



**TÉCNICO**  
LISBOA

# **Dynamic QR codes for Ticketing Systems**

**José Miguel Faustino Oliveira**

Thesis to obtain the Master of Science Degree in

## **Information Systems and Computer Engineering**

Supervisor: Prof. José Manuel da Costa Alves Marques

### **Examination Committee**

Chairperson: Prof. Paolo Romano

Supervisor: Prof. José Manuel da Costa Alves Marques

Member of the Committee: Prof. Miguel Nuno Dias Alves Pupo Correia

**November 2021**



# Acknowledgments

First of all I would like to thank my supervisors Prof. José Manuel Alves Marques, Eng. Joel Teixeira and Eng. Hugo Bicho. Their knowledge and experience were very appreciated and they provided me the right guidance to complete this work.

I would like to thank all my friends and those who accompanied me in this journey. They encouraged me to never give up and provided tons of moments which I will cherish forever, even at difficult times.

Last but not least, I would like to thank my parents and sister for their love and sacrifice. All I am is thanks to them, and without whom this project would not be possible. I would also like to thank my grandparents, aunts, uncles and cousins for their understanding and support throughout all these years.

My most sincere thanks to each and every one of them.



# Abstract

Mobile phones are invaluable systems of our society. With the increasing growth in use of mobile phones, emerging technologies are constantly being introduced and the development of technologies, such as barcodes, contributed to the rapid widespread of mobile phones over various systems, including ticketing systems. Mobile ticketing is a subset of ticketing systems and it provides a simple alternative to the use of physical tickets. Mobile ticketing makes use of mobile tickets which are digital tickets in smartphones and they have the same functionalities as their predecessors such as tickets in smartcards.

By taking advantage of these technologies, it is proposed a mobile ticketing system which performs the offline validation of tickets, in the form of QR codes. The proposed system contains a mobile application which can be used to purchase and present tickets, in the form of QR codes. Tickets are purchased from the backend server and validated in ticket validators. Ticket validators perform the offline validation of tickets and therefore, in instances where ticket validators do not have access to the internet, they do not lose functionality and can continue performing the correct validation of tickets.

This paper presents insights over existing ticketing systems, the architecture of the system and technologies related to the implementation. The details of the implementation are discussed extensively as well as the potential security issues. For the latter, there are technologies that were implemented in order to prevent against fraud. Furthermore, this paper contains the evaluation to the implementation and results that could support the reliability and meet the proposed requirements of this system.

## Keywords

Ticketing systems; mobile ticketing; QR codes; one-time passwords; public transports



# Resumo

O aumento no número de utilizadores de smartphones levou ao desenvolvimento de tecnologias como os códigos de barras nos smartphones e que contribuem para a adoção dos smartphones em novas áreas, como os sistemas de bilhética. O mobile ticketing é um tipo de sistema de bilhética e é visto como uma alternativa simples em relação aos sistemas de bilhética mais usados atualmente. O mobile ticketing faz uso de mobile tickets que são bilhetes em smartphones, com as mesmas funcionalidades que os seus predecessores, como bilhetes em smartcards.

Neste documento é proposto um sistema de bilhética móvel que faz uso de bilhetes em formato QR code. Este sistema permite que os utilizadores façam uso de uma aplicação móvel para que possam comprar e apresentar bilhetes. Os bilhetes podem ser comprados a um servidor, que está no backend, e podem ser validados nos validadores de bilhetes. Dado que os validadores fazem a validação offline dos bilhetes, em caso de falta de acesso à internet, é necessário que os validadores não percam a sua funcionalidade e que possam continuar a fazer uma validação correta dos bilhetes.

Este documento apresenta dados acerca de sistemas de bilhética existentes, a arquitetura do sistema e tecnologias relacionadas com a implementação. Os detalhes da implementação são apresentados e avaliados extensivamente bem como as ameaças ao sistema. Para combater as ameaças ao sistema, foram implementados métodos que permitem prevenir os possíveis ataques à solução e as tecnologias que contribuiram para o desenvolvimento do sistema. Além disso, o documento contém uma avaliação à implementação e os resultados que permitem assegurar que o sistema é confiável e seguro.

## Palavras Chave

Sistemas de bilhética; mobile ticketing; QR codes; one-time passwords; transportes públicos





# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Background . . . . .	3
1.2	Research Objective . . . . .	4
1.2.1	Fraud Prevention . . . . .	4
1.2.2	Interoperability . . . . .	4
1.2.3	Convenience . . . . .	4
1.2.4	Offline Access . . . . .	5
1.3	Thesis overview . . . . .	5
<b>2</b>	<b>Related Work</b>	<b>7</b>
2.1	Ticketing Systems . . . . .	9
2.1.1	Entrance and Exit . . . . .	10
2.2	QR Codes . . . . .	11
2.3	Cryptography . . . . .	12
2.3.1	Symmetric Keys . . . . .	13
2.3.2	Asymmetric Keys . . . . .	13
2.3.2.A	Hybrid Encryption . . . . .	14
2.3.2.B	Elliptic Curve Cryptography . . . . .	14
2.4	Hashing Algorithms . . . . .	15
2.4.1	Secure Hash Algorithms . . . . .	15
2.4.2	Hash-based Message Authentication Code . . . . .	16
2.5	One-Time Passwords . . . . .	16
2.5.1	HMAC-based One-Time Passwords . . . . .	17
2.5.2	Time-based One-Time Passwords . . . . .	17
2.6	Digital Signatures . . . . .	18
2.7	OpenID Connect . . . . .	19
<b>3</b>	<b>Architecture</b>	<b>21</b>
3.1	Overview . . . . .	23

3.2	Use Cases . . . . .	24
3.2.1	Ticket Acquisition . . . . .	24
3.2.2	Ticket Validation . . . . .	25
3.3	Security and Fraud Prevention . . . . .	26
3.3.1	Digital Signatures . . . . .	26
3.3.2	One-time Passwords . . . . .	26
3.3.3	Validation History . . . . .	27
3.3.4	Asymmetric Keys . . . . .	27
3.3.5	Types of Tickets . . . . .	28
3.3.5.A	Tickets . . . . .	28
3.4	Offline use . . . . .	29
3.4.1	Time Synchronization . . . . .	29
3.4.2	Key Rotation . . . . .	30
3.5	System Architecture . . . . .	30
3.5.1	Login and Registration . . . . .	31
3.5.2	QR code Generation . . . . .	32
3.5.2.A	Expiration dates in QR codes . . . . .	33
3.5.3	QR code Validation . . . . .	33
<b>4</b>	<b>Implementation</b>	<b>35</b>
4.1	Mobile Application . . . . .	37
4.1.1	Login and Register . . . . .	38
4.1.1.A	Register . . . . .	38
4.1.1.B	Login . . . . .	40
4.1.2	Purchase of tickets . . . . .	41
4.1.2.A	Storage of tickets . . . . .	43
4.1.3	Display tickets . . . . .	44
4.1.3.A	Trip history . . . . .	46
4.1.4	Screenshot blocking . . . . .	46
4.1.5	Root checking . . . . .	46
4.1.6	Code obfuscation . . . . .	47
4.2	Ticket Validators . . . . .	48
4.2.1	Authentication . . . . .	48
4.2.2	Ticket validation . . . . .	48
4.2.3	Key rotation . . . . .	50
4.2.4	Ticket transactions . . . . .	51

4.3	Back-end Servers . . . . .	52
4.3.1	Authentication Server . . . . .	52
4.3.2	Ticketing Server . . . . .	54
4.3.2.A	Access token validation . . . . .	54
4.3.2.B	Services . . . . .	54
4.3.2.C	Creation of signing keys . . . . .	56
4.3.2.D	Creation of encryption keys . . . . .	56
4.3.2.E	Anti-fraud System . . . . .	57
4.4	Clock Synchronization . . . . .	58
<b>5</b>	<b>Evaluation</b>	<b>59</b>
5.1	Performance Results . . . . .	61
5.1.1	Creation of QR codes . . . . .	62
5.1.2	Validation of QR codes . . . . .	63
5.1.2.A	Comparison between ECIES with RSA-AES . . . . .	64
5.2	Battery usage . . . . .	65
5.3	Security Issues . . . . .	66
5.3.1	Reverse Engineering . . . . .	67
5.3.1.A	Code obfuscation . . . . .	67
5.3.1.B	Encrypted QR codes . . . . .	67
5.3.2	Cloning . . . . .	68
5.3.3	Double Spending . . . . .	68
5.3.3.A	Mobile phone sharing . . . . .	68
5.3.3.B	Sharing QR codes over the internet . . . . .	69
5.3.3.C	Ticket extraction . . . . .	70
5.3.4	Phishing . . . . .	70
<b>6</b>	<b>Conclusion</b>	<b>73</b>
6.1	System Limitations and Future Work . . . . .	76
	<b>Bibliography</b>	<b>79</b>



# List of Figures

3.1	Ticket Acquisition Diagram . . . . .	24
3.2	Ticket Validation Diagram . . . . .	25
3.3	Table with the fields in the ticket . . . . .	28
3.4	Architecture . . . . .	31
3.5	QR code Generation . . . . .	32
3.6	QR code Validation . . . . .	34
4.1	Registration page in the mobile application . . . . .	38
4.2	Registration process sequence diagram . . . . .	39
4.3	Sign in page in the mobile application . . . . .	40
4.4	Payment confirmation page . . . . .	42
4.5	Add new card page . . . . .	42
4.6	Empty home page in the mobile application . . . . .	43
4.7	Tickets in the home page of the application . . . . .	43
4.8	Display ticket page . . . . .	44
4.9	Green progress bar . . . . .	45
4.10	Yellow progress bar . . . . .	45
4.11	Red progress bar . . . . .	45
4.12	Extraction of tickets from QR codes . . . . .	49
4.13	Ticket accepted . . . . .	50
4.14	Ticket denied . . . . .	50
4.15	Key rotation in ticket validators . . . . .	51
4.16	Relevant fields in the access token . . . . .	53
4.17	Table with the services provided by the ticketing server . . . . .	55
5.1	Performance results from the creation of QR codes in different devices bar chart . . . . .	62
5.2	Performance results from the validation of QR codes bar chart . . . . .	63

5.3	Performance results from the comparison between ECIES and RSA-AES bar chart . . . .	64
5.4	Results from battery usage testing results line chart . . . . .	66

# Acronyms

<b>AEAD</b>	Authenticated Encryption with Associated Data
<b>AES</b>	Advanced Encryption Standard
<b>BLE</b>	Bluetooth Low Energy
<b>CA</b>	Certificate Authority
<b>CBC</b>	Cipher-Block Chaining
<b>CICO</b>	Check-In/Check-Out
<b>CIO</b>	Check-In Only
<b>DES</b>	Data Encryption Standard
<b>DH</b>	Diffie-Hellman
<b>EC</b>	Elliptic-Curve
<b>ECC</b>	Elliptic-Curve Cryptography
<b>ECIES</b>	Elliptic-Curve Integrated Encryption Scheme
<b>FIFO</b>	First In, First Out
<b>GCM</b>	Galois/Counter Mode
<b>GPS</b>	Global Positioning System
<b>HMAC</b>	Hash-based Message Authentication Code
<b>HKDF</b>	HMAC-based Key Derivation Function
<b>HTTP</b>	Hypertext Transfer Protocol
<b>HTTPS</b>	Hypertext Transfer Protocol Secure

