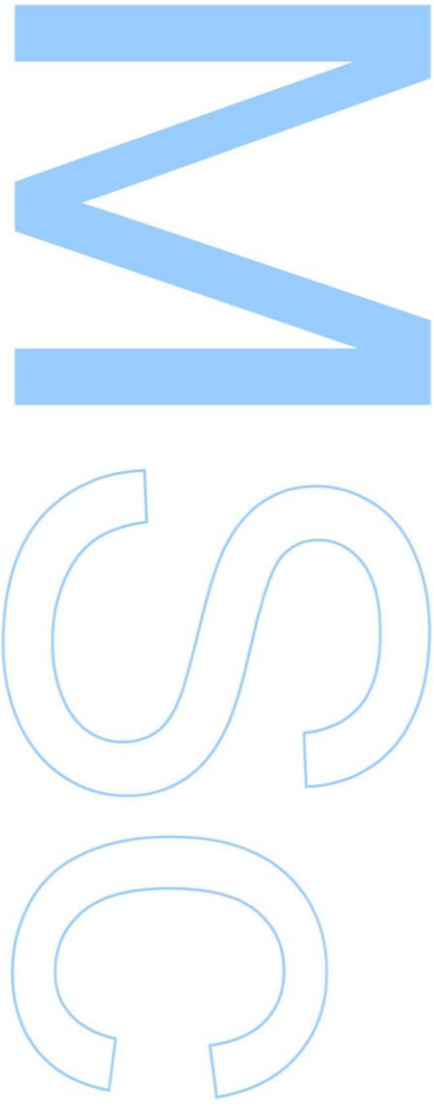
Blue Door

Tiago Manuel Vieira Martins
Engenharia de Redes e Sistemas Informáticos
Departamento de Ciências de Computadores
2018

Orientador
Luís Filipe Coelho Antunes, Professor Associado,
Faculdade de Ciências da Universidade do Porto

Coorientador
Pedro Manuel Roque Cabral, Investigador,
Faculdade de Ciências da Universidade do Porto



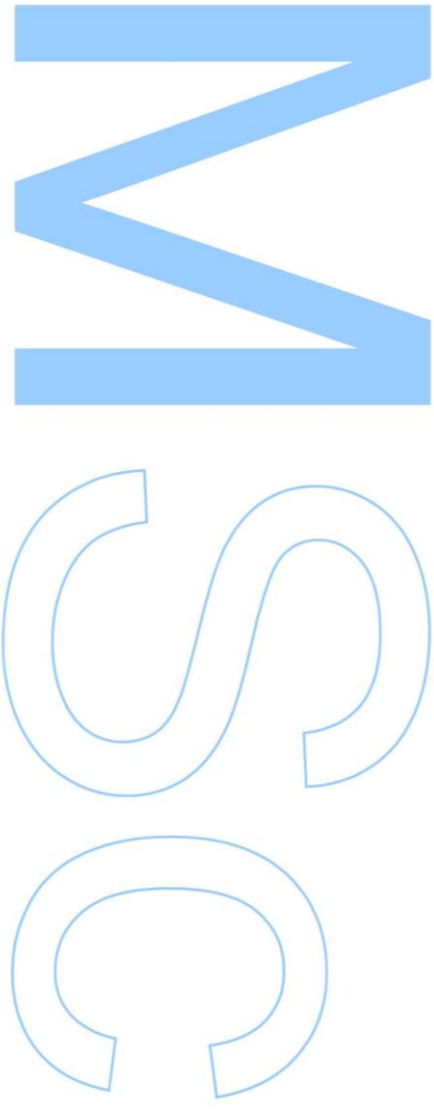
U.PORTO

FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

Todas as correções determinadas pelo júri, e só essas, foram efetuadas.

O Presidente do Júri,

Porto, _____/_____/_____



Abstract

Nowadays, a large number of physical access control systems, that are implemented in many distinct environments, such as airports, office buildings and even universities, are based on contactless smartcards. However, it has already been shown that these cards have several flaws that can be easily exploited, also putting their surrounding system at risk. Therefore, it is necessary to develop an alternative to achieve a truly secure access control system.

In the recent years, the smartphone became an almost indispensable equipment for people, replacing several tasks and even several objects from our everyday life, with a simple mobile application.

With that in mind, we designed a physical access control system, that instead of using the insecure smartcard, it takes advantage of the smartphone capabilities, thus creating a possible alternative way for securely authenticate the user.

In this thesis, we first present the desired properties and characteristics that our idealized access control system would have. We also explain the overall composition and operation of the system, including the two main activities: the enrollment, where the user registers himself in the system, and the authentication, where the actual user's access permissions are verified.

We then describe a prototype implementation of the presented access control system, composed by an Arduino Uno board, a BLE Nano 2 module and an Android smartphone, that is able to create a secure communication channel for the authentication to be fulfilled.

Resumo

Hoje em dia, um grande número de sistema de controlo de acesso físico, que são implementados em vários ambientes distintos, tais como aeroportos, edifícios empresariais e até universidades, são baseados em *contactless smartcards*. No entanto, já foi demonstrado que este tipo de cartões têm várias falhas e que podem ser facilmente exploradas, pondo também o seu sistema envolvente em risco. Devido a esta problemática, é necessário desenvolver uma alternativa para obter um sistema de controlo de acesso verdadeiramente seguro.

Nos últimos anos, o *smartphone* tornou-se num equipamento quase indispensável para as pessoas, substituindo várias tarefas e até vários objetos do nosso quotidiano, com uma simples aplicação móvel.

Com isso em mente, concebemos um sistema de controlo de acesso físico, que em vez de utilizar o *smartcard* inseguro, utiliza e tira vantagem das capacidades do *smartphone*, criando assim uma possível alternativa para autenticar o utilizador com segurança.

Nesta tese, começamos por apresentar as propriedades e características desejadas que o nosso sistema de controlo de acesso idealizado teria. Também explicamos a composição e funcionamento geral do sistema, incluíndo as duas principais atividades: o *enrollment*, onde o utilizador se regista no sistema, e a autenticação, onde as permissões de acesso do utilizador são verificadas.

De seguida, descrevemos a implementação de um protótipo do sistema de controlo de acesso apresentado, composto por um Arduino Uno, um módulo BLE Nano 2 e um *smartphone* Android, que é capaz de criar um canal de comunicação seguro, para que a autenticação seja realizada.

Aos meus pais e à minha irmã

Contents

Abstract i

Resumo ii

Contents vi

List of Tables vii

List of Figures viii

Acronyms ix

1 Introduction 1

1.1 Motivation 2

1.2 Proposed objectives 2

1.3 Structure 2

2 State Of The Art 3

2.1 Cryptographic protocols 3

2.1.1 Rivest-Shamir-Adleman 4

2.1.2 Advanced Encryption Standard 5

2.1.2.1 Counter with CBC-MAC / AES-CCM 6

2.1.2.2 Galois/Counter Mode / AES-GCM 6

2.1.3 Diffie-Hellman 7

2.1.4 Elliptic Curve Cryptography . . . 7
2.1.5 Elliptic-curve Diffie-Hellman . . . 8
2.1.5.1 Curve25519 . . . 9
2.2 Communication protocols . . . 9
2.2.1 Quick Response Code . . . 9
2.2.2 Bluetooth Low Energy . . . 10
2.2.3 Near Field Communication . . . 11
2.2.4 Wi-Fi Direct . . . 13
2.3 Physical access control systems . . . 13
2.3.1 Bluetooth-based access control system . . . 14
2.3.2 NFC-based access control system . . . 14

3 Proposed Solution **16**
3.1 Overview . . . 16
3.2 System Functioning . . . 18
3.2.1 Enrollment . . . 18
3.2.2 Authentication . . . 19
3.3 Communication methods . . . 20
3.3.1 Communication in Enrollment . . . 20
3.3.2 Communication in Authentication . . . 21

4 System Implementation **23**
4.1 Overview . . . 23
4.2 Material Used/System Architecture . . . 24
4.3 System Operation . . . 25
4.3.1 Android application . . . 25
4.3.1.1 Features and permissions . . . 25
4.3.1.2 Operation . . . 26
4.3.2 BLE Nano 2 . . . 29

4.3.3 Arduino Uno 30
4.4 Problems found 33

5 Conclusion and Future Work **34**

Bibliography **35**

List of Tables

2.1 Communication protocols comparison 13

4.1 Material used in the implementation 24

List of Figures

2.1 AES Encryption process, image from [43] 5
2.2 Elliptic curve example, image from [6] 8

3.1 System components diagram (example) 17
3.2 Communication sequence between components 18
3.3 Communication protocols between components 22

4.1 Schematic representation of the Arduino and BLE Nano 2 circuit 25
4.2 Android application's list of found devices 27
4.3 Android application activity diagram 29
4.4 Arduino and BLE Nano 2 with green LED lighted up 32
4.5 Communication sequence between the Arduino and the Smartphone 32

Acronyms

AES Advanced Encryption Standard

AES-CCM Advanced Encryption Standard - Counter with Cipher Block Chaining-Message Authentication Code

AP Access Point

BLE Bluetooth Low Energy

CBC-MAC Cipher Block Chaining - Message Authentication Code

CCM Counter with Cipher Block Chaining - Message Authentication Code

CSRK Connection Signature Resolving Key

CTR Counter

DES Data Encryption Standard

ECC Elliptic Curve Cryptography

ECDH Elliptic Curve Diffie-Hellman

ECDLP Elliptic Curve Discrete Logarithm Problem

GCM Galois/Counter Mode

GFSK Gaussian Frequency Shift Keying

GHASH Galois Hash

GO Group Owner

HCE Host Card Emulation

HF High Frequency

IRK Identity Resolving Key

ISM Industrial Scientific Medical

LED Light-Emitting Diode

LF Low Frequency

LTK Long Term Key

MAC Message Authentication Code

MITM Man in the Middle

MTU Maximum Transmission Unit

NFC Near Field Communication

NIST National Institute of Standards and Technology

OOB Out of Band

PACS Physical access control system

PIN Personal Identification Number

PSK Phase Shift Keying

P2P Peer-to-Peer

QR code Quick Response Code

RF Radio Frequency

RFID Radio Frequency Identification

RSA Rivest-Shamir-Adleman

SE Secure Element

SIG Special Interest Group

SIM Subscriber Iidentity Module

SoC System-On-Chip

TK Temporary Key

TLS Transport Layer Security

URL Uniform Resource Locator

WPA2 Wi-Fi Protected Access II

WPS Wi-Fi Protected Setup

Chapter 1

Introduction

Bluetooth Low Energy (BLE) is a wireless network technology designed for short to medium range communications. It was developed and licensed by Bluetooth Special Interest Group (SIG), that introduced it in 2010 in the Bluetooth 4.0 specification. With BLE, it is possible to connect and share information between a wide range of electronic devices, like smartphones, computers, printers and beacons. BLE was designed to be a communication protocol with a low energy consumption, and also to be a low cost technology. With BLE, several new features were implemented with the aim of improving the security and efficiency of discovery and pairing between devices. BLE communicates via radio, using the 2.4 GHz ISM band, and can have a maximum range of several tens of meters. With this last feature, the devices do not need to have a clear line of sight from each other to be able to communicate.

