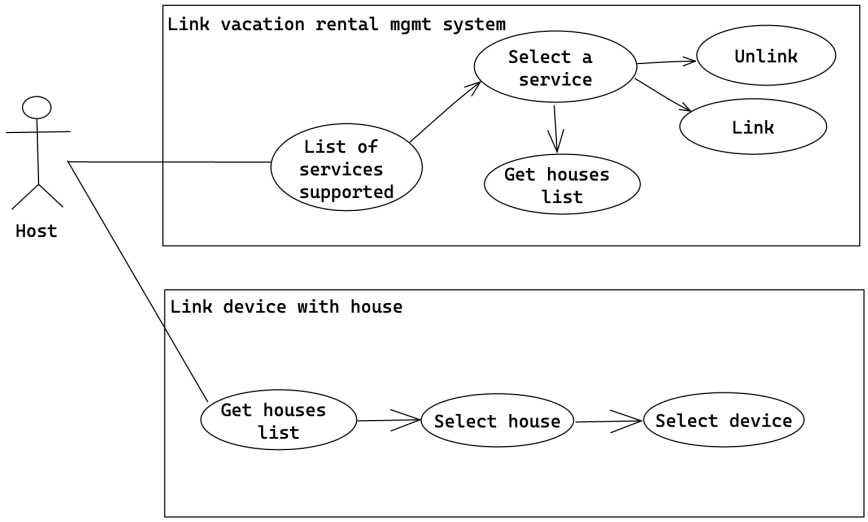


Figure 5.7: Vacation rental management system use case.

Finally, to enable automatic key generation, the system must use the webhook of the vacation rental system management. As such, as in the smart-lock case, there is the possibility of linking and unlinking an external account, through the related authentication mechanism. This is actually an optional requirement; as such, the system can be designed without those particular features.

5.1.3 Other diagrams

During the analysis, some other diagrams were created, which can now be useful to understand how many parts of the architecture work.

Create Key

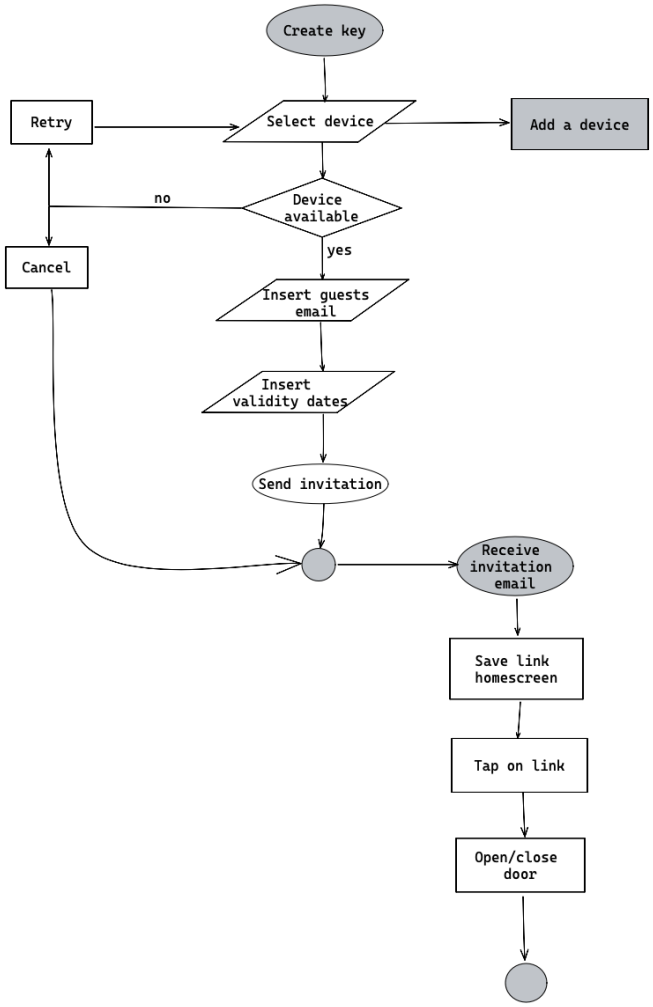


Figure 5.8: Key creation flow diagram.

Key creation is a core feature of the application; therefore, we focus on the details of its steps. As we can see in the flow diagram in figure 5.8 there is emphase on the initial check of the availability of the device. The importance of using a reachable smart-lock is fundamental,

as such the host is always notified about the status of their devices. In particular, we thought that a user should not have the possibility of creating a key on a non-accessible device.

Add a Nuki account

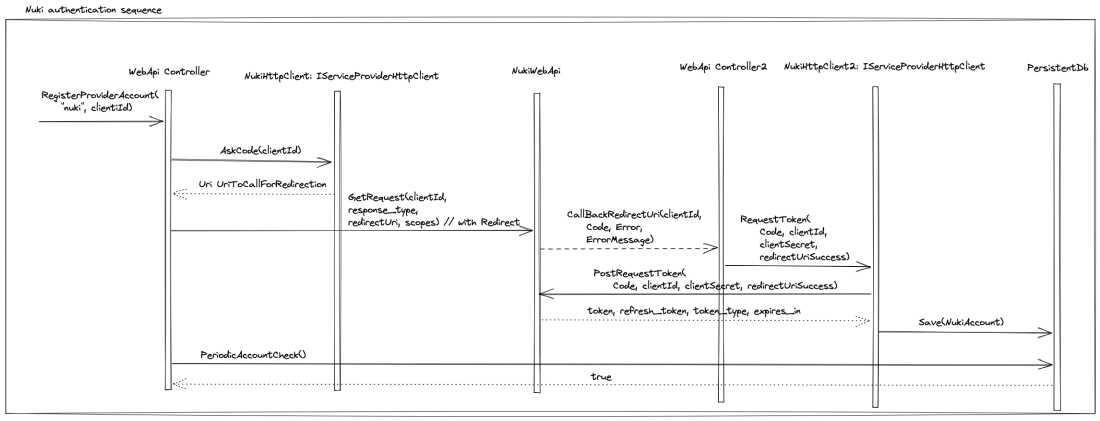


Figure 5.9: Link a Nuki account sequence diagram.

The link of a Nuki account that follows the OAuth2 authentication process was trivial during the design of the application. In particular, during the feasibility study, the initial test and the Proof of Concept implementation, as such we made a sequence diagram to clear up the entire flow. This process will be better explained in the next section 5.3.5.

Entities diagram

Once determined the use cases and requirements, we proceeded with defining the entities involved in the application. An entity is any identifiable and separate object that is significant for the application. The diagram, in figure 5.10, defines which are the entities, their attribute and how they are related to each other.
This entity diagram refers only to the host, which is identified as *user*. As we can see, a user can own more than one *ProviderAccount* or *RentProviderAccount*, which are abstractions of an external account, respectively, of a smart lock provider and of a vacation rental management system. Moreover, both of these entities are considered as interfaces with zero or one relationship between the main entity and its specifications.

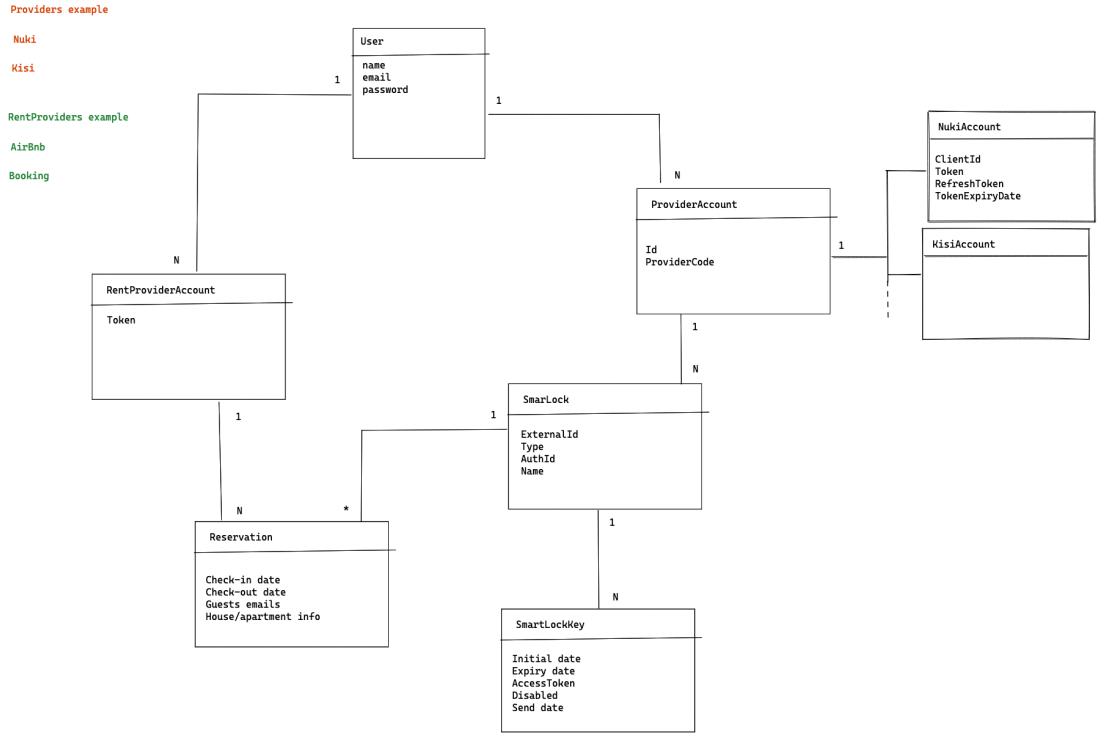


Figure 5.10: The Kerbero entity-relationship diagram.