<b>IBM</b>	International Business Machines Corporation
<b>IETF</b>	Internet Engineering Task Force
<b>IES</b>	Integrated Encryption Scheme
<b>IMEI</b>	International Mobile Equipment Identity
<b>ISO</b>	International Organization for Standardization
<b>JSON</b>	JavaScript Object Notation
<b>JWT</b>	JSON Web Token
<b>MAC</b>	Message Authentication Code
<b>MMS</b>	Multimedia Messaging Service
<b>NFC</b>	Near Field Communication
<b>NIST</b>	National Institute of Standards and Technology
<b>NTP</b>	Network Time Protocol
<b>OS</b>	Operating System
<b>PDF</b>	Portable Document Format
<b>QR code</b>	Quick Response Code
<b>RFID</b>	Radio-Frequency Identification
<b>RSA</b>	Rivest–Shamir–Adleman
<b>SDK</b>	Software Development Kit
<b>SMS</b>	Short Message Service
<b>URI</b>	Uniform Resource Identifier



# 1

## Introduction

### Contents

---

1.1 Background . . . . .	3
1.2 Research Objective . . . . .	4
1.3 Thesis overview . . . . .	5

---



## 1.1 Background

Mobile phones are invaluable systems of our society and a rapid development of the internet in recent years has allowed mobile phones to be used more than any other form of device. The development of technologies in mobile phones has contributed to the growth of use and reachability of barcodes such as QR codes, since they can be processed not only by barcode readers but also, by mobile phones.

Barcodes are methods that allow for the access to services. Since barcodes provide a uni-directional method of communication, very efficiently and free of charge, barcodes are used in the most varied of applications such as the sharing of information, URLs and even, used in payments. Taking this into account, by being able to facilitate a payment in order to obtain a service, the intention is to extend this method to ticketing systems in public transports. Besides barcodes, the development in the technologies regarding mobile phones can also be extended to ticketing systems, with mobile ticketing becoming more relevant and widely used than ever before.

Mobile ticketing aims to provide an alternative method to the acquisition and usage of tickets, with focus on allowing users to be more involved in the process of purchasing tickets and enhancing the user experience. Some users may avoid using public transports due to the fact that public transports require a great deal of physical contact, and the possibility of having to wait for long queue times. Therefore, through the study and use of recent technologies which are available in mobile phones such as QR codes, mobile ticketing can be more intuitive and suitable for their needs.

With focus on a ticketing system using tickets in the form of QR codes, the goal is to provide an alternative method to using ticketing systems, by taking advantage of mobile ticketing for the offline validation of tickets. Mobile ticketing systems provide the ability of using tickets within the mobile phone application, with a view to process the obtaining and validation of tickets. When trying to display a ticket for the validation process, the mobile application creates a mobile ticket, in the form of a QR code.

In the public transport industry, the validation of tickets is done offline since data connections are often unreliable and in order to prevent frustration when internet is not available. Besides that, the security of tickets can be achieved using bi-directional communication such as the case of smartcards which does not work for static barcodes since barcodes do not receive information.

Since QR codes do not receive information, having a static QR code leads to the problem of not being able to update the contents of the QR code and thus, the security behind tickets could become compromised. Cloning attacks are attacks that work by copying and stealing information from other users such as, in the case of this system, the copying of QR codes. In order to prevent against these attacks, the mobile application makes use of technologies such as one-time passwords. With this, the mobile application implements a system with dynamic QR codes, which protects this system against the most commonly used attacks against mobile ticketing.

Applications of this system can be extended to most ticket-based systems such as public transports.

By having users acquire and present tickets from their mobile phone, they can use public transports without the need of a public transport card. In addition, this system can be adapted to provide other additional services that exist in public transports such as the ability to select the zones, for which the ticket is valid. Upon purchasing a ticket, a QR code is generated within the user application and the user must present the QR code to a reader system which can be provided, for instance, inside the public transport vehicle.

## **1.2 Research Objective**

### **1.2.1 Fraud Prevention**

One of the main objectives with the implementation of this system is the prevention of fraud. It is mandatory that the system is able to accept valid tickets which are tickets that have not expired and that have not been previously used. Any other situations must be rejected as the implementation of this system must not lead to a monetary loss for neither the clients nor the companies that provide and manage the system.

In addition, it should not be possible to erase and change data from the mobile phone as to deceive the system and be able to create false or re-use tickets. In order to ensure security and privacy of the data, similar situations should be detected and prevented.

### **1.2.2 Interoperability**

The system should be available to work with every smartphone, independent of the manufacturer or technology, regardless of its capabilities. Any iOS and Android versions should be able to use the mobile application as long as they support internet access. With this, we will focus on versions such as:

- iOS 14.4 and later
- Android 8.0 and later

### **1.2.3 Convenience**

The system should facilitate the experience of using ticketing systems as users are provided with an alternative to purchasing physical tickets. The system should simplify the process of acquiring tickets and instead of having to wait for queue times or engage in contact with people or ticket-vending machines, users have a mobile application at their disposal, in which they can purchase tickets and use them to acquire services. Furthermore, the system must ensure that in order to process the validation of tickets, users are not required to be kept waiting longer than the current ticketing systems.

#### **1.2.4 Offline Access**

Another main objective with the implementation of this system is availability. Users should be able to use the system and perform ticket validation without the need for having an active internet connection at all times, in the event that a certain location has limited or there is a failure with the network access. Since situations like these could lead to frustration for the users of the ticketing system, the mobile application should be able to display tickets without internet connection, even if only for a brief period of time.

### **1.3 Thesis overview**

This document is organized using the following structure: Chapter 2 provides a background on ticketing systems and technologies used for the proposed solution. Chapter 3 contains the architecture of the system and an overview of the components that integrate the proposed solution. Their integration is essential for the correct functioning of the prototype of the implementation of this system. Chapter 4 contains the implementation details of the proposed solution, with technical detail for every component of the system and reasoning behind the technologies that each component uses. Chapter 5 contains the evaluation of the prototype by providing a detailed summary of the results obtained and a relationship of the obtained results with the imposed research objectives. Finally, Chapter 6 concludes the document with a summary of the work developed in the context of this dissertation and proposes future work for the further development of this system.



# 2

## Related Work

### Contents

---

2.1 Ticketing Systems . . . . .	9
2.2 QR Codes . . . . .	11
2.3 Cryptography . . . . .	12
2.4 Hashing Algorithms . . . . .	15
2.5 One-Time Passwords . . . . .	16
2.6 Digital Signatures . . . . .	18
2.7 OpenID Connect . . . . .	19

---





Nowadays, there is an increasing tendency to use public transportation in regards to environmental issues or due to the increasing costs of owning personal vehicles. With this, ticketing systems are also evolving in the forms of payment and technologies used to validate tickets, to meet customer needs and provide a better experience for users.

Hence, there have been numerous advancements in the many forms of payment and unsurprisingly, the trend towards electronic cashless commerce is growing worldwide. Ticketing systems consist of payments in the form of tickets and there have been implemented systems which take advantage of Near Field Communication (NFC), Radio-Frequency Identification (RFID), Bluetooth Low Energy (BLE) and Quick Response Codes (QR codes).

However, appropriate security mechanisms are required in order to avoid unintended results. With focus on the security of mobile ticketing systems, one must consider the basic security services to become part of the system such as authentication, confidentiality, integrity and non-repudiation. Many of these security services depend on cryptographic operations such as one-time passwords, digital signatures, etc.

In this chapter, it is intended to provide context on the technologies used and existing solutions with these technologies, which are relevant for this case, and how these can be related with the prevention of fraud.

## 2.1 Ticketing Systems

Ticketing systems are systems that are comprised of exchange of tickets in order to process payments to services. Industries that have a large amount of users, using the system at the same instant, often take advantage of ticketing systems as the use of tickets can facilitate the usability of the service and verification of which users are using and have paid for the service. Hence, ticketing systems are most commonly seen in the public transport industry.

Public transports make use of tickets that are purchased and validated, in ticket validators, in order to be able to use a desired method of transportation. These tickets are physical tickets emitted by an office worker or ticket vending machines, usually located at the entrance of a public transport, and each ticket has attributes related to the service provided by the public transport. Even though traditional tickets are still widely used, these types of tickets are falling in decay as their use is less convenient among other methods of storing tickets [1] such as in smart cards.

The use of smart cards is efficient and convenient as not only are smart cards able to store multiple tickets but also other information such as wallet balance. By taking advantage of RFID technology, purchasing and verifying tickets is a simple and rapid process.

Alternatively, there have been proposed systems which make use of mobile tickets instead of using

smart cards. Mobile tickets and e-tickets are different types of digital tickets [1] and digital ticketing systems can take advantage of technologies such as NFC, BLE and QR codes [2] [3] [4] [5] [6].

E-tickets and mobile tickets share some of the same properties and attributes as physical tickets. However, e-tickets consist of electronic tickets which are sent to users through an online mailing service and received in the form of Portable Document Format (PDF) whilst mobile tickets consist of tickets that are provided in a mobile phone application and are stored electronically on a mobile device such as a smartphone. Digital tickets provide numerous advantages over smart cards as passengers are able to purchase tickets remotely as to avoid queue times and reduce contact with other passengers which provides convenience and usability [1]. The advantages of a mobile ticketing system are not limited to passengers as the authority in charge of the system can also benefit from these types of tickets. Digital ticketing systems require no cash transactions, allows for different types of tickets and reduced maintenance costs of machines which maximizes profits and can even lead to a reduction in the price of the tickets [1] [3].

As stated above, mobile ticketing can take advantage of technologies such as QR codes, NFC and BLE. Leal et al. [7] propose multiple approaches to integrate a mobile ticketing system with Porto's public transportation system, by taking advantage of QR codes and NFC as well as BLE. With the QR code approach, each public transport vehicle has these placed inside the vehicle and passengers are required to scan the QR codes with the application in order to check-in, with check-out being optional. In addition, users have the ability to either buy tickets in advance or being charged after reading the QR code. The results of this approach provide convenience and usability as well as not requiring much effort from the passenger, in order to process the validation step.

Belani et al. [8] propose two different approaches for a mobile ticketing system. One implementation, which is an online validation system, takes advantage of Multimedia Messaging Service (MMS) technology in order to send mobile tickets to clients, and clients validate the mobile ticket which is in the form of a data-matrix, in a 2D Image Scanner. Alternatively, it is proposed an offline validation system, by using QR codes. A ticket, which contains user information and other details, is processed in order to generate a hash value, from the concatenation of the ticket identifier and the digital signature of the ticket identifier. Since there is not an established connection between the mobile phone and the validation system, the content of the ticket is encrypted using asymmetric-key cryptography. The resultant hash value is stored inside a QR code which, after being read by a web camera, is validated in the validation system.