Near Field Communication (NFC) is a communication standard created from the Radio Frequency Identification (RFID) technology, that enables two devices to communicate, simply by being close to each other. It was released as a specification in 2002, by Sony and Philips. The NFC-capable devices are divided in two categories: passive and active devices. An active device uses its own power to create the Radio Frequency (RF) field that is used to transmit the data, while a passive device uses the RF field created by other devices to communicate. Some NFC devices contain a Secure Element (SE), identical to the ones found in smartcards, which represents a environment capable of storing and processing data in a secure way.

Each contactless smartcard has an unique identifier that can be read without authentication, by any available reader. That identifier, in addition to assigning an identity to the card, also prevents collisions during the reading process, if there are other smartcards nearby. Although unique, the identifier is attributed based on the type, manufacturer and serial number of the card. Therefore, it is not difficult to find the identifiers of cards from the same lot, just by knowing one of them. This is the main reason why the identifier is not secure enough to be used in authentication and access control systems.

A correct implementation of an authentication protocol for access control, requires a system that allows authenticating a card and validating the access, by using the communication with

the SE to implement a cryptographic protocol for authentication. That is, besides identifying the read card, there is also the need for a parallel system to validate the access privileges of that card.

1.1 Motivation

A fairly large number of access control systems are based on contactless smart cards. However, as Cabral demonstrated and concluded in his thesis[10], the security in this type of cards can be almost non-existent, and even cards that have implemented some sort of cryptographic protocol, can be easily exploited.

This raises an important problem. Nearly all the access control systems that uses smart cards, are not secure and can be compromised. The motivation for this work is to design and implement a system with a high degree of usability and security, that solves the problem identified by Cabral.

1.2 Proposed objectives

We aim to design, develop and implement a prototype of a secure physical access controller based on a smartphone, mainly for proof of concept. The proposal must have the following properties:

1. Multi-platform (Android and iOS);
2. Secure cryptographic protocols;
3. Range of several meters;
4. Low energy consumption.

1.3 Structure

The remainder of this thesis is divided in 3 chapters. In Chapter 2, it is presented some background knowledge regarding some cryptographic protocols, as well as wireless communication protocols. In Chapter 3, it is presented our idea for a secure physical access control system based on the use of a smartphone. Chapter 4 contains a description of how the prototype of our proposed solution was developed.

Chapter 2

State Of The Art

In this chapter, it is addressed and explained several subjects that either are relevant or related with the work to be done in this thesis.

First, it is mentioned some basic concepts about cryptographic protocols. Following, the Rivest-Shamir-Adleman (RSA), Advanced Encryption Standard (AES), Diffie-Hellman, Elliptic Curve Cryptography (ECC) and Elliptic-curve Diffie-Hellman (ECDH) protocols are presented.

Next, it is given a general overview of the Quick Responde Code (QR code), BLE, NFC and Wi-Fi Direct. Finally, some already implemented physical access control systems are presented.

2.1 Cryptographic protocols

A protocol specifies a sequence of steps of how an interaction between to or more parties must occur, in order for a task to be accomplished or a problem to be solved. A protocol needs to have the following characteristics:

- All the parties involved must, beforehand, know the protocol and all its steps;
- All the parties involved must agree with all the protocol steps;
- There cannot be any ambiguity in any step of the protocol;
- The protocol must be complete, in a way that it covers all possible situations.

A cryptographic protocol follows the previous definition. A key characteristic of this type of protocol, is that it uses cryptographic algorithms. The parties taking part in the protocol can share secrets, prove one another identity, or sign a contract. The main utility of cryptographic protocols is to ensure privacy and prevent eavesdropping[41].

Cryptographic protocols can be divided in two different types: Symmetric Key, also known as Secret Key Cryptography, and Asymmetric Key, also know as Public Key Cryptography.

In Symmetric Key algorithms, the same key is used for both the encryption and decryption functions. The sender and the receiver must then, agree on a secret key K, before they can send encrypted messages to each other. The sender, encrypts some message M with the key K, obtaining a ciphertext C. Then, the receiver decrypts the ciphertext C with the same key K, getting the original message M. This two processes can be represented as $E_K(M) = C$ and $D_K(C) = M$, respectively[27]. Symmetric Key algorithms can use stream ciphers, meaning that it encrypts/decrypts the bits from a message, one at a time. It can also use block ciphers, where the message is divided into groups of bits, and each one is encrypted/decrypted as a unit.

In the case of Asymmetric Key algorithms, the keys used for encryption and decryption are different. The key used in the encryption of a message can be known by anyone, since only one person has another specific key that can decrypt that message. Given that the encryption key is public, it must be infeasible to compute the decryption key only with that knowledge[27]. The encryption key can be referred as public key (K_p), and the decryption key is also named as private key (K_s). The processes of encrypting a message M and decrypting a ciphertext C can be represented as $E_{K_p}(M) = C$ and $D_{K_s}(C) = M$, respectively.

Forward Secrecy is an important feature in key agreement protocols, that protects past shared secret keys even if future private keys get compromised. If both parties agree in a unique secret key for each session, then if that secret key is discovered by a third party, only the data transmitted in that session can be affected, while the previous sessions remain protected[29].

2.1.1 Rivest-Shamir-Adleman

RSA is a public-key encryption cryptosystem designed in 1977 and named after its creators, Ron Rivest, Adi Shamir and Leonard Adleman. This algorithm is used for data encryption, key exchange and digital signatures. In public-key cryptography, there is a key pair associated with each individual. It contains a public key that is shared with everyone and used to cipher, and a private key, that is secret and is used to decipher. RSA's security is based on the difficulty of factoring large numbers, more specifically, large integers that result from the product of two large prime numbers[41].

In the key generation process, first it's selected two different, random and large prime numbers, p and q. Then, the value n is calculated from the product of p with q. The value n is the connection between the public and the private key, since it is used to compute both of them[13]. The size of n represents the key size that RSA uses, which is typically between 2048 and 4096 bits. The next step is to randomly choose the encryption key e, which needs to be co-prime with (p-1)(q-1). The final value to be obtained is the decryption key d, which is the multiplicative inverse of n mod $(p-1)(q-1)$. Using the extended Euclidean algorithm it's possible to calculate d, such that $ed \equiv 1 \bmod (p-1)(q-1)$.

In the encryption operation, a message m is transformed in an encrypted message c. First, m is divided into blocks smaller than the value of n, meaning that c will be composed from several

c_i blocks. Each one of those blocks will contain the value of $m_i^e \bmod n$. Finally, to decrypt an encrypted message, it is simply applied the formula $c_i^d \bmod n$ to each c_i block, in order to obtain all the m_i blocks that compose the original m message.

As previously stated, the security of the RSA algorithm relies on the difficulty of factoring large numbers. To date the biggest factorized RSA number has 232 digits and a size of 768 bits[31]. However, it was already possible to break a 4096-bit RSA, using an acoustic cryptanalysis attack[23].

2.1.2 Advanced Encryption Standard

In 1997, the need for a substitute of the Data Encryption Standard (DES), lead the National Institute of Standards and Technology (NIST) to accept a new encryption algorithm, the AES[12]. From all the different implementations of the algorithm that were received, fifteen went through a thorough evaluation. In October 2000, NIST announced that it had chosen the Rijndael cipher, developed by Vincent Rijmen and Joan Daemen, as the algorithm for AES.

The Rijndael cipher is a symmetric key algorithm, meaning that it uses the same key in the encryption and decryption process. It supports three different key sizes, 128-bit, 192-bit and 256-bit, but only one block size of 128-bit. Since AES works with bytes, the 128-bit data block is divided into 16 bytes, that are arranged in a 4x4 matrix, called the state matrix. Both in the encryption and decryption process, the AES algorithm performs 10 rounds if the key size is 128-bit, 12 rounds if the key size is 192-bit, and 14 rounds for 256-bit key size[43].

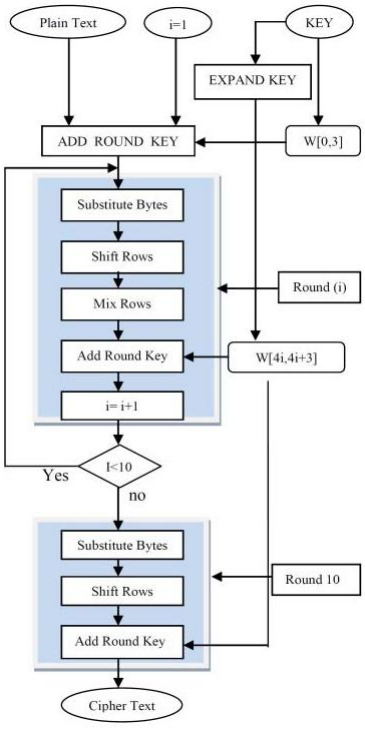


Figure 2.1: AES Encryption process, image from [43]

There are four different transformations in each one of the rounds, with only the exception of the last round, that only executes three of those four transformations. In the *SubBytes* operation, each one of the 16 bytes from the state matrix is converted into another block using an 8-bit substitution box, known as Rijndael S-box. In the *ShiftRows* transformation, a simple shift operation is applied to each row of the matrix. The first row is not shifted at all, the second row is shifted one byte to the left, the third row two bytes to the left, and finally, the fourth row is shifted 3 positions to the left. Next there is the *MixColumns* operation, where each four byte column of the state matrix is transformed, by multiplying it with a fixed matrix. The *MixColumns* operation is not executed in the last round of the AES algorithm. The last operation is called *AddRoundKey*. In this function, the 128 bits of the state matrix are XORed with the 128 bits of the round key. The state matrix is replaced with that result. In the last round, the state matrix corresponds to the ciphered text.

The decryption process is very similar to the encryption one, but in an opposite order. Both the sequence of the functions and the functions themselves are reversed[44].

The AES has demonstrated to be a very reliable and secure cipher. To date, there is no known practical attack against it. Some attacks like [24] and [8], were able to break the AES cipher. However these attacks were made on reduced round versions of AES, meaning that they are not effective in the normal version of AES.

2.1.2.1 Counter with CBC-MAC / AES-CCM

Counter with Cipher Block Chaining Message Authentication Code (CCM) is an authenticated encryption cipher mode, designed to work with 128-bit block ciphers, such as AES. CCM works by combining the CBC-MAC technique and the Counter (CTR) mode[46].

Let us assume that a message m can be viewed as a sequence of blocks. In the first stage, CBC-MAC is applied to that sequence in order to obtain a message authentication code (MAC) representing the message m. A MAC is a tag that is used to provide authentication and integrity to a message[38]. CBC-MAC works together with a block cipher encryption, so, in this case, it is used the AES encryption function, hence the name AES-CCM.

On the second stage, the message and the MAC obtained on the first stage are encrypted together with Counter mode.

2.1.2.2 Galois/Counter Mode / AES-GCM

Galois/Counter Mode (GCM) is a mode operation used for symmetric key cryptographic block ciphers, such as AES, that provides authenticated encryption, thus achieving both data integrity and confidentiality[2].

This mode works with block ciphers with an 128-bit block size, and its operation consist of

two major separate functions. First, it is used a variation of the Counter mode of operation for the encryption process. Then, the authentication process is accomplished by using the Galois Hash (GHASH) function, that creates an authentication tag[33].