5.2 Feasibility study

While performing the requirement analysis, we conducted a feasibility study to check whether our assumptions were actually implementable with existing technologies.
The focus was, of course, on the state-of-the-art technologies involving smart-locks, which are already and extensively treated in the previous chapters. However, it was important searching and selecting the smart-lock solution, which could actually satisfy the use cases we had defined.
The most important feature a product must have is, of course, the availability of the public APIs. The integration of Kerbero with manufacturer software and systems was an essential precondition that we searched over all the alternatives taken into account.
Moreover, we searched for the characteristic of the product, the independence of the main operation from the manufacturer application and an integration with the already existing invitation system, which is translated in our project as the creation of virtual keys.
In light of this, a wide search on internet of the products was performed, which better fit the model we traced. In particular, we classified products according to the characteristic of our

interest, as shown in the table 5.1.

Host device Bluetooth pairing without app	C1
System invitation acceptance without app	C2
Possibility to Lock/Unlock without app	C3
Must have to apply for more integration	C4
Already offers an integration with a vacation rental management service	C5

	Nuki	Kisi	Salto	Latch	Operto	Yale/August	Brivo
C1	x	x	x	?	?	x**	x
C2	x	✓***	✓	?	?	✓	✓
C3	✓	✓	✓	?	?	✓	✓
C4	✓	✓	?	✓*	✓*	?	?
C5	✓	✓	x	x	✓	x	x

Legend:

✓	Feature available
x	Feature not available
?	No access to this information
*	no access without
**	wi-fi only - need a code
***	with white labelling

Table 5.1: Comparison between product of different manufacturer, related to some characteristics of interest.

The choice of the first device to integrate in our system was relapse in the Nuki smart lock, the description of which is already widely covered in the chapter 4.

5.2.1 The vacation rental management integration

The integration with the vacation rental management systems are intentionally excluded from the mandatory requirements of the project, because of the difficulties of accessing their APIs. In particular, Airbnb and Booking APIs were considered the most used platform worldwide. We found that their APIs was selectively open, and as such if a developer wants to use them, he must perform a request to the vacation rental management system support service.
Moreover, it is a common thought that Airbnb is very selective with companies asking for access to their APIs. As such, when a company wants to apply, it is suggested that they already

have a working application to give as proof. * Those are the reasons we decide to develop the application modularly, from the core features to the plugins for each of the providers. However, the main module of the application was only prepared to work without adding any integration with a vacation rental management system.

5.3 Software design

In this section, I would like to outline most of the design choices we decided to adopt to implement the previous detected features. For each of the following, a PoC was developed to prove the validity of the assumption and to test the features in a real world scenario.

5.3.1 Smart-lock keys design

An essential decision during the design of the application was how to properly model the concept of virtual key. In the real world, keys are the only way to lock/unlock a door. We expect that this behavior remains invariant with the virtual model, as well. However, a virtual key has an important upcoming that the traditional key does not have: the possibility to be invalidate immediately. This benefit brings many others, such as the possibility to temporize the key or to give access to a person once and then revoke the permission.
Another important characteristic of the Kerbero keys is the ease of use of guests. As such, the requirements we fixed for the keys were:

- no mobile application to open and close the door;
- availability for mobile;
- security relying on a password.

Therefore, we decide that the key must be a public page accessible with a link and available only through the insertion of a keyword as a password, randomly generated by the application. The idea is that a guest receives a link to the virtual key page and a random password by email (fig. 5.11). The guest is suggested to save the key link on the smartphone home and

*

> "At this time, we are not accepting new access requests for our API. Our global team of partner managers will reach out to prospective partners based on the supply opportunity your business represents, the strength of your technology, and the ability to support our shared customers."
>
> (From the Airbnb site: https://www.airbnb.com/partner)

to keep the password safe and private.
This approach does not reach the security level of using a dedicated application, but is much easier to use for the end user, who does not have to download an application and create a new account. Moreover, the security level of the provider application rely on the use of a Bluetooth pairing between the devices, which assures that both of them are distinct entities that recognize each other. If the request is coming from the internet, this benefit is lost; the request is signed by the client ID of the application which is sending it, but the identity of the guest can not be tracked, except by the client itself. However, the smart-lock does not recognize information of this type coming from the client, as such that becomes useless.

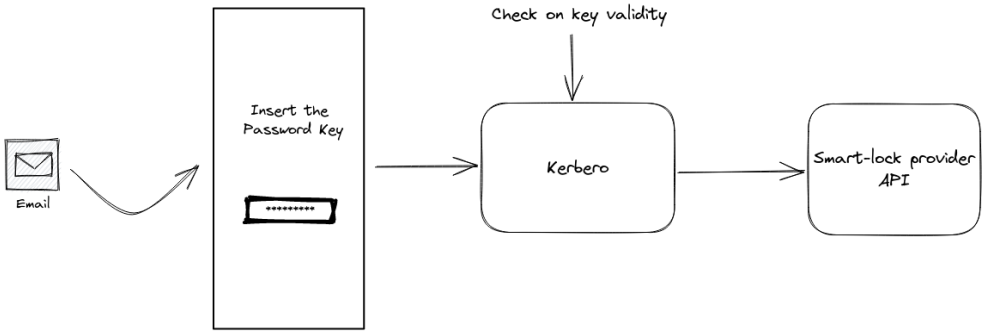


Figure 5.11: Key functional process.

Moreover, the non-usage of the official smart-lock application brings to others comedowns. Most of the features available through Bluetooth, such as auto-unlock or geo-fencing, are not usable through APIs. However, it is important to note that these features are mostly used in a smart house context, which is not our purpose. The design we proposed is guided by the assumption that an Airbnb host already knows what are the risks of giving the access to a stranger, as such, we premised that in the overall process mutual trust is involved.

5.3.2 Smart-lock management

Since the application is designed to operate with all the smart-lock through their APIs, the Kerbero specific model must be more generic than possible. In Kerbero the smart-lock data is not persistent, the information are fetched with the external API, in order to avoid out of sync and inconsistency. As such, most of the work of the plugin is to translate the information coming from the external APIs into a readable object for the Kerbero system.
About the "writing" operations, such as the open and closed, keeping the object generic becomes more complicated. The process divides into various steps, as can be seen in figure

5.12. The logged in user has saved different credentials for each of the provider supported by Kerbero. As such, the request must supply a *Provider Identifier*, which serves the Kerbero component to switch between the right credentials and the repository to select.

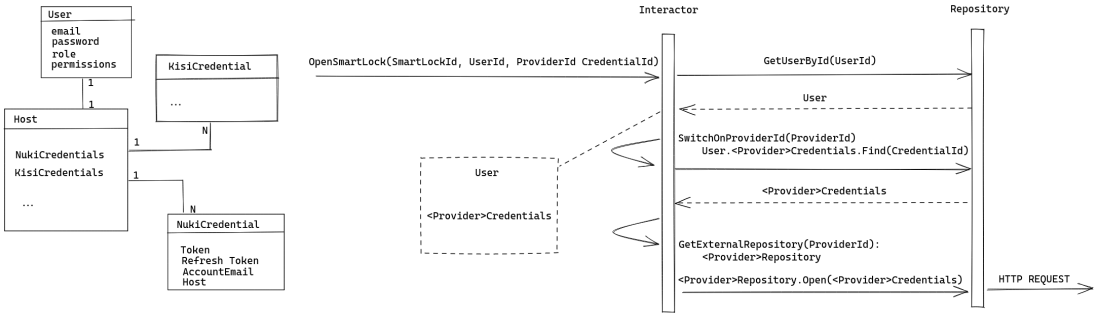


Figure 5.12: A schema for opening a Kerbero smart-lock, an interactor (a method satisfying a use case) communicate con repositories on the data layer.