### **2.1.1 Entrance and Exit**

In a public transport infrastructure, the entrance and exit of vehicles can be achieved using two different approaches: the Check-In Only (CIO) approach or using a Check-In/Check-Out (CICO) approach. A check-in only approach is where the user is only required to check-in to make use of the services

provided by the system. In order to check-in, users must have a valid ticket which is to be presented in the ticket validator in order to validate the ticket. To complete the service, users can check-out of the system by exiting the service and are not required to advertise the system that the service is complete. Similarly to the check-in only approach, a check-in/check-out approach requires users to present their ticket at the entrance of the service provided by the system. However, at the exit of the service, users are required to present the ticket to the ticket validator, in order to exit the system.

Currently, the public transport industry infrastructure contains, in metro or train stations, a check-in/check-out approach where users are required to validate their tickets at the entrance of a public transport vehicle to be able to enter the vehicle and present the ticket to exit the ticketing system. As an example, when using the Metro in Lisbon, users are required to present their smartcard at the entrance and exit of the metro station.

## **2.2 QR Codes**

Nowadays, there is a vast use of QR codes due to their versatility, the ability to be read from any direction and their small printout size which makes their use very convenient. Their purpose is to make processes faster and more efficient. Quick Response Codes or QR codes are matrix barcodes, readable by mobile phone cameras or dedicated QR barcode readers [9]. In addition, QR codes are established as an International Organization for Standardization (ISO) standard and have the property of being able to be used offline.

QR codes consist of black squares arranged in a square grid on a white background and contain data which can be encoded as text, URLs or other data. The types in which data can be encoded are numeric, alphanumeric, byte or kanji. QR codes have a high capacity of encoding data, even though size depends on the mode which data was encoded. The maximum size of a QR code is 7089 characters when its contents contain solely numeric characters whereas when using binary data, the maximum size is reduced to 2953 bytes. Furthermore, QR codes include information to provide error correction capability. Information is validated using Reed-Solomon error correction algorithm, which can correct errors up to 30% damage.

As stated above, QR codes are very versatile. QR codes are commonly used to share information regarding e-commerce, customer information, customer surveys, link sharing and are often used in the media. There are plenty websites provide services to generate QR codes, some even free of charge, which leads to promoting the use of QR codes, since the process of creating QR codes is simplified and does not require previous domain knowledge. In regards to link sharing, sharing QR codes with customers instead of spelling the link is much more convenient and simple. Not only does scanning of QR codes take little to no effort and time but also, it is prone to less error typing and prevents frustration.

In addition, it is possible to combine QR codes with cryptography so that the contents of the QR code are encrypted and non-readable [10] [11].

Other applications to QR codes are in public transports [4] [12]. In public transports, the main use of QR codes are e-ticketing and user information. Real-time timetables of departures and arrivals of public transports as well as customer surveys are some examples of the information users can obtain using QR codes, with e-ticketing referring to the payment of tickets electronically. An approach by Upendra et al. [12] allows users to select the departure and arrival locations as well as the amount of people. Eventually, the QR code is created in the application of the conductor of the bus and users scan the QR code in order to complete the purchase of tickets. Another approach made by Finžgar et al. [13] consists of an online ticketing system that receives train tickets in the form QR codes, from a ticketing server which takes advantage of Hypertext Transfer Protocol (HTTP) requests. Each QR code contains user information, to and from locations, a command type and whether or not the user is checking in or checking out of the station.

## 2.3 Cryptography

Secure communication in vulnerable networks can be achieved using encryption of data sent across machines. As technology is in constant development, the opportunities to modify, use, disclose and read confidential information have grown, and therefore, the continuous development of a secure computer system will ensure that confidential information can be securely transferred across computer networks.

Encryption is the process of converting data in a form that is not able to be read without decrypting or deciphering it. The original message or text, before applying any processing, is called plaintext. By applying cryptographic algorithms with an encryption key, we can convert plaintext into non-readable data. The result of encrypted data or ciphertext is known as a cryptogram.

The process of encryption is a reversible process and thus, it is possible to obtain the original plaintext from the ciphertext. This process is called decryption and, the study and practice of encryption and decryption is called cryptography.

The goal of cryptography is to prevent an unauthorized person or group of users from accessing any confidential data, and modern cryptography relies upon subjects like mathematics and computer science. Chip-based payment cards, computer passwords and digital currencies are just a few of the applications of the use of cryptography [14]. In addition, cryptography can also be used for user authentication, with resource to using cryptographic algorithms.

Any cryptographic algorithm can be cited as secure, if its key properties can not be deduced with a given ciphertext. For instance, a key can be a secret shared between two communicating parties with a view to encrypt and decrypt data. With this, cryptography consists of methods of message en-

ryption, with symmetric-key cryptography and asymmetric-key cryptography, Message Authentication Codes (MACs) and Hash functions.

### **2.3.1 Symmetric Keys**

Symmetric-key cryptographic mechanisms rely on the existence of a single secret piece of information, also known as a secret key, shared in advance, and exclusively among the communicating parties.

Earlier private-key algorithms such as Data Encryption Standard (DES), developed by International Business Machines Corporation (IBM), are no longer considered sufficiently secure against brute-force attacks. Due to the increasing computer processing power, these algorithms were forced to be exchanged by other algorithms which provided better security and robustness, such as Advanced Encryption Standard (AES) [15]. A technique by D. Singh et al. [16], provides a solution to decrease encryption times by converting plaintext to floating point numbers.

Symmetric-key algorithms have a few drawbacks when used in a system with a lot of users. For instance, with an increasing number of users, it becomes extremely difficult to generate, distribute and keep track of secret keys as the number of user pairs increases. Besides that, using the same key for both encryption and decryption creates high risk, due to the fact that the private key must be shared between communicating parties which increases the risk of interception by attackers.

### **2.3.2 Asymmetric Keys**

Asymmetric-key cryptography eliminates the need of exchanging a single unique private key and instead, using two keys. Each user has one public key and one private key. The private key is associated with a certain user and it is not shared with other users whereas the public key is a key that can be openly distributed, without compromising security, and each key performs a unique function. Using the public key for encryption requires the private key for decryption. Conversely, using the private key for encryption requires a public key for decryption.

Similarly to symmetric key cryptography, public-key algorithms are susceptible to "brute-force key search" attacks. In order to prevent these attacks, there have been developed algorithms such as RSA, which provide randomized padding methods and exchange keys using Diffie-Hellman (DH) key exchange algorithm. Rivest–Shamir–Adleman (RSA) algorithm is an asymmetric cryptographic invented by Rivest, A. Shamir, and L. Adleman in 1997 and has become the most widely used public-key algorithm. The security of RSA is based on the difficulty of the integer factorization problem. RSA key sizes refer to the bit-length of the RSA modulus, which is a large non-prime number that is part of the public key.

RSA keys with sizes lower or equal to 1024 bits have already been cracked and thus 1024-bit RSA

keys are no longer the recommended standard [17]. Public-key algorithms are generally slow and with increments in the key size, the decryption time increases [18]. 2048-bit RSA keys provides a better trade-off between performance and security [18] whilst 4096-bit RSA keys are most commonly used in systems which make use of asymmetric keys that are not to be changed for a long period of time.

Mainanwal et al. [19] propose an authentication system which combines RSA algorithm with a zero-knowledge proof protocol which, by applying this to the client, the server is able to identify the user. This system prevents sniffing attacks due to the fact that the content travelling in the network, is encrypted. Furthermore, the authentication system uses one-time tokens to ensure that in the event that information is intercepted, attackers are neither able to use the information nor able to read the content of the intercepted data.

### **2.3.2.A Hybrid Encryption**

In public-key cryptography, there are some systems which make use of encryption methods using a combination of asymmetric key algorithms with symmetric key algorithms, also known as hybrid encryption. Hybrid encryption combines the efficiency of symmetric encryption with the convenience of public-key encryption. Similarly to symmetric encryption, a symmetric key is used to cipher the plaintext. As previously mentioned, this method creates a ciphertext in a very short period of time, despite the size of the message. Using public-key encryption, this key is ciphered with the public key of the receiver, and sent along with the ciphered message. The receiver uses its private key to decrypt and obtain the symmetric key, which is then used to decrypt the contents of the original message.

One approach to hybrid encryption is Elliptic-Curve Integrated Encryption Scheme (ECIES), which is a variant of the Integrated Encryption Scheme (IES) using Elliptic Curves. With this, ECIES is a public-key authenticated encryption scheme which combines ECC-based asymmetric cryptography with symmetric ciphers to provide the encryption of data. Similarly to asymmetric-key cryptography, ECIES can provide data encryption using one of the Elliptic-Curve (EC) keys, from the EC key pair, and decryption using the corresponding EC key.

### **2.3.2.B Elliptic Curve Cryptography**

Nowadays, there is an increasing trend to using Elliptic-Curve Cryptography (ECC) approaches in public-key cryptosystems, in a wide variety of applications such as digital signatures (ECDSA) and key agreement protocols (ECDH). This is due to the fact that, in contrast to RSA, Elliptic Curve cryptographic algorithms make use of keys with much smaller sizes in order to achieve a similar level of security.

For instance, the widely used ECC key size of 256 bits offers 128 bits of security, which is equivalent to using RSA keys with 3072 bits. Therefore, the use of ECC requires less processing power, in comparison with RSA, which is especially beneficial when using devices with limited storage and processing

power. Besides that, the use of ECC for hybrid encryption such as using ECIES, in comparison with hybrid encryption using RSA-AES, leads to a significant increase of speed, which can go up to 22 times faster [20].

## 2.4 Hashing Algorithms

Hashing algorithms were introduced as a tool to protect the authenticity and integrity of information. By definition, hashing algorithms are mathematical algorithms that map a block of data of an arbitrary size to a hash of fixed size. Nearly any piece of data can be defined as a message and the main difference between hashing and encryption is that a hash is irreversible.

Hashing algorithms make use of cryptographic hash functions and, since cryptographic hash functions are deterministic procedures, the output is always the same when given the same input. Inputs passed to hash functions are called messages and output is often referred as the message digest.

Hash functions are also called 'one-way functions' because all hash functions have the property that it is impossible to determine the input knowing only the output. A cryptographic hash function aims to guarantee a number of security properties such as being rapid to compute the hash value. Most importantly, cryptographic hash functions differ from hash functions on the fact that two different messages almost never produce the same message digest. Authenticity and integrity of information are kept due to the fact that if a message digest is changed, the file has changed.

### 2.4.1 Secure Hash Algorithms

SHA or Secure Hash Algorithms are a set of cryptographic hash functions, published by the National Institute of Standards and Technology (NIST). Despite being internally similar to MD5, SHA-1 and SHA-2 are considered more secure even though cryptographic shortcomings have been found for SHA-1, and the standard was no longer approved for most cryptographic uses after 2010.

SHA-1 creates a 160-bit hash value. SHA-2 includes SHA-224, SHA-256, SHA-384, and SHA-512, named after the length of the message digest each creates. SHA-256 uses 32-byte words whereas SHA-512 uses 64-byte words. SHA-2 hash functions are implemented in security applications and protocols such as TLS and SSL, SSH, etc. SHA-256 is the recommended standard by NIST.