2.1.3 Diffie-Hellman

The Diffie-Hellman protocol was first published in 1976 by Whitfield Diffie and Martin Hellman[17]. Diffie-Hellman is a key exchange and agreement protocol, which allows two random parties, that do not know neither have any information about each other, to exchange a shared secret key over a public and insecure communication channel. That shared key can then be used by any symmetric-key protocol, thus encrypting the communication and establishing a secure channel between the two parties. The protocol's security is based on the discrete logarithm problem[22], which is considered to be computationally infeasible.

Considering two public known numbers p and e, where p is a large prime number and e is the primitive root of p, and lets suppose that two parties, Alice and Bob, want to establish a shared key. First, Alice generates a random value a, such that $a < p$. In the same way, Bob generates a random value b. These values will remain secret to each party. Then, both calculate their public values: Alice obtains $A = e^a \mod p$, while Bob obtains $B = e^b \mod p$. Next, Alice sends its public value A to Bob, and Bob sends its public value B to Alice. Finally, Alice calculates $K = B^a \mod p$ and Bob calculates $K = A^b \mod p$, where K represents the shared secret key. It is easy to demonstrate that both Alice and Bob end up with the same K value:

$$\begin{aligned} K = B^a \mod p &= e^b \mod p^a \mod p \\ &= e^{ba} \mod p \\ &= e^b * a \mod p \\ &= e^{ab} \mod p \\ &= e^a \mod p^b \mod p \\ &= A^b \mod p \\ &= K \end{aligned}$$

In addition to the public values p and e, an eavesdropper can also get access to A and B, since these values are sent via an insecure channel. However, even with all these values, an attacker can not calculate the key K generated by Alice and Bob, because both a and b were not transmitted and are they required to accomplish that calculation.

2.1.4 Elliptic Curve Cryptography

The concept of ECC was first proposed, independently, by Neal Koblitz[32] and Victor Miller[37], in 1985. ECC can be defined as a public key cryptographic system based on elliptic curves over finite fields, and is mainly used for key agreement and digital signing.

An elliptic curve over a finite field K, is the set of all points satisfying the equation:

$$y^2 = x^3 + ax + b$$

where $a, b \in K$ and $4a^3 + 27b^2 \neq 0$. The values of a and b define the shape of the curve. Two main characteristics of elliptic curves, is that they are symmetric about the x-axis, and that by drawing a line over two points belonging to the curve P and Q, that line will intersect a third point R, that is also in the curve.

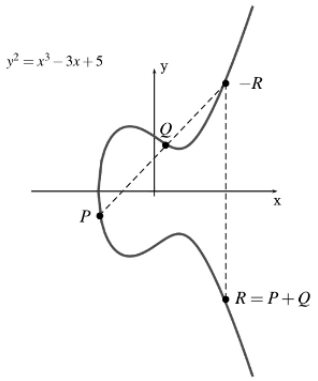


Figure 2.2: Elliptic curve example, image from [6]

Figure 2.2 shows a visual representation of an elliptic curve.

ECC is based on the Elliptic Curve Discrete Logarithm Problem (ECDLP), that raises the following question: "if we know the points P and Q (which belong to the curve), what is k such that $Q = kP$?". The difficulty in finding the value of k is the basis of the ECC security.

One advantage that ECC has over RSA, it that it can use considerably smaller keys than RSA, while maintain the same level of security[14].

2.1.5 Elliptic-curve Diffie-Hellman

ECDH is a key agreement protocol that enables two parties, with two distinct elliptic curve public-private key pairs, to achieve a shared secret key, over a possible insecure channel[22]. This generated secret key can then be used as an encryption key in a symmetric-key algorithm.

As the name implies, this protocol is a variant of the Diffie-Hellman protocol with elliptic-curve cryptography. Before the protocol itself begins, both parties must agree in the elliptic curve parameters that they are going to use[15]. These parameters are: the prime p, that represents the size of the finite field; coefficients a and b, from the elliptic curve equation; the base point G, which generates the subgroup; the order n and the cofactor h of the subgroup.

To create a private key, one must choose a random integer d in the interval from 1 to n - *1*. Only after, the public key can be obtained, by calculating $H = dG$, which means, adding the base point G to itself, d times.

When both parties have generated their private and public keys, they can share their public key with each other. This way, using their own private key and the other party's public key, it is possible to create a shared secret key S between them.

2.1.5.1 Curve25519

Curve25519 is an elliptic curve defined by the equation:

$$y^2 = x^3 + 486662x^2 + x$$

that was published by Daniel Bernstein in 2005[7], with the intent of being used in an ECDH protocol. However, the actual function that performs the ECDH key exchange is often referred as X25519.

Both the private and public keys are only 32 bytes long, as well as the generated shared secret key. In addition to having 128 bits of security, it has also a very fast computational speed and it allows any 32 byte string as a valid public key.

2.2 Communication protocols

A communication protocol is a group of rules, regulations and instructions, that makes possible for devices to communicate and exchange data with each other[5]. A communication protocol defines how the transmission is achieved, how the error detection is handled, and other properties like packet size and data rate.

2.2.1 Quick Response Code

QR code is a two-dimensional barcode, that was developed in 1994 by the Denso Wave company. Comparing with a traditional barcode, the QR Code has a faster readability and a much higher storage capacity[34].

A QR Code is represented by a black and white matrix, where the information is encoded through the squares arranged in the matrix. The data can be accessed by taking a picture of the code and processing it using a QR reader.

The usage of QR Codes have been increasing rapidly over the years, due to its simplicity and high number of potential use cases, such as product identification and tracking, Uniform Resource Locator (URL) storing, or simply data sharing.

Each QR Code is capable of storing up to 7089 numerical characters, 4296 alphanumerical characters and 2953 8-bit characters.

2.2.2 Bluetooth Low Energy

The Bluetooth Low Energy (BLE) was developed by Bluetooth Special Interest Group (SIG) and was first introduced in 2010 as a new feature in the Bluetooth 4.0 specification. It was later updated and improved in Bluetooth 4.1 and 4.2 versions. BLE, which is also know as "Bluetooth Smart", is a wireless technology for short to medium range communications. This protocol is already incorporated into billions of devices, such as smartphones, laptops, beacons, and a wide variety of Internet of Things sensors. BLE has a range of several tens of meters and it is mainly used in applications that don't need to send a huge amount of data. Unlike the classic Bluetooth, BLE doesn't stay turned on the entire time[30]. Instead, when there is no data being transferred, it enters a sleep mode, allowing the device to save battery and to remain on a single battery power for years. The main differences between BLE and the previous Bluetooth versions, are the reduction of the power consumption and shorter packet lengths, the reduced memory requirements, improvement of the efficiency and security in the connection procedures[39].

BLE uses the same frequency band as the previous Bluetooth versions, the unlicensed 2.4 GHz Industrial Scientific Medical (ISM) band. It contains a total of 40 Radio Frequency (RF) channels with 2 MHz spacing between them, which are divided in 2 different channel categories[25]. There are 37 Data Channels, that are used for communication/data exchange between the connected devices. The 3 remaining channels are the Advertising Channels, that are used for device discovery and connection, as well as broadcasting transmissions. All the 40 channels use a Gaussian Frequency Shift Keying modulation (GFSK) and have a maximum data rate of 1 Mbps.

Several new security features were implemented in BLE, such as the *low energy private device addresses* and the *data signing*. The first one uses a cryptographic key called Identity Resolving Key (IRK) that tries to mitigate the threat of being tracked by an adversary, by using a periodically-changing random address which is mapped to a device's Identity Address. The second feature, uses a cryptographic key called Connection Signature Resolving Key (CSRK), that is used to sign and verify signatures on a receiving device, providing authentication and integrity in a Bluetooth connection[47].

BLE uses the AES-CCM cipher mode for both encryption and device authentication. AES-CCM provides confidentiality, integrity and authentication.

In the versions 4.0 and 4.1 of BLE, it is used the AES-128 cipher as the pairing algorithm. However, in the BLE version 4.2, the added Secure Connections feature changed the pairing algorithms to AES-CMAC and P-256 Elliptic Curve[4] implementation of ECC. This lead to the renaming of the BLE 4.0 and 4.1 low energy pairing to low energy Legacy Pairing. In low energy Secure Connections, the use of ECDH-based cryptography during the key exchange process, offers protection against MITM attacks and eavesdropping. Since low energy Legacy Pairing doesn't use that type of cryptography, it has no protection against those attacks. From the pairing procedure it's generated a key called Long Term Key (LTK), that represents the symmetric key used in the authentication and encryption processes.

BLE offers two distinct security modes, where each one contains several levels. Security Mode 1 has levels related with encryption. In level 1 there is no security, meaning, neither authentication nor encryption. Level 2 specifies unauthenticated pairing with encryption. Level 3 requires authenticated pairing with encryption. Finally, in BLE 4.2, it was added the Level 4, that uses authenticated Secure Connections pairing with encryption. Security Mode 2 is more focused in data signing, so, although it provides data integrity, there is no confidentiality. In Level 1 there is unauthenticated pairing with data signing, while in Level 2 is required authenticated pairing with data signing.

There are four different pairing methods available in BLE: Out of Band, Passkey Entry, Just Works and Numeric Comparison. This last method is only available in low energy Secure Connections[39].

- *Out of Band:* the devices use a out of band (OOB) technology common to both, like NFC, to pair with each other;
- *Passkey Entry:* if one of the devices has input capability and the other device has a display, then this method can be used. In the device with a display, it's shown a 6 digit number that must be inserted in the input capable device;
- *Just Works:* if one the devices have neither input capability nor a display, the Just Works is used. It simply sets the temporary key (TK) used in during pairing to all zeros;
- *Numeric Comparison:* if both the devices have a display and some sort of input capability, this method can be used. A six digit number is shown in the displays, and the user must compare the values. If the values are the same, the user responds with a "yes", else, pairing is aborted.

For applications that require a high level of security, Numeric Comparison is the recommended method to use, since it provides eavesdropping and MITM protection[9].

2.2.3 Near Field Communication

Near Field Communication (NFC) is a technology for short-range wireless communications that enables simple and secure two-way interaction between two devices. It was established as a technology specification in 2002 by Sony and Philips, and in 2004, in conjunction with Nokia, it was formed the NFC Forum[3]. Like Bluetooth, NFC is integrated in almost every existing smartphone, and has been rapidly gaining a lot of use in IoT and Smart Environment sysmtems[16]. NFC is a subset of the Radio Frequency Identification (RFID) technology and it operates at the 13.56MHz frequency, the same band as the High Frequency (HF) RFID. One of NFC's main characteristics is its low communication range of only 5 centimeters on average. This small range also acts as a security measure for preventing eavesdropping attacks.

NFC works by bringing together two NFC-capable devices, in order for an interaction to occur. There are three different types of NFC devices: NFC tags, NFC readers and smartphones. The NFC protocol divides the communication modes in active or passive mode. In the active mode, both devices generate their RF fields to send data. In passive mode, only one device generate a RF field, while the other device retrieves the power from that field to send his data.