5.3.3 Application identity management

In the section 5.1.1, we had given a hint of the authentication model through the specification of the actors involved. In particular, it was revealed that the host is the only role which must have a distinguished identity. As is said before about the keys, the guests do not need to identify themselves into the system, because they use keys which are not associated with any identity.
Therefore, the host needs an identity system that he can authenticate and, as a consequence, have access to the features of the application. The design of this type of functionality generates a lot of discussion and the choices we make here have involved the overall system architecture. The identity system we were looking for was not complicated; however, it needs the following properties:

- one level permission for the user, which means that the application does not need to have different access level or locked features only for a specific type of user. This is a choice tight with the concept of ease of use outline in the requirements.;
- persistence of information related to the user;
- scalability oriented;
- email account confirmation.

The framework which grants to us this specification and the following authentication system is ASP.NET core identity, which will be described later in section 5.4.5.

5.3.4 Cookies authentication

The Kerbero client runs on a Web application which provides a Graphical User Interface (GUI) for the host. Moreover, there is a sever side application which performs the core actions. To maintain the authentication session, it was chosen to use browser cookies. They are managed in order to provide a user state both on the application side; therefore, providing a session for the client and a valid user identity on the server.
The operations on the cookies are simple and robust: the client sent the log in information to the server, the server checked the information, and if they are correct, it creates a cookie which is attached to the response of the authentication request. The browser that receives the cookie in the header of the HTTP message saves the content in its immutable cache storage, called the cookie jar. As such, every time the client does a request, the cookie is attached, the server receiving the message checks the validity and the content of it.

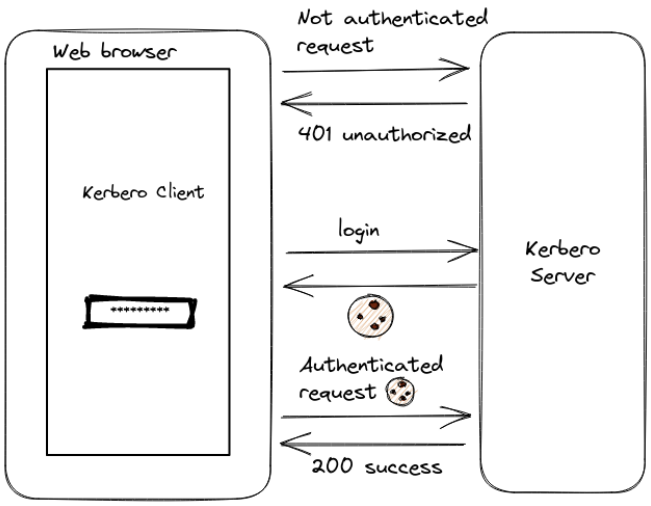


Figure 5.13: The cookie session and authentication management.

Cookies specification

In order to better understand the reason for the choice of cookie authentication, it is important to know their specifications.
Cookies are born as a way to pass a small piece of information from the client to the server and vice versa. As such, the purpose is to create a shared state between the two actors over the HTTP protocol.

By default, the cookie was not built for security; it guarantees neither confidentiality nor integrity of the transferred data. However, it is worth mentioning two attributes of the cookie: *Secure* and *HttpOnly*. *Secure* attribute limits the use of a cookie to only secure channels. As such, the client attaches the cookie only when the request is over TLS. Note that, while this protects the confidentiality of the cookie, it does not protect its integrity if an attacker sends the request from a secure site. The second attribute, *HttpOnly*, limits the cookie to be used only on HTTP request, which means that it cannot be modified by the JavaScript of the browser and, as such, it can only be writable on the server side[25]. Thanks to this mechanism, the cookies can only be saved in the browser, as such the ones created by the client and attached to an HTTP request are ignored by the server when received. As a consequence, the unique flow available to manage them starts from the server, which attaches the cookies he needs on the HTTP request and the browser receiving them has to store in the cache. Then, it can resend their content with the following request. As such, the cookie jar, if correctly managed from the server, is read-only on the client side. As a result, the server is considered a trustworthy entity that can manage the state of the browser.
Therefore, the client is resistant to many attacks related to cookies, in which malicious code writes a copy of an existing cookie, to emulate authentication, for example. If the cookie is properly configured, those attacks cannot be launched from the client side, by injecting JavaScript code. This is avoided by the fact that, due to the *HttpOnly* attribute, JavaScript scripts can access read-only cookies.
Moreover, the update process of a cookie grants a low level of integrity. In fact, all cookies are identified by a key that provides uniqueness. If a new cookie with the same key and different content is received by the browser, he must update the existing one with the new information.
An additional security layer is granted by the *SameSite* attribute. The same-site cookies are a relatively new HTTP specification, which provides to the client a way to decide which cookies can be sent in the request to the server. *SameSite* attribute was specified to avoid cross-site request forgery attacks, which are particularly dangerous to security[26]. This type of attack affects authentication; in fact, if an attacker manages to provide a stolen cookie, he can set the cookie with a malicious application and call the real server. As such, the real server recognizes the cookie as valid, and it provides a valid response. The same-site cookie prevents that with an URL restriction, which limits the cookies to the origin site with a URL domain check. *SameSite* can be set with three different values:

- *None*, which disables the whole protection mechanism;

- *Strict*, which defines that the client can send the cookie only to the origin site;
- *Lax*, which is similar to strict, but the client can send the cookie even if he is on another domain, reached after a redirection from the origin site.

Why cookies over Bearer token

As we have seen so far, cookies are strongly bound to browser technology. As such, they cannot be used for other types of client, like a mobile or desktop application. Therefore, a bearer token solution, maybe combined with a OAuth2 authentication flow, seems to be a better solution in the context of an Web API. Moreover, bearer tokens are generally more secure than cookies, since they are not stored on the client machine and are less vulnerable to tampering. However, they can be more difficult to use since the server must keep track of the tokens and implement additional security measures to prevent unauthorized access. Furthermore, cookies technology, with *HttpOnly*, *SameSite* and *Secure* attributes, improves a lot in recent years, reaching almost the same security level of the bearer token.
The two benefits we recovered from using the cookies were:

- they are lightweight and more informative;
- completely managed by the browser and the server side framework;
- they are valid for the entire session, until the browser wipes the cache.

It is important to note that most complex applications usually combine these two methods. The reason is that a large application has many different clients. If the client is a web browser it is commonly used the cookies solution; however, if the client is a dedicated mobile or desktop app, the bearer token is the only way. Therefore, Kerbero is now implementing only cookie authentication, but is already prepared for the future addition of another authentication method.

5.3.5 OAuth2 authentication flow management

During the analysis, we noticed that access to the smart-lock service REST APIs is almost always protected with an OAuth2 authorization flow. OAuth2 is an authorization framework that enables external applications to obtain access to a user account linked to an HTTP service. It is usually used when a service is integrating an external one; as such, the first one

needs to ask the permission to the user of the second. As such, OAuth2 works by delegating user authentication to the service that hosts the user account, in order to give the access permission to the application requesting the integration.
The roles involved in the process are:

- the resource owner, which is the entity that is responsible of the user account;
- the client is the application which requires the resource;
- the resource server, which is the place where is located the data the client is looking for;
- the authorization server, which is the authority managing the authentication and the permission access to the resource.

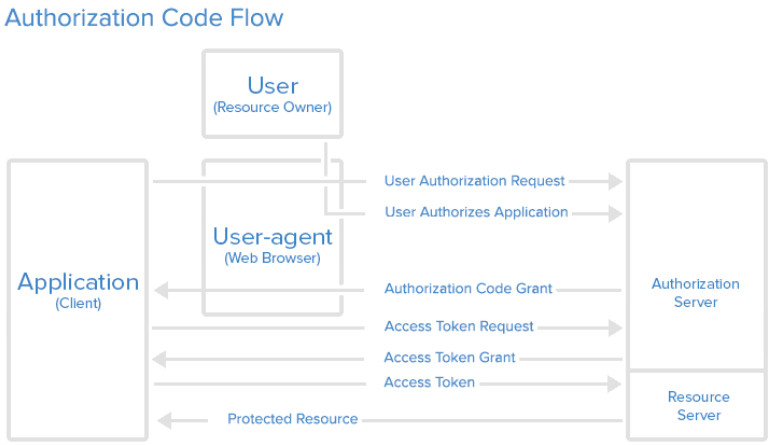


Figure 5.14: OAuth2 protocol flow.

As we can see in the figure 5.14 the components interact with each other, in order to provide the client with the possibility of reading or writing the protected resource, in the form of an access token. [27]
In order to represent a valid application, the client must register with the service. The common way is to provide the client with a *client identifier* and a *client secret*, which must be exchanged during the authorization phase.
From the client's perspective, the authorization process requires two phases.

- First of all, the client must ask the user permission to use the information in his account. In order to do so, the resource owner must expose an authentication service

dedicated to the OAuth2 service. The client user is commonly redirected to the authentication service page of the resource owner, where he must log in and accept the read or write permission request to the resource. Once the user completes this procedure successfully, the authorization server returns a code which is not yet the access token.