Performing a comparison between MD5 and SHA-1 hashing algorithms, Ratna et al. [21] perform comparative analysis on the duration of brute force testing using Simple-O which is an authentication system, in order to obtain a plaintext from a ciphertext. With this, it is observed that plaintexts that are smaller than 6 characters are easily broken, taking just a few minutes whilst plaintexts longer than 6 characters increase the time that takes to perform a brute force attack on MD5 and SHA-1, from a considerable amount of hours to a large number of days. Despite the fact that the size of the password

(plaintext) makes the biggest impact in regards to the duration of the brute force attack, it is observed the difference in time and security from both hashing algorithms against brute force attacks, with SHA-1 being more secure with taking an average of 15% more time to obtain the plaintext, from a given ciphertext. Taking into account the performance of the hashing algorithms and despite being considerably more secure than either MD5 or SHA-1, SHA-256 is about 20-30% slower [22] to calculate hash values than either MD5 or SHA-1 hashes.

Digital signatures, one-time passwords and key generation in symmetric and asymmetric key cryptosystems are examples of the applications of hash functions.

### **2.4.2 Hash-based Message Authentication Code**

HMAC is a MAC algorithm that is computed by two calls to a hash function such as MD5 or functions from the SHA family. Hash-based Message Authentication Code (HMAC) is a standard algorithm currently used [23]. When hashing algorithms are combined with encryption, hashing algorithms produce special message digests that identify the source of the data. These special digests, produced from a message and a secret key, are called Message Authentication Codes and ensure authenticity and integrity. Similarly to digital signatures, HMACs make use of hashing algorithms and cryptographic keys. However, HMACs use symmetric keys whilst digital signatures use asymmetric keys.

The cryptographic strength of the HMAC depends upon cryptographic strength of the underlying hash function. Similarly, the performance of the HMAC combined with a hashing algorithm depends on the performance of the hash function [24]. HMAC share the same properties as hashing algorithms except that some hashing algorithms are vulnerable to length extension attacks. HMAC provides security against these attacks by adding a secret key to a message. With this, HMACs began to be used as a complement to existent one-time passwords algorithms [25].

## **2.5 One-Time Passwords**

Commonly used in multi-factor authentication, one-time passwords are passwords that are valid for only one login session or transaction. The general idea of a one-time password is adding another layer of security in order to protect against attacks. These include phishing, keyboard logging and man-in-the-middle. One-Time Passwords avoid a number of vulnerabilities that static passwords have. Static passwords can be guessed, forgotten, stolen, eavesdropped or deliberately being told to other people.

Attacks such as replay attacks, which in the event an attacker gets to know an OTP by intercepting a transmission of data which has already been used, are mitigated because the code becomes invalid and cannot be reused. Advantages of using OTPs become significant in risky situations such as entering credentials to log in a bank system or making payment transactions.



The concept of passwords with a one-time use was first introduced by Leslie Lamport [26]. Nowadays, there have already been implemented plenty protocols which are based on one time passwords. Examples include authentication, integration with voice recognition, combination of sequence numbers with timestamps and GPS-based OTPs [27] [28] [29] [30].

One-time passwords are generated from applying a hashing algorithm to a seed which should only be known to the creator and receiver of the OTP. Each OTP has a number of digits which varies according to the situation with 4-digit OTPs being typically used on systems that do not require a great deal of security whereas 8-digit OTPs being usually linked to high risk situations or situations involving a large amount of OTPs being generated at the same time. Since two OTPs must not be equally generated, the number of unique combinations increases with higher digit codes.

There are two types of one-time passwords: HMAC-based or Time-based.

### **2.5.1 HMAC-based One-Time Passwords**

HMAC-based One-Time Passwords, often referred as HOTPs, was published as RFC4226 [25] by the Internet Engineering Task Force (IETF). HOTP defines an algorithm to create an one-time password from a secret key or seed and a counter. With every use, the counter is increased leading to change in the OTP value when recalculating the algorithm.

Combining HMAC with a hash algorithm, typically SHA-1 [25], results in an irreversible hash value. Since the HMAC-SHA1 hash value contains 20-bytes, truncation is applied in order to get a hash value with 31 bits. The resultant truncated hash value are the last 31 bits from the HMAC-SHA1 hash value. With this, the HOTP value is defined by extracting a number of digits, starting from the last digit, of the truncated hash value. As previously stated, the number of digits are selected according to the required level of security.

### **2.5.2 Time-based One-Time Passwords**

In [31], TOTP serves as an extension of HOTP. Being an extension of HOTP, both types of one-time passwords share similar properties such as being able to only be used once and both provide strong authentication mechanisms due to their advantages, stated above, over static passwords. However, a TOTP generates an OTP that is only valid during a timestep instead of having a counter which is incremented upon each use of the OTP. A timestep is a certain period of time which TOTPs are valid. Generation of TOTPs only change upon the passing of a timestep and their generation is deterministic. Two TOTPs generated in the same timestep with the same secret key and number of digits will be the same.

As stated above, OTPs provide a strong authentication mechanism. A technique by Chun-Ying

Huang et al. [32], focuses on preventing phishing attacks. The website has register and login methods and the OTP is sent to the user from an instant messaging service and by having the user perform additional security steps. Afterwards, the user is granted permission to insert the OTP. The website also takes in session tokens and IP addresses. Z. Huang et al. [29] proposed a system which provides a combination of OTPs with private key encryption. The mobile phone application shares a private key with the verification system and the OTP consists of 8-digit value from a timestamp and a sequence number. The verification system applies the same process of calculating OTPs as the application, and the OTP is accepted in the event that both OTPs match. In case the user sends an expired OTP to the validation system, the validation system has a resend period for convenience and usability. Liao et al. [33] proposed a novel technique which is a QR code one-time password authentication protocol, with a service provider granting access rights to authorized users. In this technique, it is described the processes of registering and validation of the OTP, which makes use of timestamps, and possible attacks to the system.

## 2.6 Digital Signatures

Digital Signatures are mathematical schemes for verifying the authenticity and integrity of digital messages or documents. Valid digital signatures give recipients of the message a strong reason to believe that the message was created by a known sender and the message was not altered in transit. In many countries, digital signatures are considered legally binding and hold the same value as traditional document signatures. The process of creating a digital signature consists of a key generation algorithm, a signing algorithm and a signature verifying algorithm.

Firstly, a message can be encrypted using symmetric [34] or asymmetric cryptography. In symmetric cryptography, it is a secret key known only to the sender and receiver whereas in asymmetric cryptography, two asymmetric keys, a private key and a public key, are generated. The sender uses their private key to encrypt the message and the receiver can decrypt the message using the public key of the sender and is able to confirm the identity of the sender.

Secondly, a hash function is applied to the message. A signing algorithm uses the hash value and the sender's private key to generate the digital signature. Afterwards, the message and the digital signature are sent to the recipient.

Finally, the receiver applies the inverse signature function, or signature verifying algorithm, using the public key of the sender to the digital signature and the output is the original hash value. The receiver can then compare this hash value with the hash value that was sent with the digital signature and in case both hash values match, it confirms that the message was sent by the sender and the content of the message has not been altered. Therefore, digital signatures have the property of non-repudiation

which does not allow the signer to deny signing the document.

There are a plethora of techniques that have been implemented using digital signatures. S. Jarusombat et al. [35] propose a technique that combines digital signatures with GPS tracking. By taking advantage of the developments in the Global Positioning System (GPS) technology, the system can generate a digital signature based on geo-encryption and a mobility model, which leads not only to non-repudiation of the sender but also non-repudiation of the receiver. Due to the risks and modern e-commerce security issues in China, C. Tian-huang et al. [36] propose a digital signature system to increase security regarding e-commerce. A technique by S. Alam et al. [37] implements the use of digital signatures with 512-bit RSA keys, in order to perform authentication and encryption of images.

## 2.7 OpenID Connect

OpenID Connect, combining OpenID and OAuth together, is an emerging representational transfer-based identity solution and it is one of the most adopted open standards to potentially become the standard for securing cloud computing and mobile applications.

OpenID Connect consists of an authentication layer on top of OAuth2.0 [38]. OpenID Connect consists of three main steps: Trust Establishment, Token Generation and Token Redemption, and provides services with the proper authorization, authentication, and Single-Sign-On capabilities.

To start off, the user authenticates with their credentials and makes an authentication request which will be returned by the server, as an authorization code. Using the authorization code, the user sends a token request to the server. In the event it is valid, the server generates an *access token*, *id token* and code which are sent to the user. Since access tokens are used for authorization, access tokens [39] [40] must remain confidential. Any user that gains access to an access token can have the control and permissions of the user for which the token was emitted. Google, Yahoo and Facebook have already implemented identity providers that are similar and based on OpenID Connect technology.

A technique by Kakizaki et al. [41] proposed an information management method using OpenID Connect for identity verification. In order to verify the identity of the user, the system relies on OpenID Connect and allows for the management of user information independently. Every attribute is assigned with a Uniform Resource Identifier (URI) and thus, OpenID Connect identity provider only persist user's unique ID and related attribute URIs. This feature caters to the healthcare organizations that have concerns about exposing some sensitive patient information to a third party.



# 3

## Architecture

### Contents

---

3.1 Overview . . . . .	23
3.2 Use Cases . . . . .	24
3.3 Security and Fraud Prevention . . . . .	26
3.4 Offline use . . . . .	29
3.5 System Architecture . . . . .	30

---



### 3.1 Overview

The main goal of this project is to develop a mobile ticketing system that allows for the offline validation of tickets, in the form of QR codes, by taking advantage of the technologies available in mobile phones. With using technologies such as one-time passwords, this system contains a method to regularly update QR codes, known as the dynamic generation of QR codes, that are to be presented to a ticket validator. In addition, this chapter contains an assessment to how this project can be implemented in different types of ticketing systems as for instance, public transports. It is presented the overall functioning of the system including the components roles and system architecture and also, this document contains an explanation of our technology choices and describe how the main objectives can be achieved.

Unlike smartcards which receive information from validators, QR codes are methods of uni-directional communication and thus, the application is not able to recognize that a ticket has been used unless it receives confirmation from other methods of communication such as the online confirmation from ticket validators or the backend servers. In the case of this system, ticket validators are not guaranteed to have full internet availability and therefore, the implementation of online communication with near offline ticket validators would be infeasible, since ticket validators would not be able to send ticket validations, in real time, to a backend server which would then be responsible for notifying the mobile application that the user presented a ticket to a ticket validator.

Despite working in a near offline state, ticket validators should contact the backend server regularly as to obtain new information and send the validations of tickets, which they recorded. One of the most innovative points of this system is that, even though ticket validators perform the validation of tickets offline, they are able to remain usable for various days in case they face problems contacting the backend server. Taking into account that in a real environment such as public transports, these devices require very low maintenance, failures in communication between ticket validators and the backend servers, if not taken in consideration, could be detrimental to the correct functioning of this system.

Currently, the public transport infrastructure in metro or train stations contains a check-in/check-out approach, where users are required to validate their tickets at the entrance of a public transport vehicle, to be able to enter the vehicle, and they are required to present the ticket to leave the ticketing system. The proposed ticketing system implements a check-in only approach so that users are only required to confirm their entrance to the service, by validating their tickets at the entrance of the public transport. At the exit of the service, users can check-out of the system by exiting the service, whenever the user considers the service finished. Using buses as an example, users scan their tickets at the entrance of the vehicle and they are not required to present the ticket to leave the bus.