NFC is standardized by ECMA[1] and by ISO/IEC[2]. It follows the standards ISO/IEC 18092 / ECMA-340 - NFCIP-1 and ISO/IEC 21481 / ECMA-352 - NFCIP-2. These standards specify the modulation schemes, codings and transfer speeds of the RF interface, initialization schemes and data collision control, transport protocol, as well as the communication modes and their selection mechanism[18][19].

There are three different NFC operating modes, that vary accordingly to the devices used. In the Reader/Writer mode, the smartphone acts as an active device, and is the initiator of the interaction. It communicates with a NFC tag, that is a passive device, and can read or write to it. The RF layer in this communication mode is based on the ISO/IEC 14443 - Identification cards – Contactless integrated circuit cards standard[16].

Another operating mode is the Peer-to-Peer (P2P). This mode involves an interaction between two active-capable devices. They establish a two-way communication channel, used for exchanging any type of data.

The third operating mode is the Card Emulation mode. This mode allows a smartphone to behave as a smart card, enabling a NFC-reader to interact with it. The data that the NFC-reader reads from the smartphone is stored in a Secure Element (SE). A SE is a secure and controlled environment where sensitive data can be saved. Secure elements can be found in the smartphone's Subscriber Identity Module (SIM) card, or in an chip embedded into the smartphone's hardware[35].

NFC only by itself, is not capable of providing protection against eavesdropping and data manipulation[28]. Thus, each application that uses NFC must implement its own security measures and cryptograhic protocols. Another option is to use already implemented security standards for NFC, such as, for example, the ISO/IEC 13157-2 / ECMA-386 - NFC-SEC-01, that uses ECDH for key agreement and AES for data encryption[20], or the ISO/IEC 13157-4 / ECMA 410 - NFC-SEC-03 standard, that specifies key agreement and confirmation mechanisms that provide mutual authentication, using asymmetric cryptography[21].

An important setback with NFC is the Apple restrictive policies about the usage of NFC in their devices, since they only allows applications to detect and read NFC tags.

[1] https://www.ecma-international.org
[2] https://www.iso.org/isoiec-jtc-1.html

2.2.4 Wi-Fi Direct

Wi-Fi Direct is a Wi-Fi specification, develop by Wi-fi Alliance, that was first released in 2010. This protocol allows devices to create a Wi-Fi connection and communicate with each other, without the need for an Access Point (AP). In a normal Wi-Fi network, the devices connected to an AP can not communicate directly between them, instead, they interact through the AP. Wi-Fi Direct enables the devices to establish a direct connection, without requiring an AP[11].

After two devices discover each other, they have to determine who is going to be the group owner (GO), which will operate as an AP for that connection, and who is going to be the client. After this group is established, more devices can connect to it, as clients of the GO. Not all the connected devices need to be Wi-Fi Direct certified.

Wi-Fi Direct acquired several features from the traditional Wi-Fi, such as QoS, power saving, range of connection and transmission speeds[42].

Wi-Fi Direct also implements the Wi-Fi Protected Setup (WPS) for connections establishment. However, several flaws in the WPS implementation were found[45], making this protocol vulnerable to brute-force attacks.

The communications are encrypted by the Wi-Fi Protected Access II (WPA2) security protocol, that uses the CCM mode Protocol (CCMP).

It is possible to work with the Wi-Fi Direct in the Android mobile operating system, using the Wi-Fi Peer-to-Peer[3] framework. The same is not possible in the iOS operating system, since the peer-to-peer Wi-Fi implemented by iOS is not compatible with Wi-Fi Direct.

Comm. Protocol	Integrated Cryptography	Multi-platform	Range (meters)	Consumption (mA)	Bit rate (Mbps)
BLE 4.0 / 4.1	Yes	Yes	$\sim$ 1-10	<15	1
BLE 4.2	Yes	Yes	$\sim$ 1-10	<15	1
Wi-Fi Direct	Yes	No	100		250
NFC	No	No	0.05	<15	0.1 - 0.5

Table 2.1: Communication protocols comparison

Table 2.1 contains several characteristics from the communication protocols that were presented, with a distinction between the versions 4.0/4.1 and 4.2 of the BLE protocol, due to some important pairing and security features that were introduced in the version 4.2.

2.3 Physical access control systems

Physical access control systems (PACS) can be defined as a system that controls, manages and monitors the access to a physical resource, like a door, a room or a machine. A PACS needs to feature both authentication and authorization procedures. The authentication process verifies

[3]https://developer.android.com/guide/topics/connectivity/wifip2p.html

the identity of a person trying to get access to a resource, that is, it confirms if that person is who he is claiming to be. Then, the authorization process ensures that that person can in fact, access the specific resource.

It is relatively easy to find several recent and well implemented access control systems based on the NFC technology. However, it is much harder to find implementations that use Bluetooth, and almost every example is outdated and/or don't have any regards for the security that the system offers.

2.3.1 Bluetooth-based access control system

A Bluetooth access control system was implemented by Shadrack Mehlomakulu[36], where the user needs to enter a Personal Identification Number (PIN) in the smartphone in order to gain access to a physical space. The system's architecture consisted in a mobile phone, an access server, a database and a door controller. Initially, there is the Bluetooth handshake between the user the access server. After the connection is established, the user has to enter the pre-configured PIN in the mobile-phone. The PIN is sent to the Access Server, which checks the user's PIN and MAC address in the database. If the values match, it is sent a wired signal to the door controller to open the door. In addition to not being mentioned which Bluetooth version is used, there is also no discussion about the security of the system, if it even has any, or how it is achieved.

2.3.2 NFC-based access control system

Dominik Gruntz *et al.* implemented a mobile NFC-based physical access control system[26] that is divided in three main modules: a smartphone with NFC support, one or several Access points and a Central Access server. A key characteristic of this system is that there is no direct communication between the Access point and the Server. Instead, the Access Point uses the smartphone as a proxy to send it's messages and requests to the main Server. It is also allows an offline access, in the case that the smartphone can't communicate with the Server due to the lack of an internet connection.

The developed system is divided in two protocols: the authentication protocol, used to authenticate the smartphone to the access point, and the authorization protocol, used to send authorization data from the server to the access point. Prior to the authentication protocol, it's created a symmetric key known to both main Server and the Access Point, to symbolize the trust between them. The Authentication protocol is based on public key cryptography, and is accomplished with the use of NFC in it's entirety. The Access Point sends a nonce to the smartphone, which he signs with its private *RSA* key and sends back the signature. In the Authorization protocol, the communications between the Server and the smartphone are made through the network, using Transport Layer Security (TLS). First, the Access Point sends another nonce and its ID to the smartphone, via NFC, and then the smartphone sends those two values and also his certificate (that contains his ID) to the Server. The Server signs the

Access Point ID, the nonce, the smartphone's public key and the information that represents if the access was granted or not. This data is sent to the smartphone, that relays it to the Access Point. Now, using the smartphone's public key sent from the Server, the AP can conclude the Authentication protocol, and verify if the access was granted.

Although Dominik *et al.* used several tools and technologies in order to reduce its security risks, both the Authentication and Authorization protocols are still susceptible to attacks. Regarding the Authorization attacks, since it's used a TLS-secured connection, an attacker that controls the network can still read all the encrypted data sent between the Server and the smartphone. However, the manipulation of that data would be detected. If an attacker as control over the smartphone, it can also obtain information about the access rights of that smartphone, by sending multiple authorization requests to the Server. The main problem of the Authentication protocol is how it stores the secret key on the smartphone. In a simple software solution, if an attacker has system-level access, he can easily obtain the private key used to authenticate the smartphone. However, the introduction of hardware-based key stores in Android version 4.3, lead to some solutions that mainly resolved that problem.

Chapter 3

Proposed Solution

In this chapter, we will explain what our proposed solution is composed of, how it should work in a controlled environment, as well as why several decisions regarding its characteristics were made.

We are sure that using a smartphone is the right choice for our application. The smartphone's computational power has been rapidly increasing over the years, which meant that many actions and objects of our day-to-day life were simply replaced by a smartphone and their innumerable capabilities. A relevant example of this fact is the usage of a smartphone as the substitute of a smart card, like for example, a credit card or an access control card.

3.1 Overview

As previously stated in chapter 1, the main objective of this thesis is to design an application for a PACS, that uses a smartphone as its basis as a replacement of the insecure smart card. This is one more step of transforming the smartphone in our "smart-wallet", leading to the dematerialization of smart cards.

In a PACS, the object one is trying to gain access to, is, in most cases, a physical door. The same goes for our system, although the system could be applied to any physical object one would want to get access to, such as a vault, a machine or IoT devices.

We want a system that allows the user to gain access to a physical door, in a truly secure way but by simply clicking a button in his smartphone. In our case, the different buttons will represent different doors.

Two important system aspects that are needed to be taken into account are how secure and how usable the system must be. It is necessary to find a compromise between the two of them, meaning that we do not want an overly secure system that is hard and unpractical to use, neither want a very simple and functional system, that does not protect the infrastructure against attackers.

Our proposed application/system is composed by three distinct components, that communicate with each other in a specific order. The first component is the server, the second is the controller of the door we are trying to get access to, and the third is, obviously, the smartphone, the core component of the system. But even more important than the smartphone, is the mobile application developed specifically to work with this system. It is through this application that the user can see the available doors and choose which one he wishes to open.

Each door has a single controller associated with it, and it is that controller that the smartphone communicates with, and not with the door itself. Depending on whether or not the smartphone has access to the door, the controller opens the door or simply ignores the request.

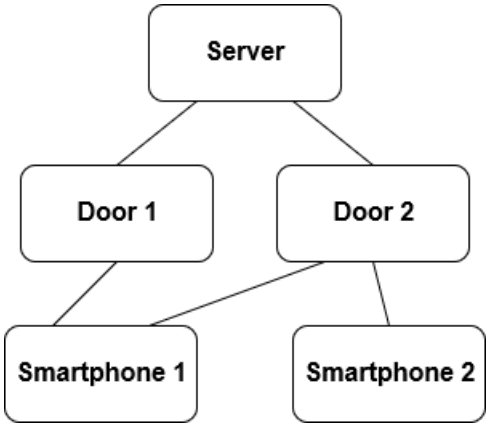


Figure 3.1: System components diagram (example)

Figure 3.1 contains a simple representation of the application's components, and how they can interact with each other. Apart from the main server, there are also represented two doors/controllers, as well as two smartphones.

In this figure, we can also observe an important characteristic that we want the system to have. Different users have different access privileges to the available doors, meaning that not all doors can be opened by every smartphone, neither every smartphone can open all the doors.

For example, in Figure 3.1, while the Smartphone 1 has access to both the available doors, the Smartphone 2 was only given permission to access Door 2. Even if it tries to open the Door 1, the server must detect that attempt and act accordingly, that is, not opening the door.

In order to prevent the attempt of gaining access to an unprivileged door, each smartphone only displays to the user the doors that it was given the permission to open. So, in the previous example, Smartphone 1 would display Door 1 and Door 2 to the user, while Smartphone 2 would only display Door 2.

3.2 System Functioning

The functioning of the system can be divided in two distinct phases: the Enrollment and the Authentication.