- In the second phase, the client must request the access token to the authorization server. The request is performed using the code given in the previous phase, which is temporary and usually single-use. The response of the authorization server contains the access token, which is the bearer token, and the refresh token, which is used, indeed, to update the access token when it expires.

One of the most difficult challenges faced during the design of the application is the integration of the OAuth2 authentication flow with our authorization system. The problem relies on the end of the first phase, when the authorization server calls the client with a callback to provide the code. The endpoint exposed by the client must be anonymous, that is the server can be not authenticated to call it. This choice is mandatory since the authorization server does not know anything about the client authentication system. However, when the client endpoint computes the information, it does not have clues about the identity of the user who started the OAuth2 flow.
In this case, the cookie authentication flow is handy, in particular, the *SameSite* attribute *Lax*. This attribute solves the problems of applying a less rigid same-site cookie policy, as such the authentication cookies are sent back from the user-agent (in fig.5.14) to the client, which in this case is Kerbero. As a result, the client can use the cookie to authenticate and retrieve all the information about the user.
It is important to note that this solution can only be implemented with cookies. As such, being only available within the browser, the only possible way to manage OAuth2 authentication seems to be restricted to web applications. However, most of the development framework, like Java/Kotlin Android, Swift, etc., has libraries which can typically manage the flow with a redirect to a web view.

5.3.6 Error management design

HTTP errors

The error management was another challenge during application design. It is important to assume that the application is client-server. The client, as we already mentioned, is a web

application, while the server is a REST API.
The REST API typically responds to client requests with an HTTP response, which contains a JSON object, which is a standard data format, used to define data objects that can be read by different languages and frameworks. Inside the JSON we usually find couple formatted as keys and values. In case of error, the HTTP message contains a specific field called response status code, which contains a number indicating the a standard HTTP error.
Kerbero uses a message-status code approach, as such the client everytime receives an error as response, the latter will contain both a proper status code and a message inside the JSON object. As such, the client has a filter which determines if the error has to be shown as a pop-up or managed internally.

Language exceptions

About the exceptions, the strategy applied was "catch everything". The server side approach is translated in functional, as such every source of error is closed in a try-catch block. Therefore, the catch encloses the return object of the function in a specific result type. This approach has proven to fit particularly well, with the architectural choices we made for the application, treated deeply in section 5.4.1. In general, we can say that a functional approach grants modularity because force the error to be managed in the function scope. Moreover, the object result can contain more than one error, as such it can be more informative of a managed exception. However, the correctness of this approach depends a lot on the implementation and the responsibility relies on the developer. Moreover, many server-side frameworks offer an automatic way to filter and return HTTP status code based on the type of exception returned. However, this approach most of the time does not give the possibility of returning other information, such as a message, which brings this feature out of our scope.

External errors

The last design choices on error management are related to external services. Kerbero is implemented to communicate with external APIs, as such it needs a well design way to handle the errors coming from them. As we said in the previous section, all the exceptions are wrapped in a try-catch and translated into a result object with an error field. This approach is used with external errors too, in fact the client HTTP is designed to throw exception in case of status code different from success (i.e. 2XX). Then, the exception is caught and wrapped in a result object, including a custom error or a list of custom errors. This approach is more

flexible than the exception propagation, because it allows the server to decide which errors have to be managed or ignored.

5.4 Project plan

In this section, we will discuss architectural choices and a brief description of the technologies involved in the project. These choices were particularly crucial during the development of the application. Architecture, in particular, was difficult to design because we had to take into account many different factors. At first sight, from the requirement analysis, it seems that there are not many problems that require a complicated solution. However, Kerbero has a scaling nature; in fact, the goal is to implement more and more plugins in the future, in order to give support to new smart-lock devices. As such, the modularity and, as a consequence, the dependencies question was crucial on the application designing.

5.4.1 The Kerbero architecture

In order to provide a robust split between the user interface and the business logic, Kerbero is divided into a client-server application (fig. 5.15). The reasons for this choice are based on allowing the application to implement different types of client. If, in the future, the resources will allow an implementation of a mobile application, the architecture is studied to give the possibility to be upgraded with only few changes server-side.
The server-side is also designed as modular as possible, dividing layers and business logic in low code dependency sections. Moreover, these modules are organized inside the architecture to have isolated communication with all external resources. This will be discussed in depth in the next section.

The clean architecture

The specification. As we said previously, the application must implement software that is as less dependent as possible. As such, the choice of architecture falls on the Clean architecture.
The Clean architecture, also known by the name Onion architecture, is not a properly defined standard pattern, but a collection of best practice and rules to apply to achieve separation of concerns[28]. This allows the architect to produce a system which is:

- independent of frameworks and existing library;

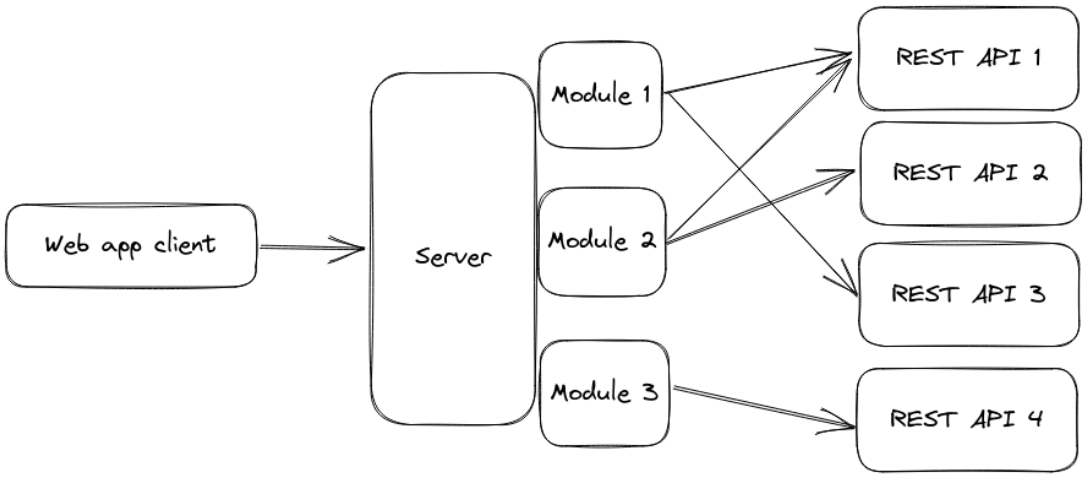


Figure 5.15: High level architecture of Kerbero.

- independent of any external agency, database or UI;
- testable, thanks to the low dependency between the components.

The main rule to follow in Clean architecture is *Dependency rule*, which says that all dependencies of the source code can only point inward. As such, we can see the system as a set of concentric circles in which the inner ones do not know anything about the outer ones.
In the clean architecture the *Entities* play an important role. They are object containing methods, attributes, or data structures that represent system-wide business rules. The latter are the high-level behaviors of the application, which represent the smallest and least dependent unit of logic. Furthermore, entities should not be affected by operational changes. As such, they are inserted in the inner circle of the architecture.
The entities are wrapped by the use cases, which represent the application specific business rule. They should encapsulate exactly the use cases defined in the requirement analysis.
The following circles are out of business logic and have to deal with external resources. As such, the first layer we encounter is the interface adapters, which translate the data into a convenient format for the underneath layers.

The implementation. Following the Clean architecture rules, the system obtained results to be robust and with a low level of dependencies. The architecture is divided as we can easily see in figure 5.16.
We managed to combine the entities and use cases in a single module, which is called *Do-*

main. The unusual thing in this architecture is the separation of the identity module from the business logic of our application. This was forced by the fact that this library was already implemented and we simply imported it inside the project with few modifications. The library was not implemented in clean architecture, as such, in order to not disrupt the dependencies, we managed to put it as inner circle next to the Domain layer, however, without any dependency with it.
The middle circle contains the infrastructure level. As such, here we can find the interface to the external resources and the mappers, which manage to translate the data from and to the inner circle. Finally, the outermost circle physically represents external resources, such as the database, external APIs, and the web application, which is the user interface.

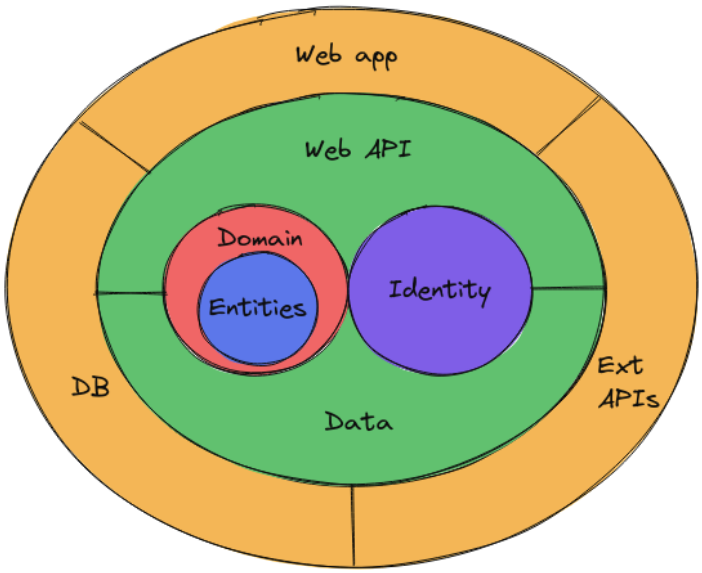


Figure 5.16: The Kerbero components organized with the clean architecture.