The implementation of this system can be subject to modifications to support other entrance/exit approaches such as a check-in/check-out approach, commonly used in public transports. In this scenario, similarly to the process of entering a service, users would present the ticket to a ticket validator which,

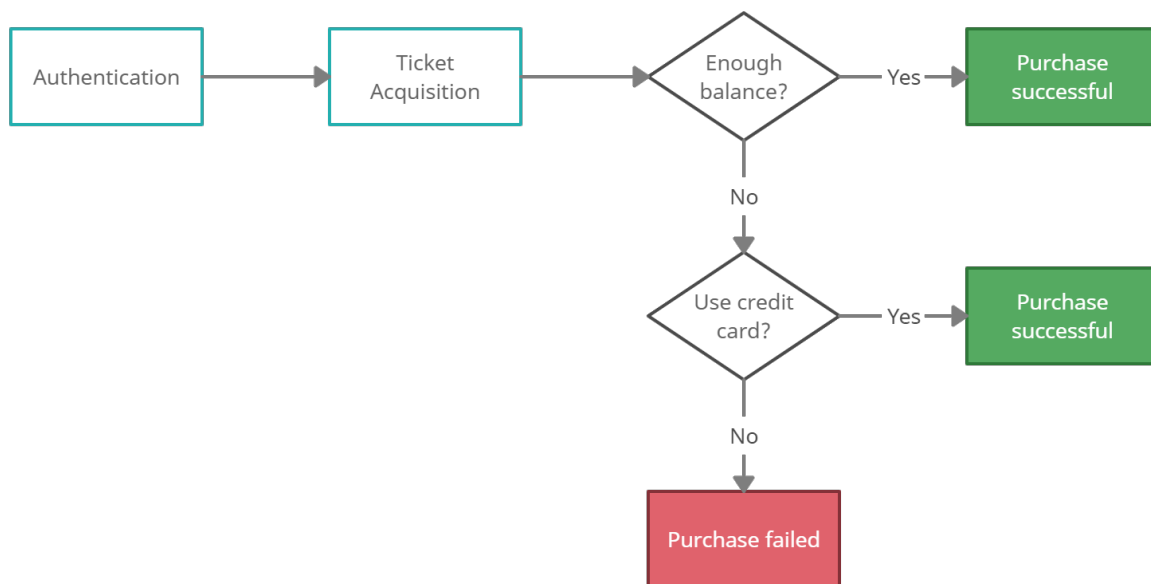
upon presenting a valid ticket, users would be allowed to exit the system.

## 3.2 Use Cases

The proposed solution is a mobile ticketing system based on the dynamic generation of QR codes through the use of a mobile application, ticket validators and backend servers. The process of using this system can be divided in two use cases: the first use case is related to the process of acquiring tickets and the second use case is related to the process of validation of tickets.

### 3.2.1 Ticket Acquisition

The figure 3.1 contains a diagram with the comprehensive representation of the method of purchasing tickets. To start off, users need to authenticate in the application with an individual user account. When using the application for the first time, users are required to create a new user account. After creating their account, users are able to sign in the application and use the system.



**Figure 3.1:** Ticket Acquisition Diagram

Following the authentication step, users can proceed to acquire tickets. After requesting the list of available tickets from the ticketing server, the mobile application displays the catalog of products for users to choose.

The purchase of tickets was not a subject of study for this project and therefore, the implementation of this system contains a simulation of the payment method. In order to implement this system in a real

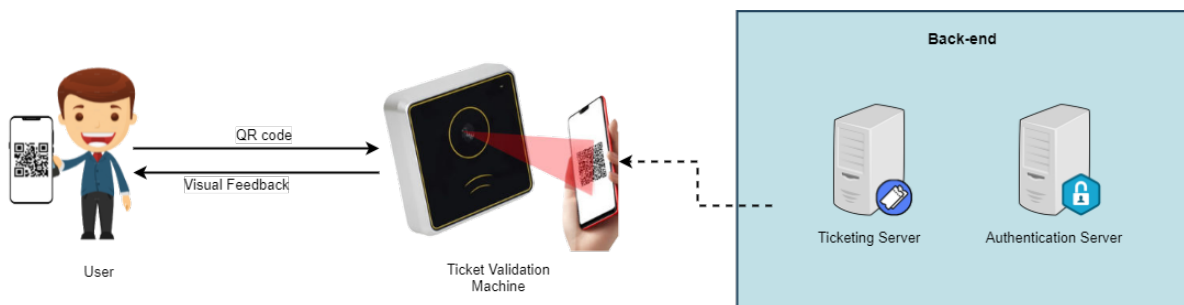


environment, the method of payment would need to be revisited, as stated in the future work section 6.1. There are two possible approaches to purchasing tickets 3.1: one is using the balance in the account and the other is using a credit card. Users that have enough balance in the account, can use their balance to make purchases to the ticketing server, whereas users that do not have enough balance can solely purchase tickets using their credit card. Users that do not have enough balance and have not introduced their credit card in the mobile application, are going to fail the process of purchasing tickets due to the fact that there is no alternative payment method.

After selecting their preferred payment method, the mobile application processes the purchase request and stores the ticket in the internal storage of the mobile phone. Before storing tickets, the contents of the ticket are encrypted to prevent the extraction of tickets from the mobile phone.

### 3.2.2 Ticket Validation

In the mobile application, users can select the ticket which they intend to use and the mobile application generates a dynamic QR code, as it can be seen in the figure 3.2. This QR code can then be presented to a ticket validator, in order to gain access to a specific service.



**Figure 3.2:** Ticket Validation Diagram

The only instance where users interact with ticket validators is when presenting a QR code to the ticket validator and, after scanning the QR code, ticket validators deliver a response to the user, in the form of visual feedback. To complete the process of validating tickets, ticket validators record a ticket validation, which is going to be sent to ticketing server, at a later time. The ticketing and authentication servers are located in the backend and the user does not have contact with them.

As aforementioned, in order to exit the system, users can simply leave the service and the journey is considered complete.

### **3.3 Security and Fraud Prevention**

With the development of an innovative ticketing system, which makes use of the offline validation of tickets, it is of utmost importance that this system can prevent attackers from committing fraud. Some users with malicious intents might attempt to attack this system, by exploiting its vulnerabilities, to gain advantage such as making use of the ticketing system whilst not paying for tickets.

Therefore, the problems which were needed to be taken into consideration when developing this ticketing system: the verification of authenticity of users and the prevention against cloning and double spending attacks.

Starting off with cloning and double spending attacks, attackers attempt to commit fraud by using tickets from other users and therefore, avoid paying for tickets. It is of utmost importance that this system can be resilient to these attacks, in order to prove its reliability.

The problem of the authenticity verification lies in the fact that the communication between the mobile application and ticket validators is done offline. As previously stated, a user presents a QR code to a ticket validator which performs the validation of ticket and emits visual feedback to the user. In order to assert authenticity and to prevent tickets from being either forged or used by other entities, this system implements two security methods in every QR code, with the use of digital signatures and one-time passwords.

#### **3.3.1 Digital Signatures**

The use of digital signatures allows for the prevention against the forgery of tickets and prevent any modifications in the contents of tickets. Each ticket contains a signature field which is verified by ticket validators, when validating tickets, and tickets whose signature does not match the signature in the ticket are rejected. In addition, every signature is accompanied with an expiration date. By setting an expiration date of the signature, in the ticket, users are required to request for a renewal of the signature regularly, and this method was implemented with a view to assert the authenticity of users on a regular basis. In the event that tickets are stolen and attackers can succeed on the bypassing the other security mechanisms implemented against the stealing of tickets, they are only able to use the ticket for the short period of validity of the signature.

#### **3.3.2 One-time Passwords**

Each QR code contains a one-time password, namely a time-based one-time password. Time-based one-time passwords are used to create a QR code that can only be used once and for a very limited duration. Therefore, the use of a one-time password contributes to the prevention against cloning attacks.

In this system, cloning attacks are attacks where the attacker is able to steal the ticket from another user, by taking a photo of the QR code. With the use of one-time passwords, this system can prevent attackers from using the QR code after it has already been used and even if not used, since one-time passwords have a very limited period of validity, the one-time password may expire before the attacker is able to use the QR code. In the former scenario, the ticket validator rejects the ticket since it contains a history of ticket validations to prevent the use of the same QR code more than once.

The use of one-time passwords led to the implementation of a system with the dynamic generation of QR codes in the mobile application. This is done to avoid displaying invalid QR codes and the mobile application automatically updates the QR code, upon the expiration of the OTP value inside the QR code. As previously mentioned, the creation of time-based passwords requires the use of a timestamp and, in order to create equal OTP values in the mobile application and ticket validators, these are required to perform the synchronization of the timestamp.

Besides using clock synchronization for one-time passwords, this system makes use of this synchronization in the time-based fields inside of the ticket. Through the use of a synchronized expiration date of the ticket and expiration date of the ticket signature, ticket validators are able to correctly validate these fields.

### **3.3.3 Validation History**

Since ticket validators perform the offline validation of tickets, they are required to maintain a history of the validation of tickets so that they can verify the usage of tickets in multiple instances, and reject tickets that are used more than once.

Using public transports as an example, the implementation of the validation history may differ depending on the vehicle. In case there is a single validator such as at the entrance of buses, the ticket validator is responsible for the validation of every ticket since all passengers must present their tickets to that ticket validator. In case there are multiple ticket validators such as at the entrance of metro stations, the station must contain a physical centralized server, which ticket validators are connected to, in order to share the validation of tickets across all ticket validators. This server may operate offline and its purpose is to keep the centralized registry of ticket validations and allow other ticket validators, located inside the metro station, to reject QR codes which were scanned in a different ticket validator.

### **3.3.4 Asymmetric Keys**

This system uses asymmetric keys due to the fact that QR codes contain sensitive information and, to provide resilience against reverse engineering attacks to the QR code. The mobile application receives a public key from the ticketing server and uses it to cipher the ticket and OTP value, and its result is the

message stored inside QR codes. A ticket validator receives the corresponding private key, also from the ticketing server, and can use it to decipher the QR code and validate its contents.

### 3.3.5 Types of Tickets

Due to the fact that the validation of tickets is performed offline and that QR codes cannot receive information, ticket validators are not able to notify the mobile application that a certain ticket has already been used. Besides that, the mobile application as well as the rest of the components of this system, implies a zero-trust policy where this system does not allow users to execute specific privileged actions and thus, not giving users the ability to commit fraud against this system.

Taking this into account, the types of tickets used by this system are time-based tickets which are tickets that can be used multiple times until the ticket becomes expired. Through the synchronization of clocks and the aforementioned security methods, this system can ensure that tickets are only usable for the specified time. In addition, it can also ensure that ticket validators are going to reject tickets whose time has already expired.