A flowchart of the communication that takes place during the two phases can be seen in Figure 3.2. In this section it is only explained which values are exchanged between the different components and how they are used. How the communication itself is achieved will be explained in the next section.

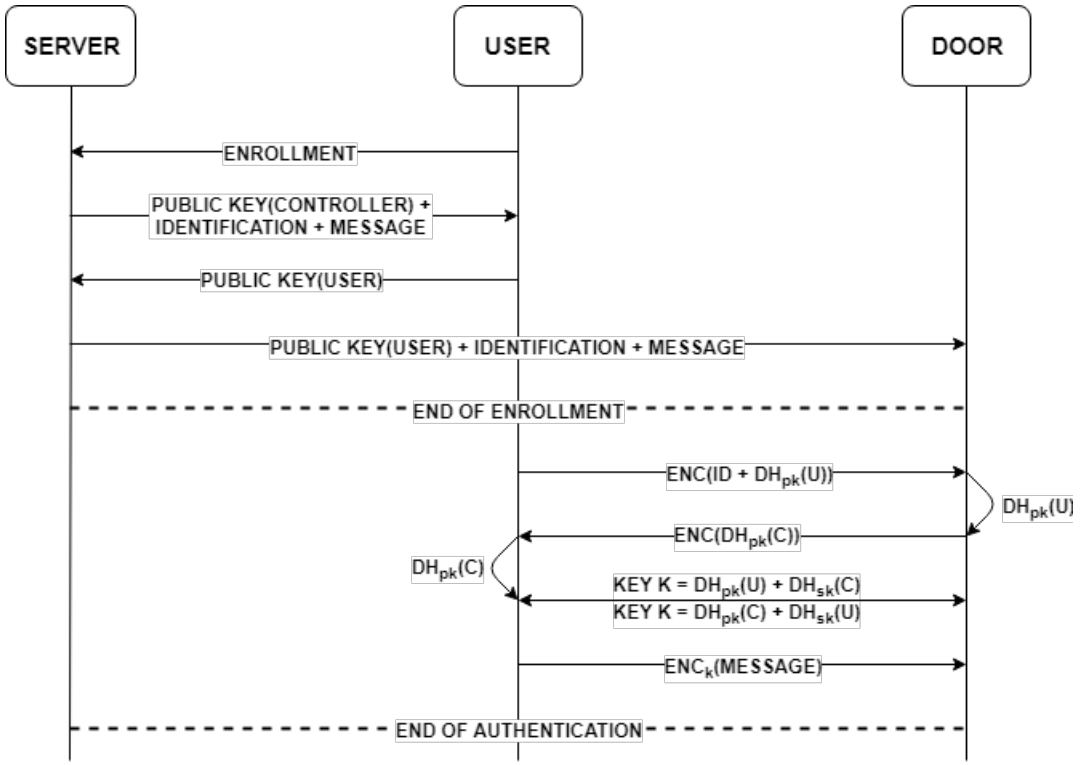


Figure 3.2: Communication sequence between components

3.2.1 Enrollment

The first contact the user has with the application is with the enrollment phase, that represents the registration of a new user in the system. In this phase, several pieces of information are shared between the server and the smartphone, as well as the controller, information that will later be used in the authentication process. Due to the importance of the information, the enrollment phase must be conducted in a secure environment, so that nobody can eavesdrop on the shared data.

When the application is first launched, but before the enrollment process itself begins, it is necessary to create a public/private key pair, both in the smartphone and in each door controller. These keys will be used to encrypt the messages between them, later in the authentication phase.

It is also important to note that each controller sends its public key to the server, meaning that the server knows the public key associated to each individual controller.

The enrollment process starts with the server generating a random identifier and a random secret message, that will represent the smartphone that is being registered. These two values will be used during the authentication phase, and they will serve to identify and authenticate the smartphone.

After the two values being created, the server then sends them to the smartphone, alongside with the public keys of the different doors that it will be able to gain access to. Upon receiving and storing this information, the smartphone shares its public key with the server.

In the last step of the enrollment, the server sends the received smartphone's public key and the created identifier and secret message to each one of the controllers that the smartphone will be able to communicate with. This way, each controller knows the public keys of the smartphones that are allowed to open their corresponding door, and they also have the identifier and secret message associated with each smartphone.

3.2.2 Authentication

When the application is opened, after the smartphone registration is already completed, the various doors for which the user can request access are displayed. The authentication process itself only starts when a user selects the door he wants to open, from the ones available in his smartphone.

While the enrollment phase involves mainly the server and the smartphone, the authentication procedure is performed solely by the smartphone and the controller, without any interaction with and from the server.

The objective of this phase is to create a secure channel between the two components, so that the smartphone can privately send the secret message, that it received during the enrollment phase, to the controller. In order to create the secure channel, we perform an ECDH key agreement protocol between the smartphone and the controller, thus creating a shared secret key known only by both these components.

So, before the authentication procedure begins, both the smartphone and the controller must create an ECDH public/private key pair.

In the first step of this phase, the smartphone takes its identifier and its ECDH public key, encrypts these two values together using the public key of the desired controller, and finally sends the encrypted value to said controller. Now the controller knows the ECDH public value from the smartphone, and since it also has the public key corresponding to the received identifier, it can encrypt its ECDH public key with the smartphone's public key and send it. Thus, both components can derive the same shared secret key K, using their own ECDH private key and the received ECDH public key.

Using the shared secret key K, the smartphone encrypts its secret message and sends it to the controller, which uses the same K key, to decrypt the received data. The last step is to confirm if the received secret message is correct, by comparing it with the secret message associated with the smartphone requesting the authentication. If so, the controller finally opens the door, thus ending the authentication procedure.

It is important that the ECDH values are never reused. So, at the end of each authentication phase, both participating components generate a new ECDH public/private key pair to be used in the future.

3.3 Communication methods

Another important aspect is how the communication between the different components is achieved. According to the needs of each one of the two phases, different communication protocols and technologies are used. The diagram presented in Figure 3.3 illustrates how the various components interact with each other.

3.3.1 Communication in Enrollment

During the enrollment phase, two different technologies are used for the components to communicate with each other, as it can be seen in the two different connections from the server in Figure 3.3.

The communication between the server and the smartphone uses a QR code as its core. After creating a new identifier and secret message for the smartphone, the server creates an URL that contains the data to be sent. Then, it generates a QR code from that URL, and makes it available for the smartphone to scan it. After the smartphone obtains the URL, it sends a *GET* request to it, in order to actually receive the data. Then, it sends a *POST* request to the same URL, containing its own public key. This implies that both the server and the smarthphone are connected to the same network.

Upon receiving this data, the server must distribute it within the different controllers that the smartphone will be allowed to access. There are two different ways that this communication could be achieved: via a network cable or by Wi-Fi. The network cable is the best choice, since it is simpler and safer to use. Using Wi-Fi means that we would have to find a way to make the communication secure.

As a safety feature, each URL generated by the server can be used once and only once. This way, any request other than the one from the registering smartphone, will not be accepted neither will receive any data.

3.3.2 Communication in Authentication

In order for the smartphone to "talk" with the controller during the authentication process, we must use a wireless communication protocol to establish a channel between those two components. As previously stated in 1.2, the system must respect a set of properties, meaning that we have to choose the communication protocol that better suits our needs. First, it is necessary to have a range of several meters, so that the user can request the access to the door without necessarily being near the door, or even if he does not have a line of sight to the door. Second, we want the system to be as generic as possible, so that it can be implemented in various platforms (Android and iOS). Finally, although not the most important property, the system must also have a low power consumption.

Modern smartphones are equipped with a wide variety of wireless communication protocols. The three most known and used communication protocols, which are in turn the possible choices for our system, are BLE, NFC and Wi-Fi Direct.

The NFC was excluded because of two reasons: it has a very limited range of just 5 to 10 centimeters and, second, it also misses the multi-platform property due to Apple restrictions on the usage of NFC within their products. As for the Wi-Fi Direct, although it has a fairly large range of up to 100 meters, it also lacks the compatibility in the Apple environment. This leaves us with BLE, which is actually our best option, since BLE fullfils all our requirements: it has a range of around 10 meters, it can be implemented both in Android and in iOS, and it has a low energy consumption.

Since BLE has several available versions, we needed to choose which one to use. BLE 4.2 was our final choice, instead of the two other versions, BLE 4.0 and BLE 4.1, because the 4.2 version has newer and more secure features, as previously stated in 2.2.2.

Despite BLE having a fairly decent range, it is possible that some doors may be too far from the smartphone for it to detect their signals, meaning that those doors will not appear as available in the smartphone. However, this situation can be seen as a positive feature, since it will not allow a smartphone to request access to a door that is relatively far away from it, which could represent a security problem.

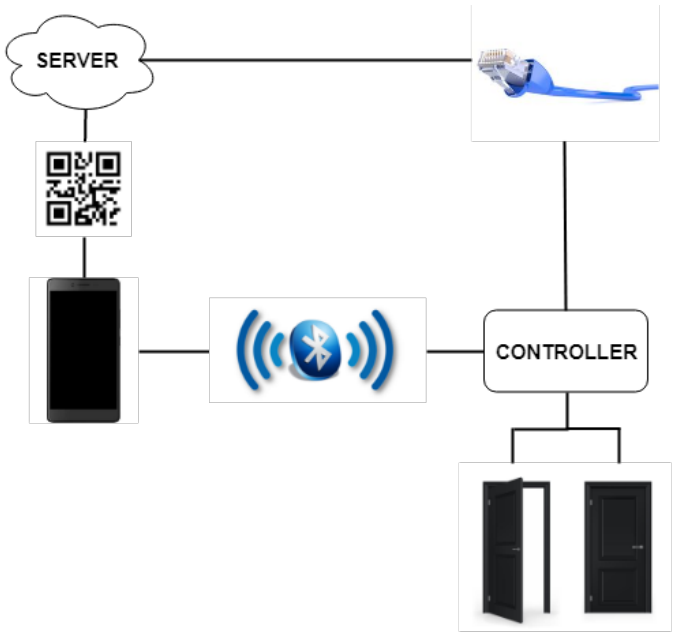


Figure 3.3: Communication protocols between components

Chapter 4

System Implementation

In addition to designing the application, we also set out to implement a first application prototype, purely for a proof-of-concept aspect. This chapter describes the implementation details and the steps taken in order to develop both the system and the Android application.

4.1 Overview

The main focus of this implementation was to develop and work primarily on the authentication procedure. We wanted to show that it is possible to create a secure channel between the smartphone and the controller, which can be used to communicate privately and to authenticate the user.

With that in mind, we assembled and implemented a system, as well as an Android application that interacts with that system, where the primary function is for a smartphone to select a specific Light-emitting diode (LED) in order to light it up. As previously stated in 3, a PACS is usually implemented to interact with physical doors. However, since this is a proof-of-concept implementation and we are more concerned with the application usability itself rather than with the actual physical process of opening the door, it is not relevant which physical object we use. So in our case, an authentication success and door opening is represented by lighting up a green LED, while an authentication failure is represented by lighting up a red LED.