Evaluation of the architecture. After the implementation of the architecture, the dependencies schema appears to be as in the figure 5.17.

Following the previous rules was not that easy, in fact, the figure 5.16 represents the last version of the architecture to which we came. As such, the initial version of the architecture was substantially different from the latest one. The analysis and the feasibility study were not enough to determine the final architecture a priori. This is caused by the rigidity of *Dependency rule*, which sometimes does not conform to the best practices of external libraries or frameworks, as it declares. In fact, it turns out that if a library needs to be inserted in-

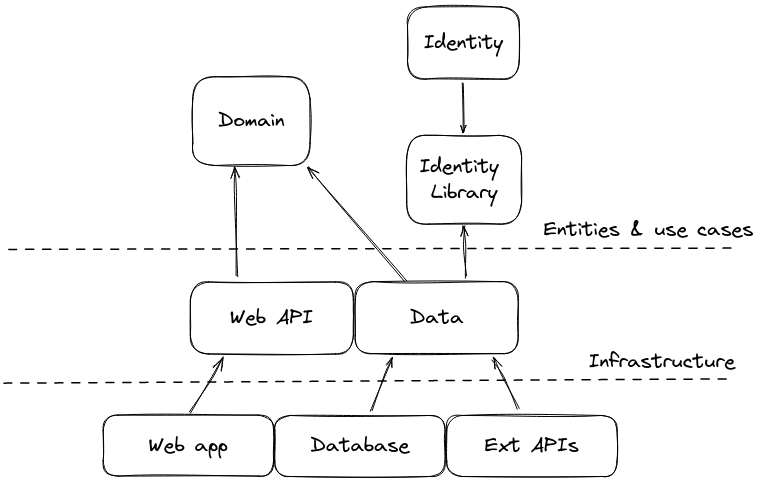


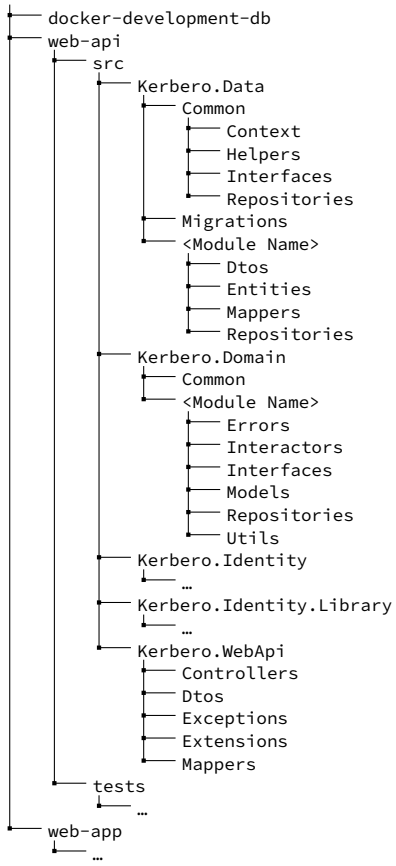
Figure 5.17: The dependencies schema of Kerbero.

side an internal circles (which in most of the cases must be avoided), then many problems emerge, especially if the latter does not implement the clean architecture first. As such, during the development can happen that the library, chosen as data access framework, needs a dependency on an object in order to work and the developer must move this object from an inner circle to an outer one; or he must create a copy of that class inside the module, and, as a consequence, he has to code the mappers and the utilities for that specific unit. As such, Clean architecture requires a large codebase to be implemented in a real-world scenario, with respect to the alternatives, such as the layered or the monolithic architecture. Moreover, we found that it was not that easy to read by an external developer, as such the components need a detailed description first.
However, the resulting application turned out to be, as expected, modular and with a low level of interdependencies between its components. This was a requirement for the scalability of the application, but the latter was not the only benefit. The application tests were easier to implement than expected, even the integration and end-to-end tests. Also, the debugging of the application was helped by the modularity, thanks to the ease of finding the error in isolated components.

Project structure and component naming

The architecture, finally, has determined the structure of the project. As such, from there is possible to understand even better the concepts outline in the previous sections.
From the below directory tree we can also notice the naming of the component in the archi-

tecture. Starting from *Domain*, we can identify the clean architecture entities and use cases, respectively, as *models* and *interactors*. Then, there are the errors, the interactors and the repositories interfaces, useful to be used with the Dependency Injection (DI).



Inside the *Data* component, we identify some of the classes and concepts related to external libraries, such as *Context* of the Entity Framework core tool. Specific of this layer is the implementation of the repositories, which are specific for each entity and provide the basic methods to manage them, as such the creation, reading, updating and deletion (CRUD). Moreover, there is a strangeness, which is the presence of *entities*. The latter are not to be interpreted in the clean architecture way, but they are named according to the naming of the library with which they are used, EF core. Both in the Data and WebApi layers, there are *dtos* and *mappers*. The first one are the Data Transfer Objects (DTOs), which represent the data structure entering and exiting the system. The characteristic of this type of data is the possibility of being serialized and deserialized the object, in order to fit in an HTTP message as a JSON. Finally, to deal with the different forms that the same data type can have (DTO,

entity, or model), there are *mappers*, which have the task of transforming the object from one type to another, modifying and filtering their attributes.

5.4.2 Workflow, versioning and conventions

The implementation of the application never reached the production state, therefore, it remained in a development state for the duration of the project. As such, we needed only two types of environment in which run the application. We used a workbench in which we can experiment the technologies and test the integration between the components, as such here we used to build Proof of Concept software. The other environment is the development one, which includes a database on a virtual machine and the client-server architecture, which run with ad hoc scripts on the localhost.
While the first environment was not versioned, the development environment is traced in a Version Control System. In particular, we use Git and Github to maintain a shared and remote repository for our codebase. Moreover, we used branching, rebasing and merging, in order to manage the concurrent and parallel development of features. In particular, there were three types of branch: *master* contains the most stable and reviewed code; the *feature* branches, which were used to develop a new module starting from the master or another feature branch code; and *fix* branches, which contain little, but disruptive, modification, aimed at solving bugs or errors.
Github has played an important role in the workflow during the development of the application. In particular, we exploited the Pull Requests (PR) feature. A PR is a merging request of a non-master branch into the master one. The importance of requesting first relies on the code that, before becoming part of the codebase, must be reviewed. Reviews were an important part of the project implementation phase. Moreover, Github offers other services, such as the possibility of launching actions on code pull, such as building and test commands. This grants having a working code in all branches.

5.4.3 The client architecture

The client of the application is developed as a Single Page Application (SPA). A SPA is a web application or website that loads a single HTML page and dynamically updates the content as the user interacts with the application. It is designed to provide a smooth user experience similar to a traditional desktop application. SPAs are built using client-side JavaScript frameworks, such as Angular, React, and Vue, which allow them to update the content dynami-

cally without having to refresh the entire page. When a user interacts with the application, the JavaScript framework updates the necessary parts of the page rather than loading a new page from the server. This results in a faster and more responsive user experience, since the application does not need to reload the entire page every time a user takes an action. SPAs typically use a routing mechanism to handle different URLs and will update the content based on the URL. They usually communicate with the server using APIs and JavaScript libraries, such as Axios or Fetch, to retrieve and update data. Kerbero uses a wrapped HTTP client around the Fetch library.
One of the main benefits of SPAs is that they can provide a smooth user experience, with fast and responsive navigation, similar to a traditional desktop application. They also reduce the amount of data transfer between the server and the client, which can improve the performance of the application. However, one of the main challenges of SPAs is that they are not as SEO-friendly as traditional web applications, as search engines may have difficulty indexing the dynamic content of the application. Additionally, if the JavaScript code fails to load or execute correctly, the SPA may not be able to function at all. This limitation was taken into account before choosing this approach. In fact, we think that, like many other web applications, Kerbero does not need the search engine indexing the site; moreover, the problem can be bypassed easily with a workaround, such as linking the application from an indexed presentation page.