Field in the ticket	Definition
id	Unique identifier of the ticket
sub	Identifier of the owner of the ticket
code	Code of the ticket
purchaseDate	Purchase date of the ticket
expirationDate	Expiration date of the ticket
validOperators	Operators where the ticket is valid
validZones	Zones where the ticket is valid
signKeyId	Key id of the signing key
signExpDate	Expiration date of the signature
signature	Signature in the ticket

**Figure 3.3:** Table with the fields in the ticket

#### 3.3.5.A Tickets

As seen in the figure above 3.3, each ticket contains several fields which can either be related with the owner of the ticket, intended usage and security of the ticket. For instance, the signature fields are fields

which are related with the security methods in the ticket.

In this iteration of this system, there are some fields such as the intended usage fields (valid operators and valid zones), which can be considered as optional. This is due to the fact that the purpose of these fields is related with the usage of tickets in public transports and thus, the removal or substitution of these fields from the ticket would not compromise the correct functioning of this system.

## **3.4 Offline use**

Since the main goal of the proposed system is that the system can also be used offline, there are certain aspects of this system that were modified in order to support the validation of tickets without internet connection. The two most prominent scenarios where these aspects are important for the offline use of this system are the synchronization of clocks and the rotation of the encryption key.

### **3.4.1 Time Synchronization**

The synchronization of clocks to the time server requires active internet connection and there are two instances that require synchronization of clocks: the fields in the ticket and the one-time password in the QR code.

In order to be able to achieve synchronicity in times within the entities of this system, when offline, the mobile application and ticket validators use the Network Time Protocol (NTP) [42].

With this, these entities can calculate the offset of their internal time clock to the time clock in the time server. The offset value is calculated to obtain the time difference from these entities to the time server, so that the synchronization of clocks can be achieved.

In addition, the ticketing server operates online but also uses NTP to remain synchronized with the

This value is then stored in memory and when there is the lack of internet connection, the mobile application and ticket validators use the offset to become synchronized with the system. Ticket validators work offline for large periods of time and thus, it is required to make use of an offset value for the correct validation of tickets. Furthermore, in the mobile application, the offset value is essential in instances where the the mobile application fails to connect to the internet, to obtain the synchronized time.

Without the calculation of the offset value, the mobile application would be required to be connected to the internet, at all times, which goes against one of the objectives of this system. The mobile application should be able to generate QR codes, without internet connection, even if only for a short period of time.

### **3.4.2 Key Rotation**

Encryption keys are changed regularly to prevent brute-force attacks. As previously mentioned, since ticket validators require very low maintenance, in the event that ticket validators fail to obtain a new private key, used to decipher the contents of the QR code, they cannot properly validate tickets.

With this, instead of receiving a single private key, each ticket validator receives a batch of private keys from the ticketing server. The ticketing server is responsible for the management of the batch of keys and for securely distributing the batch of keys to ticket validators.

In the current iteration of this system, the ticketing server performs the rotation of the encryption keys, in the batch, on a daily basis. This is a security procedure with focus on preventing fraud such as reverse engineering attacks to the QR code, since attackers can attempt to brute-force the encryption key in order to extract more information from tickets and the security mechanisms inside QR codes.

## **3.5 System Architecture**

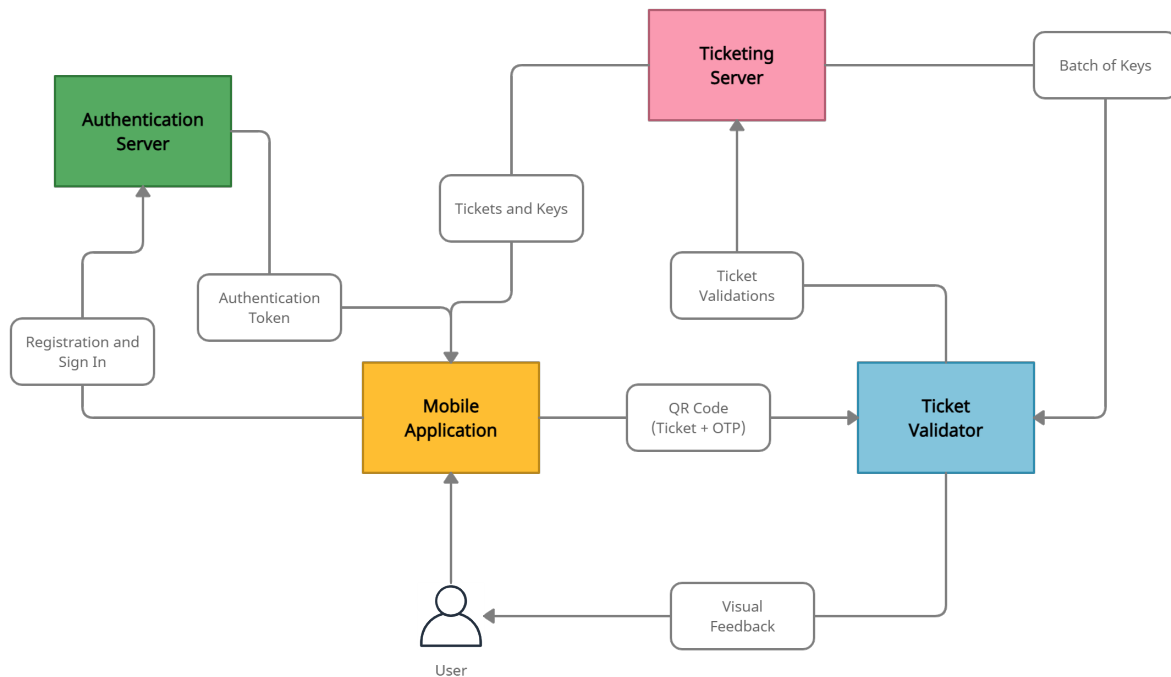
In the figure below 3.4, there is a representation of the system architecture which provides a comprehensive diagram of the relationship between every component of the system. In addition, this section contains a description of how every component interacts within the system as well as their behaviour and responsibilities. In the next chapter, it is discussed the implementation of these components and technologies which allow for the correct functioning of the system.

The mobile application is the only user side entity which allows the user to interact with the system. It contains a login system as well as a dynamic QR code generation system which allows the user to present the QR code to a ticket validator.

As stated above, ticket validators are set up outside the application and operate offline to ensure there are no failures in the communication between ticket validators and the mobile application, and that internet availability is not a limiting factor for the validation of tickets.

Besides that, the mobile application and ticket validators interact with the backend servers, which are the authentication and ticketing servers. The authentication server is used on authentication in the mobile application and ticket validators, and the ticketing server is responsible for the purchase of tickets and the rotation of keys.

The architecture of the system is divided in three phases which are the login system, generation of the QR code and the validation of the QR code.



**Figure 3.4:** Architecture

### 3.5.1 Login and Registration

The login system found in the mobile application allows users to register or authenticate in the system, through the authentication on the authentication server. The authentication server is an OpenID Connect server which emits an authentication token which consists of an access token that is going to be used on the ticketing server, for instance, to authorize the user when trying to purchase a new ticket.

In order to authenticate in the mobile application, it is required for the application to have internet access. However, users do not have to insert their credentials in the application every time they want to use the application, provided that the authentication token is valid. In case the authentication token expires, the mobile application tries to use the refresh token to obtain a new authentication token and upon the expiration of both tokens, users are prompted to reintroduce their credentials.

At the moment of login, the mobile application is required to be connected to the internet and therefore, the mobile application obtains a public key from the ticketing server. This key is used to encrypt the contents of QR codes and the rotation of this key is done, by the ticketing server, on a regular basis, to prevent brute-force attacks against the key. In the current prototype of this system, the rotation of keys occurs once a day, though this value can be subject to modifications, without compromising the correct functioning of this system.

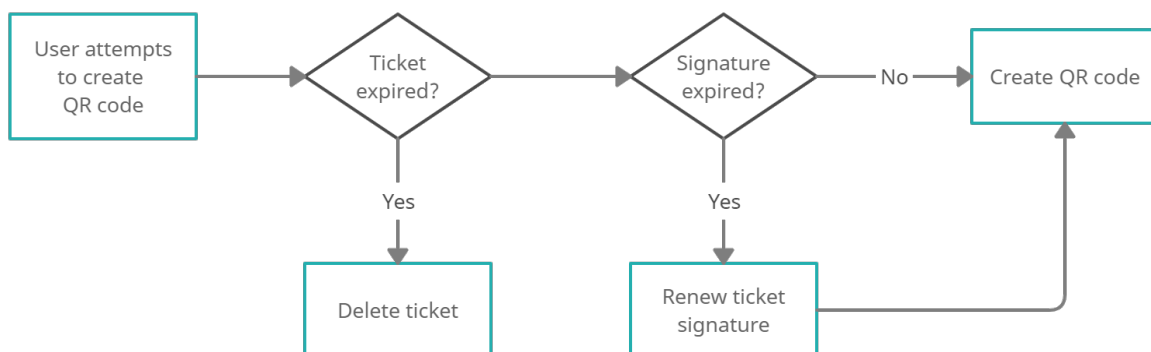
The ticketing server is responsible for the generation of these encryption keys and, in order to allow ticket validators to read the contents of QR codes, ciphered with the public key sent to the mobile

application, the ticketing server sends the corresponding private key to ticket validators. With this, ticket validators are the only entities able to decrypt and thus, validate the QR codes generated in the mobile application. Users that attempt to scan the contents of the QR code are not going to be able to extract information unless they get access to the private key.

### 3.5.2 QR code Generation

Upon entering their credentials and signing in the mobile application, users are presented with the home page which contains tickets that the user has purchased. On the first run of the mobile application, the home page does not contain any tickets and, in order to obtain tickets, users are required to purchase tickets from the ticketing server. Upon purchasing a new ticket, the mobile application redirects the user to the home page.

By clicking on the ticket, the mobile application displays the purchased ticket in the form of a QR code, so that the user can present the QR code to the ticket validator.



**Figure 3.5:** QR code Generation

From the figure 3.5, the process of generating a QR code starts by ensuring that the ticket does not contain expired parameters such as the expiration date and the expiration date of the signature. Upon the expiration of the signature in ticket, the mobile application is able to request for a renewal of the ticket signature from the ticketing server. Eventually, the mobile application proceeds to generate a time-based one-time password (TOTP) based on the ticket and a timestamp of the current time. The timestamp is obtained from a time server, to ensure that there is the synchronization of clocks across the system. Finally, the mobile application displays a QR code which consists of the encryption of the ticket and one-time password.



### **3.5.2.A Expiration dates in QR codes**

A QR code contains three different expiration dates: the expiration date of the ticket, the expiration date of the ticket signature and the period of validity of the TOTP. A ticket is valid for the duration which the user preferred such as a certain number of hours or days, whereas ticket signatures are short lived to ensure that users are regularly authenticated in the system. Time-based OTPs have a very limited period in which they are valid, usually less than one minute, for their one-time usage.

Taking this into account, a QR code is only valid for the period of validity of the OTP, and the mobile application needs to refresh the QR code, by generating a new one, upon the expiration of the previous QR code. This process is also known as the dynamic generation of QR codes since QR codes are constantly being refreshed, to prevent users from presenting invalid QR codes to a ticket validator. In addition, ticket signatures can become expired and the mobile application is able to recover these tickets, by requesting the ticketing server to renew the ticket signature as well as the expiration date of the signature. Tickets that become expired, are no longer accepted in the system and they are removed from the mobile application.