In addition to this change, our developed application differs from our proposed solution presented in previous chapter 3, in the sense that some features were modified or simply not implemented. The most relevant change is related with the enrollment procedure, or the lack thereof. We decided, in a first phase, not to implement that procedure, and instead to focus only on the authentication aspect. Therefore, in our application, it is assumed that the smartphone registration has already been accomplished. However, in this registration, it is only assigned a single random identifier to each smartphone, instead of the identifier/secret message pair. This means that each controller only contains a list of various identifiers that are able to gain "access" to the LED, and during the authentication process it checks to see if the received identifier is

present in that list.

A final difference from the proposed solution to this prototype is that each smartphone displays all the available LEDs, regardless of whether it has access to it or not, in order to show the authentication process detecting a non compliant request.

4.2 Material Used/System Architecture

Our implemented system consists of two main components: a smartphone and a controller. However, the controller can be divided into two individual elements, them being an Arduino Uno board and a BLE module.

The BLE module we choose for this project was the RedBear's BLE Nano 2. This module is equiped with a Nordic nRF52832 BLE System-On-Chip (SoC), and can be programmed using the Nordic's nRF5 SDK, mbed and Arduino[1]. In our case, we programmed it in Arduino, using an open source library[1] for the nRF52832 chip, developed by the RedBear team. Besides its very small size, it is also BLE 4.2 certified, which is precisely the BLE version we previously decided to use.

The Android application was developed and tested with an Alcatel A7 smartphone, with version 7.0 (Nougat) of the Android operating system.

The full list of material used to develop this system is summarized in the Table 4.1

Arduino Uno
RedBear BLE Nano 2
Smartphone (Alcatel A7)
Breadboard
LEDs
Jump wires
Resistors

Table 4.1: Material used in the implementation

The Arduino Uno board and the BLE Nano 2 module form a circuit between them, in which the green and red LEDs associated with that controller are also connected. Figure 4.1 presents the schematic representation of the circuit with all the connections between the various components.

The authentication process happens between the smartphone and the Arduino. So, as expected, the smartphone communicates via BLE with the BLE Nano 2, which, in turn, communicates with the Arduino Uno via its Serial port, and vice-versa. The BLE Nano 2 module simply acts as a middle-man between those two components.

[1]This library is available at https://github.com/redbear/nRF5x

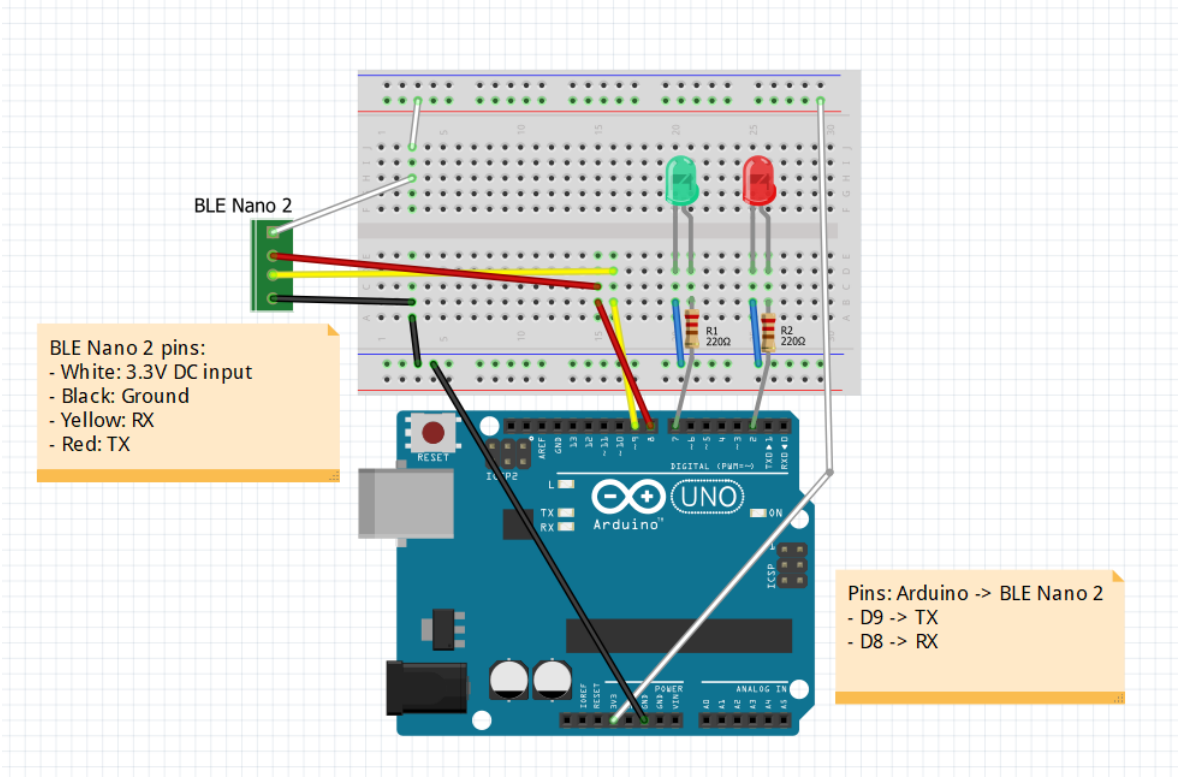


Figure 4.1: Schematic representation of the Arduino and BLE Nano 2 circuit

4.3 System Operation

All the three components (smartphone/Android application, Arduino Uno and BLE Nano 2) operate differently from one another. In this section, it is explained how each one of them was developed and how they function within the system.

In figure 4.5 it is represented the sequence of steps, functions and communications established during the authentication process. The content of this diagram will be explained throughout the next sections.

4.3.1 Android application

4.3.1.1 Features and permissions

Since this application revolves around the usage of BLE, several permissions and features need to be declared for it to work properly. The **BLUETOOTH** permission is required, in order to perform any Bluetooth communication at all, while the **BLUETOOTH_ADMIN** permission allows the application to start device discovery.

To ensure that the target Android smartphone is BLE capable, the **bluetooth_le** feature is declared with the *required* attribute set to true.

Due to BLE beacons being associated with physical location applications[40], this application also needs the **ACCESS_COARSE_LOCATION** permission, otherwise the BLE device discovery will not return any results. The first time the application is opened, it asks the user to grant this permission to it. If the user do not accept the request, the application become unusable until the permission is granted. In the following uses, the application always checks if it still has the permission.

4.3.1.2 Operation

The application itself starts by verifying if Bluetooth is enabled. If it is not, the application requests the user to enable it first. Only then the application is able to start the device discovery, by scanning for nearby devices for about 3 seconds. For each device found, the callback function represented in Listing 4.1 is triggered. First, it checks if the found device is already present in the device list. If not, it simply checks if its name is not "null", before adding it to the list.

If no device is discovered during this process, the application displays a message alerting the user. Otherwise, it displays the list of devices, as it is possible to see in Figure 4.2.

This procedure can later be executed again, in order to update the device's list, by clicking in the **Scan Again** button, which is also visible in the application screenshot in Figure 4.2.

```
private final ScanCallback leScanCallback = new ScanCallback() {
    @Override
    public void onScanResult(int callbackType, final ScanResult device)
        {
        runOnUiThread(new Runnable() {
            @Override
            public void run() {
                if(!bleDevicesList.contains(device.getDevice())) {
                    String device_name = device.getDevice().getName();
                    if(device_name == null){
                        return;
                    }
                    mLeDeviceListAdapter.add(device.getDevice());
                    mLeDeviceListAdapter.notifyDataSetChanged();
                }
            }
        });
    }
};
```

Listing 4.1: Device scan callback

When an item from that list is clicked, a new authentication request begins. First the smartphone needs to establish a connection with the selected BLE Nano 2, by sending a

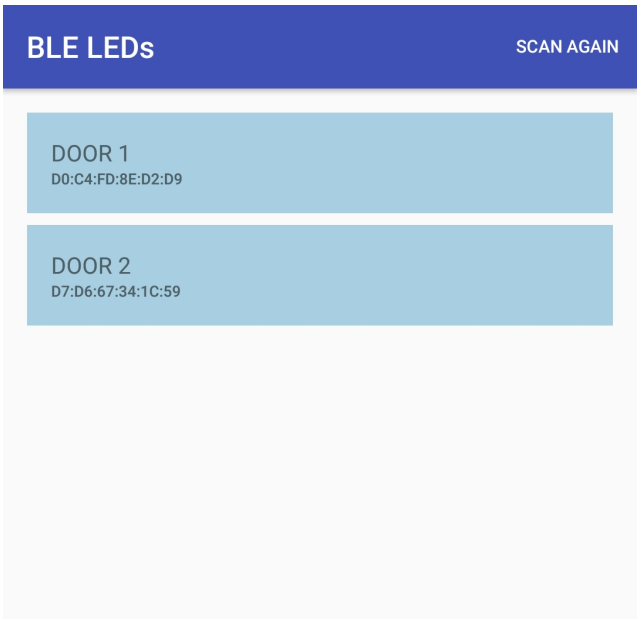


Figure 4.2: Android application's list of found devices

connection request to it. Again, a callback function, present in Listing 4.2, awaits the response from the module. If the connection is accepted, and if it currently does not have a bond established with that device, a pairing/bonding request is sent to it. Otherwise, if it already has a bond with the device, the application goes directly to the next step. If either the connection or the bonding request fails, the current user authentication process is canceled.

```
public void onConnectionStateChange(BluetoothGatt gatt, int status, int
    newState) {
     String intentAction = ACTION_DISCONNECT;
     if (newState == BluetoothProfile.STATE_CONNECTED) {
          if(getBondState() ==  BluetoothDevice.BOND_BONDED){
               intentAction = ACTION_GATT_CONNECTED_BONDED;
          }
          else {
               device.createBond();
               intentAction = ACTION_GATT_CONNECTED_BONDING;
          }
          broadcastUpdate(intentAction);
     } else if (newState == BluetoothProfile.STATE_DISCONNECTED) {
          broadcastUpdate(ACTION_GATT_DISCONNECTED);
     }
}
```

Listing 4.2: Device connection status callback

The next step, after the connection and bonding requests are established, is to generate the public and private ECDH keys, by using the Curve25519 elliptic curve/function. So, in order to

create the public/private key pair, we used the *ecdh-curve25519-mobile* library[2].