5.4.4 Tests and security

Most of the application was implemented using Test-Driven Development (TDD). The TDD is a software development process that involves writing automated tests for a specific feature or behavior the actual implementation. The tests are then continuously run to ensure that they fail until the feature is not completely implemented. When the tests are passed, it means that the feature is ready and that the developer can refactor its code to make it more efficient and readable. This process helps to ensure that the code is thoroughly tested and that the new changes do not break existing functionalities. It also encourages a more modular and flexible design, as developers are forced to think about how their code will be used and how it will interact with the other parts of the system.
We focus, in particular, on server tests, which are of three types: unit tests, integration tests, and end-to-end tests (e2e tests). Unit tests were organized as a mirror of the source code directory. As such, all methods and functions are designed and tested before implementation following the TDD process.

The integration tests ensure that developers understand that all modules and layers communicate correctly with each other, while the end-to-end managed to test an entire feature from client input to response.
Testing was also designed to improve the security aspects, particularly those that involve authentication. During the end-to-end tests, for each of the actions exposed by the endpoint, it was verified that request with not valid cookies does not pass the identity controls, acted by the framework library.

5.4.5 Technologies and frameworks

The committer did not imposed any limitation about the technologies to use during the implementation. As such, having this freedom, we managed to choose the best frameworks and libraries that fit the requirements of the application. This process involves many elements and, as such, takes a significant part of the scheduled time for application development. In fact, for each of the following frameworks, a PoC was produced first, and then an evaluation, to better understand which combination of them can generate an application with such designing choices.

.NET and ASP.NET core

About the implementation of the application server and the web API, .NET was chosen a framework maintained and developed by Microsoft. .NET is a free, open source, cross-platform framework for building various types of applications, including web, mobile, desktop, gaming, and IoT. It includes a large library of pre-written code, a common runtime environment, and a set of tools and languages that can be used to build and run applications. Some of the languages that can be used with .NET include C#, F#, and Visual Basic. The .NET framework supports multiple operating systems such as Windows, Linux, and macOS.
Kerbero was initially be implemented with .NET 6, but during the development we found out that many features of the just released .NET 7 might be useful to us. As such, we managed an upgrade rollout of the application, upgrading the associated libraries as well.
The specific framework used to develop the web API was the ASP.NET core. ASP.NET is part of the .NET platform and is designed to be lightweight, high-performance, and modular. It provides a number of features that make it well-suited for building web applications, including support for routing, middleware, persistence and dependency injection, as well as

built-in support for security features like authentication and authorization. One of the main benefits of ASP.NET Core is its ability to be hosted in different ways, such as IIS, Apache, or self-hosting. Additionally, it provides a flexible pipeline for handling requests and responses, which allows developers to easily add custom middleware and services to handle specific functionality. ASP.NET Core is designed to be highly performant, scalable, and easy to test, and it can be used to build a wide variety of web applications, including web APIs, MVC web applications, and the one of our interest: SPAs.

The dependency injection. A feature that helps to implement the clean architecture is the Dependency Injection (DI). The DI is a design pattern that allows a class or component to receive its dependencies from an external source, rather than creating them internally or hard-coding them. These dependencies are typically services or objects that the class needs to perform its intended function. The key benefit of using dependency injection is that it promotes loose coupling between classes, making an application more flexible, maintainable, and testable. When classes are loosely coupled, they can be easily replaced or modified without affecting other parts of the application. This makes it easier to add new features, fix bugs, and improve performance.
One way to implement DI in .NET is to use constructor injection. This involves injecting the dependencies of a class through its constructor, allowing the class to use the dependencies without having to create or manage them. For example, using the built-in DI feature in .NET Core, you can register a service and its dependencies in the *Startup* class and then use the service in a controller by injecting it through the constructor.
Another way to implement DI in .NET is to use property injection. This involves injecting the dependencies of a class through its properties, allowing the class to use the dependencies without having to create or manage them. Overall, the basic steps to implement DI in .NET are:

- create an interface for the service that you want to inject;
- create a class that implements the interface;
- register the class and its dependencies in the container;
- inject the service into the constructor or property of the class that needs it.

EF core. Moreover, we managed to choose a framework that allows abstract database management, integrated with the .NET framework. As such, the choice has been left to the EF Core. Entity Framework Core (EF Core) is an open source and cross-platform Object-Relational Mapping (ORM) framework for .NET. In particular, it enables developers to work with relational data using domain-specific objects, and eliminates the need to write a lot of low-level data access code.
EF Core provides a set of APIs that allow developers to interact with a database using C# code, rather than writing raw SQL statements. It automatically generates the necessary SQL commands based on the C# code and the database schema, and maps the results to the appropriate domain objects.
EF Core also provides a powerful querying capability, allowing developers to use LINQ (Language-Integrated Query) to write type-safe, composable and expressive queries in C#. It also supports lazy loading, change tracking, and caching, which makes it easy to work with large data sets.
EF Core can work with different databases such as Microsoft SQL Server, MySQL, SQLite, PostgreSQL, and more through different providers, making it versatile and can be used in different types of project.
One of the key features of EF Core is its flexibility and ability to work with different data access scenarios. It can be used in server-side applications as well as in client-side applications (desktop and mobile). It also supports different deployment scenarios, such as on-premises and cloud-based.

ASP.NET core identity. For identity management, we integrate an existing project, which is a personalized version of the ASP.NET core identity library. As such, we make use of this wrapped version and adapt it to our purposes. ASP.NET Core Identity is a membership system that allows you to add authentication and authorization functionality to your ASP.NET Core web application. It provides a set of APIs and services for managing users, roles, and claims, and is built on top of the ASP.NET Core framework. ASP.NET Core Identity allows you to easily create user accounts and authenticate users in your application. It supports different types of authentication, including cookies, JWT tokens, and external providers like Google, Facebook, and Microsoft. It also provides built-in support for two-factor authentication, password hashing and salting, and account lockout policies. ASP.NET Core Identity also provides a way to manage users and their roles in your application. Provides a built-in user store that supports basic user management functionality, such as creating, updating,

and deleting users. It also allows you to define and manage roles and assign users to specific roles. Another important feature of Identity is claims-based authentication, which allows you to add additional information about a user, such as their name, email, and address, and use this information to make authorization decisions in your application.
While role management was not part of our scope, claims allow us to save user information and use it to perform our authentication checks. Moreover, ASP.NET Identity provides a testable process for implementing email confirmation of an account, which was particularly useful.

PostgreSQL

For the persistence it was selected PostgreSQL, due to its renown quality, such as stability, robustness, and feature richness. It is often used for web and enterprise applications, as well as data warehousing and analytics workloads. PostgreSQL supports a wide range of data types, including text, numbers, dates, and binary data, and also supports advanced data types such as arrays, hstore (a key-value store) and JSON. It also supports full-text search and GIS (Geographic Information System) data types. Additionally, it has a large number of built-in functions, operators and aggregates. PostgreSQL supports multi-version concurrency control (MVCC), which allows multiple transactions to access the same data simultaneously without conflicts. With respect to performance, they have particular impact features such as point-in-time recovery, hot standby, and logical replication, making them a suitable option for large-scale, high-availability systems. Moreover, PostgreSQL is known for its robustness and stability, and it is widely used in production environments, it is also a good option for large-scale and complex projects, it is supported by many operating systems, and it can be easily integrated with other software and tools.

Vue.js

The Single Page Application was developed with Vue. Vue.js is an open-source JavaScript framework for building user interfaces and single-page applications (SPAs). It is known for its simplicity and ease of use, making it a good choice for developers who are new to JavaScript frameworks. It uses a template syntax that allows you to declaratively render dynamic data into the DOM, making it easy to understand and debug the application. It also supports a component-based architecture, allowing developers to create reusable and composable UI components. Vue also provides a powerful set of directives and built-in directives

that allow you to easily manipulate the DOM, listen to events, and handle form inputs. It also provides a centralized state management system called Vuex, which makes it easy to share data between components and manage the application's state. Vue is also known for its flexibility and adaptability, it can be easily integrated with other libraries or existing projects, it also has a large and active community, which provides many resources such as tutorials, plugins, and packages.

6 Evaluation

The analysis and design of the application occupied a lot of time, as such the final Kerbero implementation was limited to the essential requirements. However, the system was developed to scale, therefore, the project is open to future improvements.
In this chapter, I will perform an evaluation of the results and give my opinion on the choices made during the design of the application.

6.1 Requirements satisfied

During the analysis, we identified few mandatory, but essential requirements. The discriminating factor was the requirements that classify essential and improving features. The following are the ones which were actually implemented in the system. The properties are all verified by integration and end-to-end tests.

User authentication. The user authentication is the basic way to protect user data and provide identity for the system; as such, this was an essential requirement for the system. Kerbero provides the possibility to sign in, with a classic flow, with the possibility of filling a form with email and password, and the email confirmation. The login is simple as well; therefore, the user inserts the email and password, while cookies grant the authentication state for the entire duration of the session.

Key generation and management. Key generation was a priority in Kerbero's design and development. The virtual key was actually implemented in the system, as such it is possible to operate on the following CRUD operations (Create, Read, Update, Delete). Moreover, the opening and closing of a smart-lock by its identifier from an anonymous endpoint, which does not require authentication, are available. The latter operation requires a random password that is created with the key and sent when it is shared.