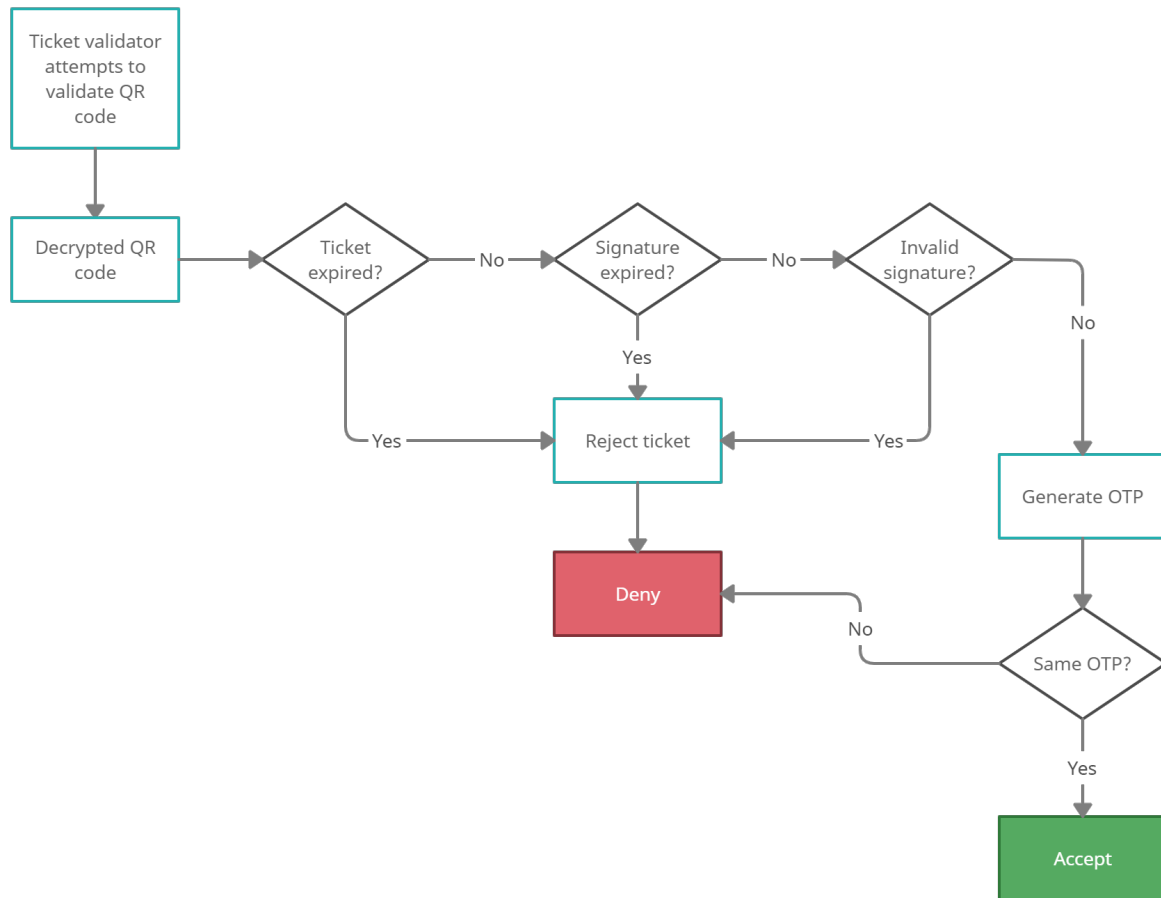
### **3.5.3 QR code Validation**

Ticket validators receive a batch of private keys, from the ticketing server, to ensure that, in case they fail to obtain a new batch of keys, they are not deemed unusable. Even though this scenario is an extreme case and this would require ticket validators not being able to communicate with the ticketing server for many consecutive days, equivalent to the number of keys in the batch of keys.

As aforementioned, the ticketing server is responsible for the emission of the batch of private keys to ticket validators, and this process is required to be a secure process to avoid the stealing of the batch of keys and prevent fraud. Users that get access to the batch of keys, are able to gain control over the system and it is of utmost importance that the batch of keys cannot be accessed by any entities, with the exception of ticket validators. Taking this into account, the ticketing server would supposedly send the batch of keys through a secure network and requests for the batch of keys can only be accepted for entities which are able to obtain the access token of a ticket validator.

As previously mentioned, in cases where ticket validators fail to obtain a new batch of keys, they are prepared to switch to the next private key in the batch of keys, which is synchronized with the corresponding public key, sent to the mobile application. The batch of keys is securely stored in the ticket validator and it is not shared with any other entities.

The process of ticket validation starts by having the ticket validator decrypt the encrypted QR code, using the private key that was received from the ticketing server. The ticket validator records the timestamp as soon as the QR code is shown to the reader and uses this timestamp to validate the expiration



**Figure 3.6:** QR code Validation

dates in the ticket and the OTP. Finally, ticket validators provide visual feedback to the user based on the validation of tickets and the result of the comparison of the OTP in the QR code and the OTP calculated by the ticket validator. After this, the ticket validator records a ticket transaction which is a registry with the information of a ticket along with the timestamp of the validation time.

The recording of tickets is done with a view to later sending these ticket transactions to the ticketing server. Since ticket validators do not have constant access to the internet, ticket transactions are stored in the memory of the ticket validator, and ticket validators attempt to send these tickets transactions, on a regular basis, to the ticketing server. The primary purpose of sending ticket transactions to the ticketing server is to allow for a centralized registry of the transactions made in the system. With this, the system allows for revenue sharing and it is able to gain insights over how users make use of the system and allow for improvements. In addition, the ticketing server contains an anti-fraud system which, by analyzing ticket transactions, this system can detect suspicious user activity and take preventive action.

# 4

## Implementation

### Contents

---

4.1 Mobile Application . . . . .	37
4.2 Ticket Validators . . . . .	48
4.3 Back-end Servers . . . . .	52
4.4 Clock Synchronization . . . . .	58

---



The current implementation of this system is a prototype and since there are various methods of implementation, this chapter contains a detailed discussion of the current method of implementation, by providing an extensive description of the entities of this system, including the technologies used and reasoning behind the use of these.

Alternatively, in cases where it is possible, this paper contains insights over other possible methods of implementation, and how they could have been applied to this system.

## 4.1 Mobile Application

The mobile application is a user side entity which refers to the fact that users are directly associated with a mobile application and use the application to interact with the system.

The system should be able to work independently of the device that the user is carrying and thus, there were developed two versions of the mobile application: one version for Android and the other for iOS devices.

The Android version of the mobile application was developed in Android Studio v4.2.2 and using Gradle build tool v6.7.1 and using the following configurations:

- Target SDK version: API 29 - Android 10.0 (Q)
- Min SDK version: API 26 - Android 8.0 (Nougat)

On the other hand, the iOS version of the mobile application was developed in XCode 12.4 and using the following configurations:

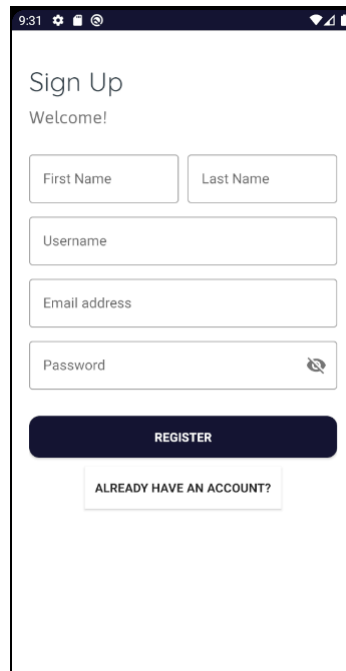
- Swift Language Version: 5
- iOS Deployment Target: 14.4

In order for our Android version of the mobile application to work properly, the required Software Development Kit (SDK) version is 26 since this is the minimum version that is able to use specific methods from Java 8. According to a study on Statista [43] and AppBrain [44], as of June-August 2021, about 15% of internet users have an Android version which is not supported by our prototype of the mobile application.

As stated above, the iOS version of the mobile application provides the same functionalities as the Android version and both versions of the mobile application were tested successfully on different devices such as Samsung Galaxy S7 Edge (Android 8.0, API 26), OnePlus 7T Pro (Android 11.0, API 30) and on iPhone 7 Plus (iOS 14.7).

### 4.1.1 Login and Register

When opening the mobile application, the user is presented with a login system which allows a user to either login to their account or create a new account. In order to access and use the application, users that do not already have an account are required to create a new account, from the registration page 4.1.



**Figure 4.1:** Registration page in the mobile application

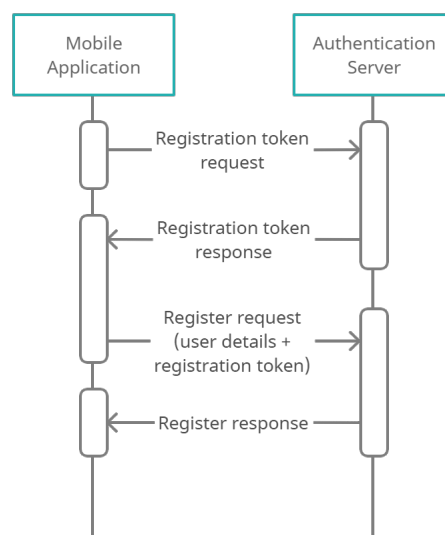
#### 4.1.1.A Register

To create a new user account, users are required to fill certain points of interest using their personal information such as username and first and last names. In this system, the username is the unique identifier of the system and used to login in the mobile application even though, in a different iteration of this system, there is also the possibility of using the email address as the username. The email address is used to validate new user accounts, to prevent an overload of fake accounts, by having the authentication server sending an email to confirm the registration and identity of the user. In addition, this system is not limited to the current information fields with the exception of the username and password and so, the registration form could include other information fields such as a mobile phone number.

Another method to confirm the registration of users could have been implemented using one-time passwords. The authentication server would send a four-digit code through a Short Message Service (SMS) to the mobile phone number of the user and the user is required to introduce the code in the

application. After inserting the code in the application, the mobile application would send the code to the authentication server and in case the code was valid, the account would become verified. Even though the use of one-time passwords provide greater security than sending a confirmation email, the former method was not implemented due to the difficulty in accessing a free SMS system.

The process of registration consists of two steps: obtaining a registration token that allows the application to be able to register a user by sending the user details to the authentication server 4.2. A registration token is an access token, more specifically a bearer token, emitted by the authentication server which contains the required permissions to register new users. The purpose in obtaining a registration token is to allow users to self-register instead of having an administrator manually register the account on the behalf of the user. This is specific to the authentication server and in order to let users to self-register without the need of a registration token, it is required to have a different authentication server as the current one does not allow registration without a registration token.