The public key, which is represented in a byte array, is then encoded using Base64 into a new String object. Without this step, some of the public key's bytes could represent non-printable ASCII characters, which would make it difficult to transmit the public key to the Arduino. So, encoding it using Base64 assures that the Arduino receives all the characters in a readable way.

```
SecureRandom random = new SecureRandom();
byte[] secretKey = ECDHCurve25519.generate_secret_key(random);
byte[] publicKey = ECDHCurve25519.generate_public_key(secretKey);
String publicBase64 = Base64.encodeToString(publicKey, Base64.DEFAULT);
```

Listing 4.3: Android Curve25519 keys generation

After sending it, the application waits for the Arduino to send its public key. The data received is then decoded using Base64, in order to actually obtain the Arduino's public key. With this value and the smartphone's private key previously created, it is possible to generate the shared secret key, again using the *ecdh-curve25519-mobile* library.

```
private static void getArduinoPublicKey(String encodedPubKey){
    arduinoPubKey = Base64.decode(encodedPubKey.getBytes(),
        Base64.DEFAULT);
}

private static void generateSharedSecret(){
    sharedSecretKey = ECDHCurve25519.generate_shared_secret(secretKey,
        arduinoPubKey);
}
```

Listing 4.4: Decoding Arduino public key and generating shared secret key

In the final step to complete the user's authentication request, the smartphone's identifier is encrypted. For that, we use the AES-GCM symmetric key cipher, available in Java's *javax.crypto.Cipher* class, that encrypts the identifier with the previously generated shared secret key. Since the AES-GCM cipher uses a random initialization vector, which is needed for both the encryption and decryption, it is also necessary to send it to the Arduino. So, the initialization vector is agglomerated with the encrypted identifier, before they are encoded together using Base64. This data is then finally sent to the Arduino.

```
SecureRandom secureRandom = new SecureRandom();
byte[] initVector = new byte[12];
secureRandom.nextBytes(initVector);
SecretKey secretKey = new SecretKeySpec(shared_secret_key, "AES");
final Cipher cipher = Cipher.getInstance("AES/gcm/NoPadding");
GCMParameterSpec parameterSpec = new GCMParameterSpec(128, initVector);
```

[2]This library is available at https://github.com/duerrfk/ecdh-curve25519-mobile

```
cipher.init(Cipher.ENCRYPT_MODE, secretKey, parameterSpec);
byte[] cipheredMessage = cipher.doFinal(secretMessage.getBytes());
ByteBuffer byteBuffer = ByteBuffer.allocate(initVector.length +
    cipheredMessage.length);
byteBuffer.put(initVector).put(cipheredMessage);
String encodedBuffer = Base64.encodeToString(byteBuffer.array(),
    Base64.DEFAULT);
```

Listing 4.5: Encrypting random identifier with AES-GCM

The connection with the BLE Nano 2 is terminated, thus ending the authentication procedure.

Figure 4.3 represents the application activity diagram, that summarizes the application operation that was explained during this section.

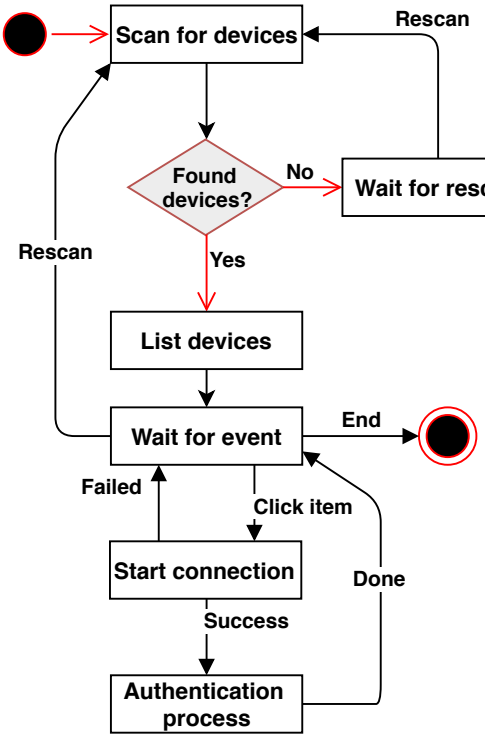


Figure 4.3: Android application activity diagram

4.3.2 BLE Nano 2

Since the code for BLE Nano 2 was implemented using Arduino, its code is divided in two parts: the initial setup, and the main loop. The initial setup only happens when the BLE Nano 2 is turned on. After that, only the main loop is executed.

The setup is mainly dedicated to initializing and configuring the BLE settings. Some of the defined values and characteristics are: the device's name that will be presented to the smartphones

during their scanning process, the power of the radio transmission and a characteristic that states that it only supports BLE connections and not the "classic" Bluetooth.

At the end of the setup, the module starts advertising itself, meaning that the smartphones are now able to find the device during their scanning.

The connection and bonding requests sent by the smartphone in the beginning of a new authentication process are treated directly by the BLE Nano 2 itself, and not by our implemented code. It is important to note that when a connection is established, the BLE Nano 2 stops its advertisement, since it can only have one active connection simultaneously. This means that during an authentication process, the other smartphones will not be able to scan it.

After the setup is done, the BLE Nano 2's work is fairly simple, since it only needs to retransmit the received data. Upon receiving the first message from the smartphone, containing its encoded ECDH public key, the BLE Nano 2 sends this data through its Serial connection with the Arduino. After that, it waits for the Arduino to send its encoded ECDH public key, to then relay the data to the smartphone. At this point, the public keys have been shared, and the two components can generate the shared secret key.

The second message received from the smartphone contains its identifier encrypted with the shared secret key. Again, the BLE Nano 2 sends this data to the Arduino, thus ending its role as the middle-man.

The module only needs to wait for the smartphone to disconnect, before waiting again for a new smartphone's authentication request.

4.3.3 Arduino Uno

In the same way as the BLE Nano 2, the Arduino code is divided into the initial setup and the main loop. However, its setup is relatively simpler since it only generates the Curve25519 public/private keys, which will be used in the first authentication request. To create the key pair, and later generate the shared secret key, it was used the *Curve25519* class from the *Crypto* library[3].

```
void generateECDHkeyValues(){
    Curve25519::dh1(arduino_public_key, arduino_secret_key);
}
```

Listing 4.6: Arduino Curve25519 keys generation

As previously stated, in this prototype we are assuming that the enrollment procedure was already accomplished. For that, each Arduino has a specific list containing the smartphone identifiers for which authentication requests will be accepted.

[3]This library is available at https://rweather.github.io/arduinolibs/crypto.html

After the initial setup, the Arduino simply waits until it receives any data in its Serial port connected to the BLE Nano 2.

The first received message contains the smartphone's encoded public key. Before decoding the data using Base64 to obtain the actual public key, it immediately sends its public key to the BLE Nano 2, obviously encoded with Base64. Only then, it generates the shared secret key.

```
void decodeSmartphoneKey(char* sp_pub_key, int sp_pub_key_size){
    int decodedLen = base64_dec_len(sp_pub_key, sp_pub_key_size);
    char decoded[decodedLen+1];
    base64_decode(decoded, sp_pub_key, sp_pub_key_size);
}

void generateSharedSecret(){
    Curve25519::dh2(smartphone_public_key, arduino_secret_key);
}
```

Listing 4.7: Decoding Smartphone public key and generate shared secret key

The second message received by the Arduino includes the smartphone's encrypted identifier, together with initialization vector used by it during the encryption. First, the data is decoded using Base64, and then, using the *AES* and *GCM* classes, from the previously mentioned *Crypto* library, it is performed an AES-GCM decryption on the received data, using the proper initialization vector.

```
uint8_t iv[12];
int decodedLen = base64_dec_len(enc_msg, enc_msg_size);
char decoded[decodedLen+1];
base64_decode(decoded, enc_msg, enc_msg_size);
for(int j=0; j<12; j++){
    iv[j] = decoded[j];
}
uint8_t decryptedBuffer[20];
GCM<AES256> gcm;
gcm.setKey(smartphone_public_key, 32);
gcm.setIV(iv, 12);
gcm.decrypt(decryptedBuffer, &decoded[12], decodedLen);
```

Listing 4.8: AES-GCM decryption of smartphone's identifier

The final step is to verify if the received identifier is present in the Arduino's list. If, in fact, it is, the Arduino lights up the green LED connected to it, as it is possible to observe in Figure 4.4, showing the user that the authentication request was accepted. If not, the Arduino instead lights up the red LED.

At the end of each authentication procedure, the Arduino generates a new pair of Curve25519 keys, to be used in the next request.

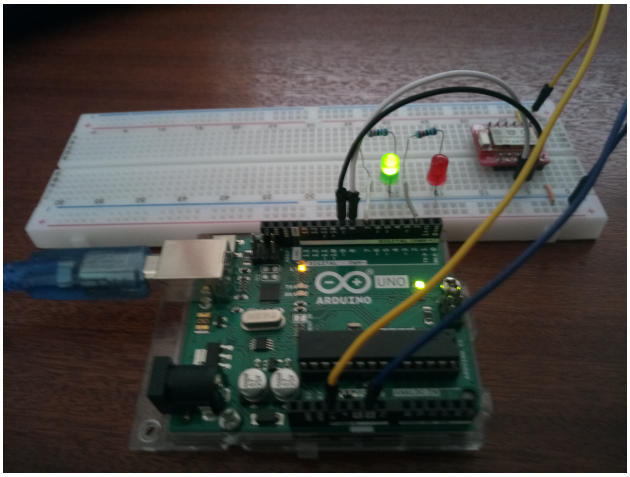


Figure 4.4: Arduino and BLE Nano 2 with green LED lighted up

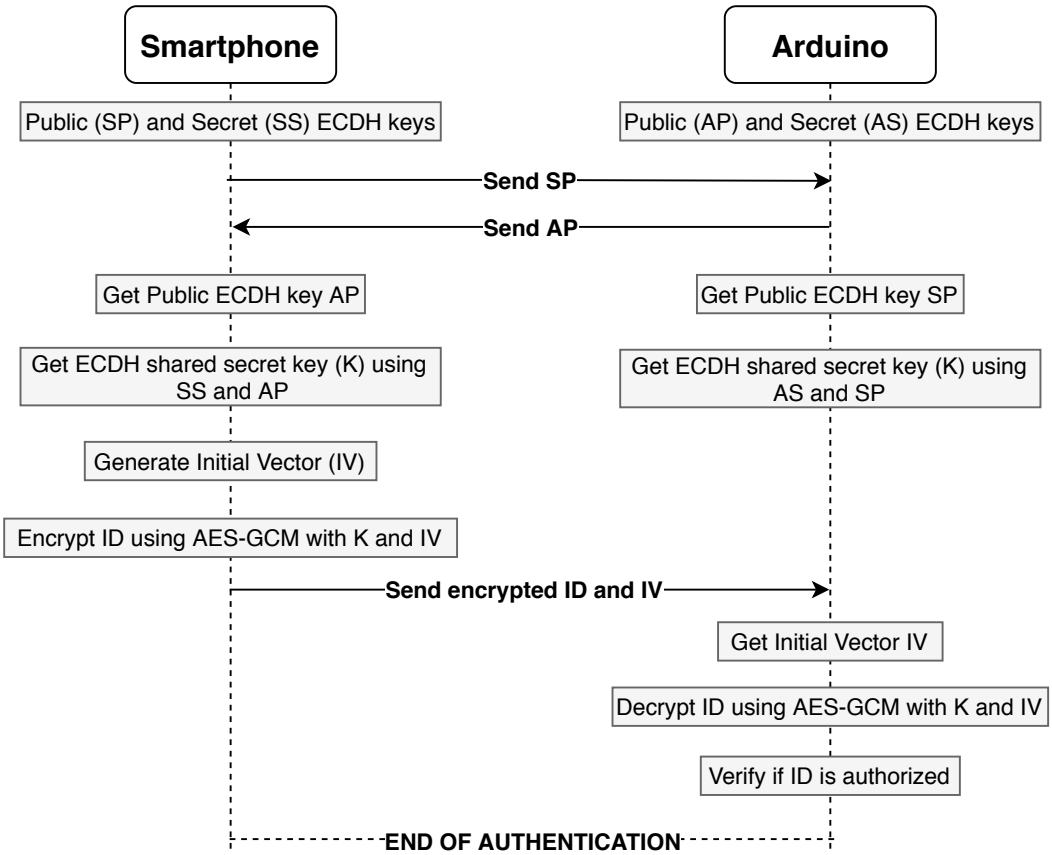


Figure 4.5: Communication sequence between the Arduino and the Smartphone

4.4 Problems found

During the implementation of the different components of the system, several problems and setbacks were found and were necessary to deal with.