Generic smart-lock management. The design of generic smart-locks was described in previous sections. The operations on smart-lock we managed to achieve with Kerbero are the display of all the smart-locks associated to a Kerbero user account, as such iterating on all the Provider account available. Moreover, a single smart-lock can be opened and closed remotely by an authenticated account.

Nuki authentication integration. It is possible to perform all OAuth2 authentication for the Nuki REST API, giving Kerbero user access to the operations on the Nuki smart-locks through the provider account. The credentials are persistent, as such the user does need to perform the authentication once and Kerbero remains linked to the Nuki account unitl the user decides to remove the integration.

Nuki smart-lock integration. The entire information, coming from Nuki REST API, is filtered and adapted to the Kerbero use cases. Moreover, Kerbero has a plugin that integrates the main operations (open and close) of the Nuki smart-lock devices.

6.2 Requirements not satisfied

Some of the expected requirements that we aim to achieve were not implemented. The reasons are different, but it is reasonable to think that they will be implemented in the future.

Integration with more than one provider. Integration of more than one smart-lock provider was a desirable requirement because it can verify the feasibility of a system that acts as a proxy interface for different REST APIs. A study of a new provider was started on the Kisi products. However, the lack of time and tools changes the priority of this particular requirement.

Integration with a Vacation Rental Management system. The idea of Kerbero starts from the analysis of how Vacation Rental Management systems, like Airbnb, can better implement

remote check-in. However, during the feasibility study, we identified some limitations on the integration of the APIs of the latter. In fact, all the analyzed platforms do not provide any public API with which to interface. Access to the actual existing APIs of Airbnb and Booking, is available only after an explicit request. We decided that this feature can be delayed after the development of a first version of the application in such a way that we can attach an example software to the integration request.

6.3 Limitations, future works and improvements

On the current implementation we discover several limitations and defects that can be improved in future versions.
The first we highlighted is the still presence of coupling between the Kerbero and Nuki smart-lock. During the design, there was a focus on generalizing the model and the operation related to the smart-lock. However, there are still too many references from the Kerbero core operations to the Nuki plugin. This is provoked by a less generic authentication flow, which couples the Nuki credentials (access token and refresh token) to the user account directly. We are aware that in this part of the application, a different design pattern can be applied to reduce dependencies.
Another improvement can be made in error management, flowing from the Web API to the client. In fact, all the error management has been done manually by the developer, which is not always a good idea. As such, a better system handled by the framework is desirable, but the current one does not provide a process that fits our requirements.
Concerning the security and authentication, there is a limitation of the cookies client authentication with the OAuth2 one server-side. In fact, the OAuth2 flow forces the cookie policy to be *Lax*, which means that the authentication session is valid for all domains reached by navigation from the main (which is the Kerbero client domain). As such, this exposes Kerbero to a security vulnerability. However, it is possible to create an ad hoc cookie with claims, which is lax and can only be used to perform OAuth2 authentication. The other cookie can be set as *Strict* to fill the security gap.

6.4 Workflow evaluation

Eventually, I want to express an opinion on the workflow adopted during the project. In particular, the one used by the company in which I collaborate during the development of the project.

The steps, we might have followed in order to address the release of the code, follow this order:

- the selection of an issue to solve from GitHub, preceded by the filtering using priority and affinity with others already in progress;
- the coding of a solution for the previous issue;
- the creation of a commit following the conventional rule;
- if the commit closes a feature, the generation of a Pull Request;
- finally, the reviewing and the merging of the code in the master branch.

The whole process is preceded by a weekly plan of work, through the use of milestones and the creation of the issues, based on the fixes and features to be done.
I, most of all, appreciate the review system. The creation of a Pull Request forces the developer to stop developing over the just produced code, and it allows others developers to check and comment on it. This brings about two clear advantages:

- a constant reviewed code, as such a good level of code quality;
- a positive knowledge transfer from developer to developer inside the team.

The latter grants continuous learning, both from the reviewer side, which can read the code of others, and from the reviewed one, which must adjust the code based on the comment received. In summary, this method is particularly useful for the knowledge-transfer process, even without a large documentation layer below.
This workflow can be adopted only in a continuous integration context, in which new features are constantly released in the main codebase, with the support of the automatic build and test system.
This concepts are important to me, because part of my experience was adapting to the company workflow and developing an infrastructure for my project which grants the previous assumptions.

7 Conclusion

In conclusion, the implementation of an application for managing smart-locks and Airbnb rentals has been shown to be a valuable addition to the current self check-in workflow. The application can provide a convenient and secure way for Airbnb hosts to manage access to their properties while also improving the overall guest experience. The use of the smart-lock technology ensures that access is granted only to authorized individuals and also eliminates the need for key exchanges, which can be time-consuming and potentially unsafe. Furthermore, the ability of hosts to remotely monitor and manage access to their properties through the application provides added peace of mind. Overall, the implementation of this application has been found to be a successful and practical solution for managing smart-lock and Airbnb rentals. Additionally, integration with Airbnb allows for easy management of multiple properties and bookings.

One of the most significant benefits of this technology is the convenience it provides for both hosts and guests. With the ability to remotely control access to the property, hosts no longer have to worry about coordinating key exchanges and can instead focus on managing their properties and bookings. Guests also benefit from the convenience of being able to access their rental at any time, without the need to coordinate key pick-up or drop-off.

Another important benefit is the added security layer provided by using a smart-lock. With the ability to remotely monitor access to the property, hosts can ensure that only authorized guests can enter. This added security also provides peace of mind for guests, knowing

that their rental is protected and secure.

Integration with Airbnb also allows for the easy management of multiple properties and bookings. Hosts can now manage all their listings and reservations from one central location, saving them time and effort. In addition, guests can easily book and access multiple properties through the same platform, providing a seamless and convenient experience.

Overall, the implementation of an application managing smart-lock and Airbnb has the potential to greatly benefit the vacation rental industry. It provides added convenience, security, and revenue for hosts, while also improving the customer experience for guests. It is clear that this technology should be considered for future developments in the vacation rental industry and could potentially become the new standard for managing vacation rentals.

Eventually, the technologies used in this project shows how it is possible to implement a robust and secure application. There are, of course, few compromises between usability and security, which are widely justified. The requirements outlined in the analysis bring to light the limitation and the strong points. The design of the software is an advanced example of what technology can do nowadays with less effort in implementation, in order to dedicate more time to the actual analysis.

Glossary

Airbnb is a online vacation rental management system, that enables people to rent out their properties or spare rooms to guests. It allows individuals to list their spaces on the platform and make them available to travelers looking for short-term lodging. Guests can search for available listings, view property details, and make reservations directly with the host. 2, 45, 68

Booking is an online platform that helps people find and book accommodation, transportation, and other travel-related services. It allows users to search for lodging options, compare prices, and make reservations with properties around the world. Services include hotels, vacation rentals, resorts, apartments, hostels and more. 2, 45, 69

Smart home is a residence that uses internet-connected devices to enable the remote control and automation of appliances and systems, such as lighting, heating, security, and entertainment. Smart home technology allows homeowners to monitor and control their home's functions from a smartphone or other device. 10

Application Programming Interface (API) is a set of rules and protocols that allows different software applications to communicate with each other. It is a way for one application to access the functionality of another application or service. 17, 20, 28, 30, 35, 44, 45, 47, 51, 54, 69

Bluetooth Low Energy (BLE) is a variation of the classic Bluetooth technology and operates in the 2.4GHz ISM band. It is designed to provide a low-cost, low-power and low-complexity wireless communication solution while maintaining a similar communication range as classic Bluetooth. 15, 24, 26

Extensible Markup Language (XML) is a markup language that is used to store and transport data. It is a way of encoding data in a format that can be easily read and understood by both humans and machines. XML uses a system of tags to define the structure and organization of the data. 21

HyperText Transfer Protocol (HTTP) is a protocol for sending and receiving data over the internet. It is the foundation of data communication for the World Wide Web. HTTP defines how messages are formatted and transmitted, and what actions Web servers and browsers should take in response to various commands. 15, 19, 30, 54

Institute of Electrical and Electronics Engineers (IEEE) is a membership-based organization that provides a wide range of services to its members, including publishing academic journals, organizing conferences and seminars, and developing industry standards. 12, 13, 24

Internet of Things (IoT) refers to the network of physical devices, vehicles, buildings and other items embedded with electronics, software, sensors and connectivity which enables these objects to connect and exchange data. This allows for the collection and sharing of data in real-time, enabling new efficiencies and capabilities in many industries. 1, 11

Logical Link Control and Adaptation Protocol (L2CAP) is a data link protocol that is part of the Bluetooth stack, that provides multiplexing, segmentation and reassembly of packets over the link between the two Bluetooth devices. 12

Near Field Communication (NFC) is a technology that allows two devices, such as a smartphone and a point-of-sale terminal, to communicate with each other when they are brought within close proximity, typically a few centimeters or inches. 11

Proof of Concept (PoC) is a prototype that is built to test the design or the underlying technology of a proposed solution, and to determine whether it can be developed into a full-fledged product or service. 46

Representational State Transfer (REST) is an architectural style for building web services. It is based on a set of principles for creating web services that are lightweight, client-server based, and stateless. 17, 23, 28, 30, 51, 54

Transmission Control Protocol/Internet Protocol (TCP/IP) is a set of networking protocols that define the way data is transmitted over the internet and other networks. 12

Transport Layer Security (TLS) is a widely-used security protocol for establishing secure connections between web servers and clients. It is the successor to SSL (Secure Sockets Layer) and is used to encrypt data sent over a network, such as the internet. 15, 22, 30, 50

Uniform Resource Identifier (URI) is a string of characters used to identify a resource, such as a web page, an image, or a file, on the internet.. 18

Universal Serial Bus (USB) is a standard type of connector for devices such as computers and consumer electronics. USB is used to connect devices, as well as to charge and transfer data between devices. 9

Universal Unique Identifier (UUID) is a 128-bit string that is guaranteed to be unique across all devices and all time. 26

Wi-Fi Protected Access 2 with Pre-Shared Key (WPA2/PSK) is a security protocol for wireless networks that uses a pre-shared password, known as the PSK, for authentication. It is an improvement over the original WPA standard and provides stronger security by using Advanced Encryption Standard (AES) encryption. 15

World Wide Web (WWW) is a system of interlinked hypertext documents accessed via the internet using a web browser. 13, 15

References

[1] C. Wilson, T. Hargreaves, and R. Hauxwell-Baldwin, "Benefits and risks of smart home technologies," *Energy Policy*, vol. 103, pp. 72–83, 2017.

[2] A. Y. Mulla, J. J. Baviskar, F. S. Kazi, and S. R. Wagh, "Implementation of zigbee/802.15.4 in smart grid communication and analysis of power consumption: A case study," in *2014 Annual IEEE India Conference (INDICON)*, 2014, pp. 1–7.

[3] Anon., *smart lock market size, share &; trends analysis report by type (deadbolt, lever handle, padlock), by application (residential, hospitality, enterprise), by region, and segment forecasts, 2022 - 2030*. Grand View Research, 2020. [Online]. Available: https://www.grandviewresearch.com/industry-analysis/smart-lock-market

[4] C.-Y. Law, K.-O. Goh, W.-S. Ng, C.-Y. Loh, and Y.-W. Sek, "The integration of smart lock in vacation rental management system," in *2020 IEEE 20th International Conference on Communication Technology (ICCT)*, 2020, pp. 846–850.

[5] T. Ikkala and A. Lampinen, "Monetizing network hospitality: Hospitality and sociability in the context of airbnb," in *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing*, ser. CSCW '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 1033–1044. [Online]. Available: https://doi.org/10.1145/2675133.2675274

[6] R. Dey, S. Sultana, A. Razi, and P. J. Wisniewski, "Exploring smart home device use by airbnb hosts," in *Extended Abstracts of the 2020 CHI Conference on Human Factors in Computing Systems*, ser. CHI EA '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 1–8. [Online]. Available: https://doi.org/10.1145/3334480.3382900

[7] Airbnb, "The do's and don'ts of providing self check-in," https://www.airbnb.com/resources/hosting-homes/a/the-dos-and-donts-of-providing-self-check-in-238, 2020.

[8] Anon., "About us." [Online]. Available: https://news.airbnb.com/about-us/

[9] M. Umair, M. A. Cheema, O. Cheema, H. Li, and H. Lu, "Impact of covid-19 on iot adoption in healthcare, smart homes, smart buildings, smart cities, transportation and industrial iot," *Sensors*, vol. 21, no. 11, 2021. [Online]. Available: https://www.mdpi.com/1424-8220/21/11/3838

[10] K. Sairam, N. Gunasekaran, and S. Redd, "Bluetooth in wireless communication," *IEEE Communications Magazine*, vol. 40, no. 6, pp. 90–96, 2002.

[11] C. M. Ramya, M. Shanmugaraj, and R. Prabakaran, "Study on zigbee technology," in *2011 3rd International Conference on Electronics Computer Technology*, vol. 6, 2011, pp. 297–301.

[12] G. Ho, D. Leung, P. Mishra, A. Hosseini, D. Song, and D. Wagner, "Smart locks: Lessons for securing commodity internet of things devices," in *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*, ser. ASIA CCS '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 461–472. [Online]. Available: https://doi.org/10.1145/2897845.2897886

[13] S. Kim, J.-Y. Hong, S. Kim, S.-H. Kim, J.-H. Kim, and J. Chun, "Restful design and implementation of smart appliances for smart home," in *2014 IEEE 11th Intl Conf on Ubiquitous Intelligence and Computing and 2014 IEEE 11th Intl Conf on Autonomic and Trusted Computing and 2014 IEEE 14th Intl Conf on Scalable Computing and Communications and Its Associated Workshops*, 2014, pp. 717–722.

[14] A. Zagorac and M. Antić, "Integration of third-party smart locks into the smart home system," in *2022 30th Telecommunications Forum (TELFOR)*, 2022, pp. 1–4.

[15] Jmaxxz, "Backdooring the frontdoor," 2016. [Online]. Available: https://www.youtube.com/watch?v=MMB1CkZi6t4

[16] E. Fernandes, J. Jung, and A. Prakash, "Security analysis of emerging smart home applications," in *2016 IEEE Symposium on Security and Privacy (SP)*, 2016, pp. 636–654.

[17] M. Ye, N. Jiang, H. Yang, and Q. Yan, "Security analysis of internet-of-things: A case study of august smart lock," in *2017 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2017, pp. 499–504.

[18] K. T'Jonck, B. Pang, H. Hallez, and J. Boydens, "Optimizing the bluetooth low energy service discovery process," *Sensors*, vol. 21, no. 11, 2021. [Online]. Available: https://www.mdpi.com/1424-8220/21/11/3812

[19] M. Mikolits, "The nuki encryption concept," Oct 2015. [Online]. Available: https://nuki.io/en/blog/nuki-news/nuki-encryption-concept/

[20] D. J. Bernstein, "Salsa20 specification," *eSTREAM Project algorithm description, http://www. ecrypt. eu. org/stream/salsa20pf. html*, 2005.

[21] G. Procter, "A security analysis of the composition of chacha20 and poly1305," *Cryptology ePrint Archive*, 2014.

[22] C. Caballero-Gil, R. Álvarez, J. Molina-Gil, and C. Hernández-Goya, "Smart-lock attack through bluetooth communications replication," in *Proceedings of the International Conference on Ubiquitous Computing & Ambient Intelligence (UCAmI 2022)*, J. Bravo, S. Ochoa, and J. Favela, Eds. Cham: Springer International Publishing, 2023, pp. 977–982.

[23] P. Chavan, "Nuki web api," *Nuki Home Solutions GmbHMünzgrabenstrasse 92/4, 8010 Graz*, Nov. 2021. [Online]. Available: https://developer.nuki.io/page/nuki-web-api-1-4/3/

[24] ——, "Nuki web api," *Nuki Home Solutions GmbHMünzgrabenstrasse 92/4, 8010 Graz*, Feb. 2021. [Online]. Available: https://developer.nuki.io/page/nuki-web-api-webhooks-11/8/

[25] K. LaCroix, Y. L. Loo, and Y. B. Choi, "Cookies and sessions: A study of what they are, how they work and how they can be stolen," in *2017 International Conference on Software Security and Assurance (ICSSA)*, 2017, pp. 20–24.

[26] S. Bingler, M. West, and J. Wilander, "Cookies: HTTP State Management Mechanism," Internet Engineering Task Force, Internet-Draft draft-ietf-httpbis-rfc6265bis-11, Nov. 2022, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-httpbis-rfc6265bis/11/

[27] M. Anicas, "An introduction to oauth 2," Jul 2014. [Online]. Available: https://www.digitalocean.com/community/tutorials/an-introduction-to-oauth-2

[28] R. C. Martin (Uncle Bob), "The clean architecture," Aug 2012. [Online]. Available: https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html

Acknowledgments

First, I would like to thank my supervisor, Prof. Eleonora Losiouk, who helped me write this thesis and assisted me during the development of the project.
I also must thank Kuama, the hosting company, and the tutor, who gave me all the tools and knowledge to implement such a project.
Finally, I would like to thank, my University mate, Alessandro Sgreva, without which I couldn't overcome most of the challenges we faced during the courses projects; and my friends and family, that supported me during all these years and eventually allows to achieve this important goal.