The Maximum Transmission Unit (MTU) represents the maximum data size that each message can carry. The BLE Nano 2 uses a default MTU value of 23 bytes, meaning that each BLE packet can transmit only 20 bytes of actual data since 3 of those bytes are reserved. Unfortunately, during this implementation, we could not find a way to increase that value. This means that each "message" sent between the smartphone and the BLE Nano 2, is actually a set of messages of 20 bytes each that make up the full data.

Another problem found is related with the Arduino itself. The Arduino Uno has a relatively limited amount of SRAM memory, of just 2 KB. During our implementation, the Arduino often reached the SRAM limit, meaning that it would crash and restart itself. However, we were able to solve this, by simply storing each received message in a predefined size *char* array, instead of storing it in a *String* object.

The Arduino Uno has another drawback, which is its low computational power. The Curve25519 public/private key pair and shared secret key generation, represented in Listings 4.6 and 4.7, respectively, takes about three seconds each to compute. The AES-GCM decryption, shown in 4.8, also takes almost one second. All these seconds, together with the time for the smartphone to connect and bond with the BLE Nano 2, make the authentication process relatively time consuming.

In the final stage of the implementation, it emerged a problem that changed substantially the behavior of the system. The Android application started working only once, in the sense that after a single authentication request, successful or not, it was necessary to restart the smartphone for the system to work again. We could not find the cause of this problem. The only workaround that we discovered was for the smartphone to not engage in the bonding process with the BLE Nano 2. So, after the connection is accepted, the smartphone starts the communication right away, instead of sending first a bonding request. This way, the application works as intended.

Chapter 5

Conclusion and Future Work

As Cabral stated in his thesis, near all PACS based on smartcards are not secure and can be easily exploited[10]. With that in mind, we set on to design a PACS based on a smartphone/Android application, with the intent of creating a secure and reliable alternative to smartcards, that would authenticate and authorize the user to access a physical door. That system would be composed by three main components: the server, the controller, and the smartphone. It would allow a user to enroll himself in the system, before being able to start the authentication process with a door.

Despite not having been possible to implement the designed system in its entirety, mainly referring to the enrollment phase, we showed that it is possible to implement a PACS, using BLE as the communication protocol, and using an Arduino Uno as the controller, that successfully creates a secure channel between the smartphone and the controller, allowing them to privately share information. Although some obstacles were encountered during this implementation, the system still complies with the previously defined requirements: multi-platform compatible, secure cryptographic protocols, reach of multiple meters and a low energy consumption.

Future work includes, obviously, implementing the server component, together with the enrollment phase, and subjecting the system to an usability test and security analysis, and also evaluating an implementation in a controlled environment.

We suggest, in a future implementation, changing the Arduino Uno for a more powerful board, like an Arduino Due or a Raspberry Pi, which would help reducing the time to generate the Curve25519 public/private key pair and shared secret key, as well as the AES-GCM decryption time.

Bibliography

[1] Bluetooth 5 ready: Ble module, nano 2 & blend 2 [online]. Nov 2016. [Online; accessed March 22, 2018].

[2] Developer reference for intel® integrated performance primitives cryptography 2018 [online]. Nov 2017. [Online; accessed August 20, 2018].

[3] Tomi Aarnio. Near field communication using nfc to unlock doors. Master's thesis, Aalto University School of Science, Espoo, 2013.

[4] Mehmet Adalier. Efficient and secure elliptic curve cryptography implementation of curve p-256.

[5] Tarun Agarwal. Overview on electronic communication protocols [online]. n.d. [Online; accessed January 06, 2018].

[6] aleden. Is the bijection of scaling a point of an elliptic curve, an isomorphism? Mathematics Stack Exchange.

[7] Daniel J Bernstein. Curve25519: new diffie-hellman speed records. In *International Workshop on Public Key Cryptography*, pages 207–228. Springer, 2006.

[8] Alex Biryukov, Orr Dunkelman, Nathan Keller, Dmitry Khovratovich, and Adi Shamir. Key recovery attacks of practical complexity on aes variants with up to 10 rounds. Cryptology ePrint Archive, Report 2009/374, 2009. https://eprint.iacr.org/2009/374.

[9] Matthew Bon. A basic introduction to ble security [online]. 2016. [Online; accessed January 14, 2018].

[10] Pedro Cabral. Rfid clone. Master's thesis, Faculdade de Ciências da Universidade do Porto, Porto, 7 2017.

[11] Daniel Camps-Mur, Andres Garcia-Saavedra, and Pablo Serrano. Device-to-device communications with wi-fi direct: overview and experimentation. *IEEE wireless communications*, 20(3):96–104, 2013.

[12] Michael Cobb and Borys Pawliw. Advanced encryption standard (aes) [online]. 2014. [Online; accessed November 29, 2017].

[13] Michael Cobb, Fred Hazan, and Frank Rundatz. Rsa algorithm (rivest-shamir-adleman) [online]. Nov 2014. [Online; accessed November 28, 2017].

[14] Andrea Corbellini. Elliptic curve cryptography: finite fields and discrete logarithms [online]. May 2015. [Online; accessed January 16, 2018].

[15] Andrea Corbellini. Elliptic curve cryptography: Ecdh and ecdsa [online]. May 2015. [Online; accessed September 14, 2018].

[16] Vedat Coskun, Busra Ozdenizci, and Kerem Ok. The survey on near field communication. *Sensors*, 15(6):13348–13405, 2015.

[17] Whitfield Diffie and Martin Hellman. New directions in cryptography. *IEEE transactions on Information Theory*, 22(6):644–654, 1976.

[18] Standard ecma-340 near field communication interface and protocol (nfcip-1) [online]. June 2013. [Online; accessed January 08, 2018].

[19] Standard ecma-352 near field communication interface and protocol -2 (nfcip-2) [online]. June 2013. [Online; accessed January 08, 2018].

[20] Nfc-sec-01: Nfc-sec cryptography standard using ecdh and aes [online]. June 2015. [Online; accessed January 08, 2018].

[21] Nfc-sec-03: Nfc-sec entity authentication and key agreement using asymmetric cryptography [online]. June 2017. [Online; accessed January 08, 2018].

[22] Rupa Ganjewar. Diffie hellman key exchange.

[23] Daniel Genkin, Adi Shamir, and Eran Tromer. Rsa key extraction via low-bandwidth acoustic cryptanalysis. In *International Cryptology Conference*, pages 444–461. Springer, 2014.

[24] Henri Gilbert and Thomas Peyrin. Super-sbox cryptanalysis: Improved attacks for aes-like permutations. Cryptology ePrint Archive, Report 2009/531, 2009. https://eprint.iacr.org/2009/531.

[25] Carles Gomez, Joaquim Oller, and Josep Paradells. Overview and evaluation of bluetooth low energy: An emerging low-power wireless technology. *Sensors*, 12(9):11734–11753, 2012.

[26] Dominik Gruntz, Christof Arnosti, and Marco Hauri. Moonacs: a mobile on-/offline nfc-based physical access control system. *International Journal of Pervasive Computing and Communications*, 12(1):2–22, 2016.

[27] Darrel Hankerson, Alfred J Menezes, and Scott Vanstone. *Guide to elliptic curve cryptography*. Springer Science & Business Media, 2006.

[28] Ernst Haselsteiner and Klemens Breitfuß. Security in near field communication (nfc). In *Workshop on RFID security*, pages 12–14, 2006.

[29] Scott Helme. Perfect forward secrecy - an introduction [online]. May 2014. [Online; accessed September 18, 2018].

[30] Clinton Francis Hughes. *Bluetooth Low Energy*. PhD thesis, Arizona State University, 2015.

[31] Thorsten Kleinjung, Kazumaro Aoki, Jens Franke, Arjen Lenstra, Emmanuel Thomé, Joppe Bos, Pierrick Gaudry, Alexander Kruppa, Peter Montgomery, Dag Arne Osvik, et al. Factorization of a 768-bit rsa modulus. In *CRYPTO 2010*, volume 6223, pages 333–350. Springer, 2010.

[32] Neal Koblitz. Elliptic curve cryptosystems. *Mathematics of computation*, 48(177):203–209, 1987.

[33] Stefan Lemsitzer, Johannes Wolkerstorfer, Norbert Felber, and Matthias Braendli. Multi-gigabit gcm-aes architecture optimized for fpgas. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 227–238. Springer, 2007.

[34] Yue Liu, Ju Yang, and Mingjun Liu. Recognition of qr code with mobile phones. In *Control and Decision Conference, 2008. CCDC 2008. Chinese*, pages 203–206. IEEE, 2008.

[35] Ganeshji Marwaha. Mobile payments: What is a secure element? [online]. September 2014. [Online; accessed January 08, 2018].

[36] Shadrack Mehlomakulu. *Bluetooth Access Control*. PhD thesis, University of the Western Cape, 2010.

[37] Victor S Miller. Use of elliptic curves in cryptography. In *Conference on the Theory and Application of Cryptographic Techniques*, pages 417–426. Springer, 1985.

[38] Message authentication codes [online]. Feb 2017. [Online; accessed January 08, 2018].

[39] John Padgette. Guide to bluetooth security. *NIST Special Publication*, 800:121, 2017.

[40] Bluetooth Low Energy BLE Permissions [online]. [Online; accessed March 28, 2018]. [link].

[41] Bruce Schneier. *Applied Cryptography (2Nd Ed.): Protocols, Algorithms, and Source Code in C*. John Wiley & Sons, Inc., New York, NY, USA, 1995. ISBN: 0-471-11709-9.

[42] Wenlong Shen, Bo Yin, Xianghui Cao, Lin X Cai, and Yu Cheng. Secure device-to-device communications over wifi direct. *IEEE Network*, 30(5):4–9, 2016.

[43] Gurpreet Singh. A study of encryption algorithms (rsa, des, 3des and aes) for information security. *International Journal of Computer Applications*, 67(19), 2013.

[44] Advanced encryption standard [online]. n.d. [Online; accessed January 03, 2018].

[45] Stefan Viehböck. Brute forcing wi-fi protected setup. *Wi-Fi Protected Setup*, 9, 2011.

[46] Doug Whiting, Niels Ferguson, and Russell Housley. Counter with cbc-mac (ccm), 2003.

[47] Junfeng Xu, Tao Zhang, Dong Lin, Ye Mao, Xiaonan Liu, Shiwu Chen, Shuai Shao, Bin Tian, and Shengwei Yi. Pairing and authentication security technologies in low-power bluetooth. In *Green Computing and Communications (GreenCom), 2013 IEEE and Internet of Things (iThings/CPSCom), IEEE International Conference on and IEEE Cyber, Physical and Social Computing*, pages 1081–1085. IEEE, 2013.