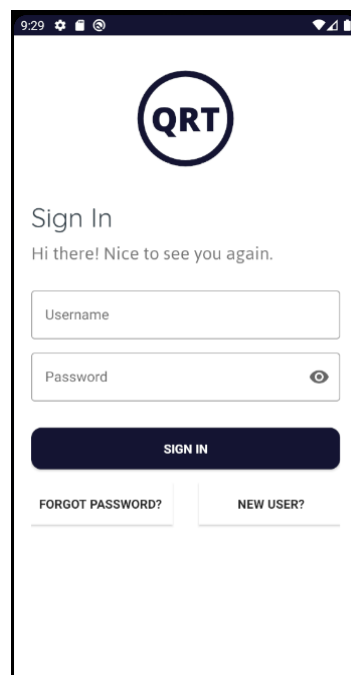


**Figure 4.2:** Registration process sequence diagram

Upon receiving the registration token, the mobile application sends a registration request to the authentication server. The request contains the registration token as well as the user information. Upon receiving a response from the authentication server, the mobile application displays a message containing the status of whether the creation of user account was successful or not. Due to the fact that the username must be unique, users that attempt to register a new account with a username that is found in the authentication server, meaning that it has already been used, are presented with an error message to alert users that they should pick a different username.

### 4.1.1.B Login

After completing the registration process, users can proceed to sign in the application. As it can be seen in the login form page below 4.3, in order to sign in the mobile application, users are required to insert their username and password. Upon signing in, the mobile application proceeds to send a login request to the authentication server. The login request consists of an Hypertext Transfer Protocol Secure (HTTPS) request, using the Retrofit2 library on Android and the Alamofire library on iOS, sent to the authentication server.



**Figure 4.3:** Sign in page in the mobile application

HTTPS requests provide greater security than HTTP requests and they are the current standard in online communication. In this prototype of the system, we are not using point-to-point HTTPS communication due to the difficulty in obtaining a certificate from a Certificate Authority (CA), to associate to the back-end servers. This system was developed using *ngrok* to implement a one-sided point-to-point HTTPS communication, since *ngrok* allows for HTTPS communication from the mobile application to the *ngrok* servers which are then to send HTTP requests to the back-end servers, and vice-versa. With this, the mobile application is not susceptible to interception and eavesdropping attacks such as man-in-the-middle attacks when receiving tickets from the ticketing server.

Since the authentication server is an OpenID Connect server, the authentication server returns an authentication token. An authentication token is an access token, a JSON Web Token (JWT) which remains valid for one hour, since this is the recommended time according to OAuth2.0. This token is to be used only for the current user and it is required in various services provided by the mobile application



such as the purchase of tickets. In addition, users are not required to insert their credentials every time they open the mobile application, as long as the authentication token remains valid. In case the authentication token expires, users are prompted to reintroduce their credentials to authenticate, which requires access to the internet.

In a different iteration of this system, the mobile application can make use of refresh tokens to lessen the amount of times users have to reintroduce their credentials in the mobile application. Upon the expiration of an authentication token, the mobile application would send the refresh token to the authentication server and if valid, the authentication server would emit a new authentication token and refresh tokens. Having a login system which implements a refresh token does not bring advantages from a security point and it would still require internet access despite being a convenient and practical feature.

### **4.1.2 Purchase of tickets**

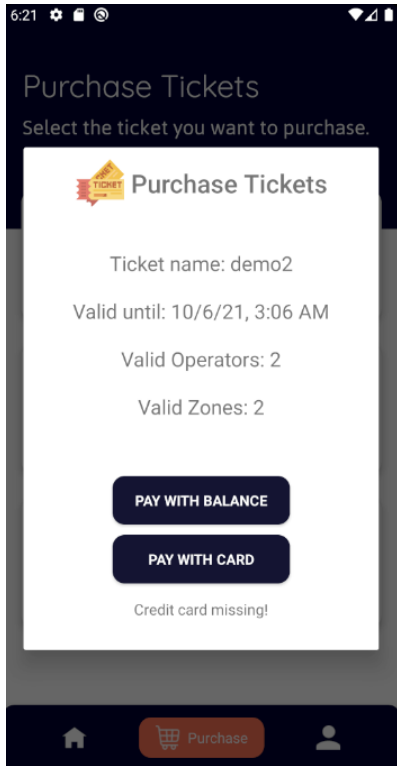
Firstly, in order to purchase of new tickets, the mobile application retrieves the list of available tickets, from the ticketing server. Only authorized users or users that hold a valid access token, with specific permissions 4.3, are able to access the list of available tickets and purchase new tickets.

The payment method was considered to be an area which was out of the scope of this system and therefore, this system contains a simulation of the payment of ticket. There are various methods of payment and different approaches to purchasing tickets which would be suitable to implement on this system. The most widely used approaches to the purchase of tickets, consist on the implementation of a system in which user accounts are associated with an account balance and users can also purchase tickets using their credit cards.

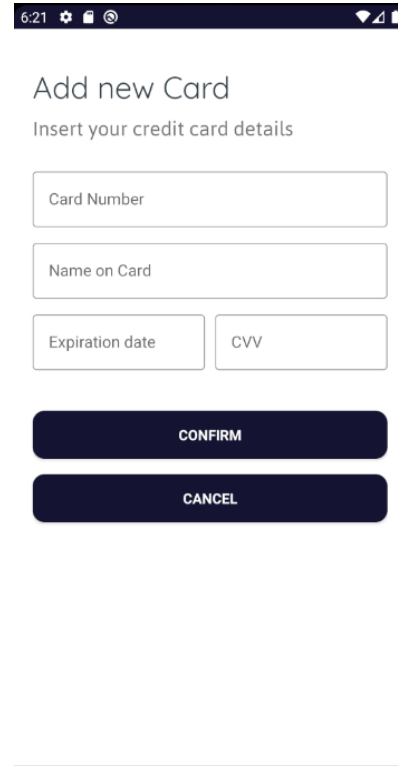
A payment system based on the account balance is more convenient to users who purchase tickets regularly such as locals or those who travel to work on a daily basis, since they are only required to have sufficient account balance to purchase tickets whereas the payment per ticket method is more convenient to those who use the system occasionally such as tourists, since they are usually looking for a ticket to arrive to their final destination and they are not looking forward to spending a lot of time using the system.

The current prototype of the mobile application contains an hybrid approach to the payment of tickets, in order to be more convenient to the end user. Upon receiving a catalog of products from the ticketing server, users are able to purchase their desired ticket.

From the figure 4.4, the mobile application prompts users with a confirmation page which includes the payment methods for purchasing tickets. The current available options allow users to either pay with their account balance or to pay for tickets using their credit card. As it can be seen in the figure 4.5, users can fill the form fields with their credit card information, and confirm the addition of the credit card to the mobile application. After setting their credit card information, subsequent purchases do not require



**Figure 4.4:** Payment confirmation page



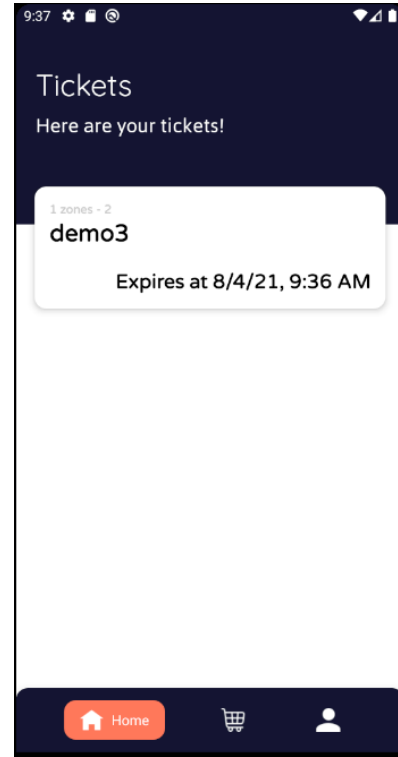
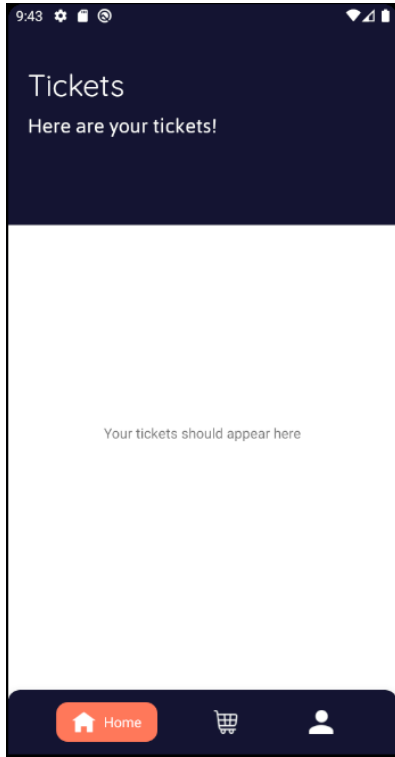
**Figure 4.5:** Add new card page

the re-introduction of credit card information as to simulate the storing of credit card information in the mobile application and, due to simplicity reasons, it is assumed that no other credit cards are going to be added to the mobile application.

Upon selection, the mobile application sends a purchase request to the ticketing server, which returns a ticket to a valid request. A purchase request consists of a HTTPS request to the ticketing server which validates the request and returns a ticket to the mobile application. The ticket is then securely stored in the internal storage of the mobile phone.

In addition, in case users do not have sufficient account balance, the application attempts to make use of the credit card inserted to purchase a new ticket. Alternatively, the mobile application could present the options such as recharging the account balance. As aforementioned, the mobile application contains a simulation of the payment method and, in a real environment, recharging the account balance and the payment per ticket method would require a payment system which would make a purchase request to a financial services company such as PayPal, MB Way and VISA.

Finally, the mobile application redirects the user to the home page. The home page contains a list of valid tickets that have been previously purchased, as seen in the figure 4.7.



**Figure 4.6:** Empty home page in the mobile application **Figure 4.7:** Tickets in the home page of the application

#### 4.1.2.A Storage of tickets

After purchasing tickets from the ticketing server, the mobile application stores them in the internal storage of the mobile phone. However, in the event that the mobile phone is victim to an attack, the attacker could try rooting the mobile phone to steal the tickets in storage and use them in a different device.

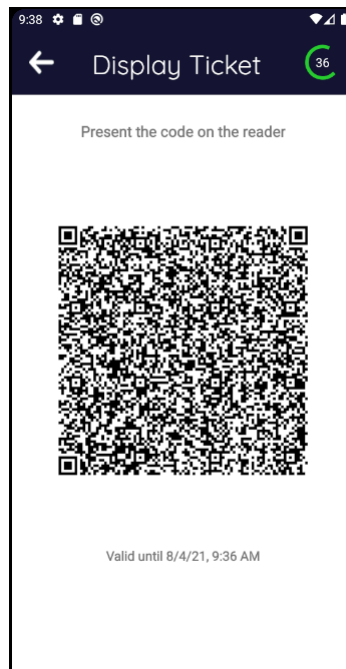
Taking this into account, it is important to store encrypted tickets instead of storing tickets in plaintext and, in order to store encrypted tickets, the mobile application takes advantage of Android Security Crypto library which contains the MasterKey and EncryptedFile classes. The Android MasterKey class is used to reference a key which is stored in the Android Keystore and to encrypt data encryption keys for encrypting files. The Android EncryptedFile class is used to create and read encrypted files and requires a MasterKey to encrypt files. The algorithm used to encrypt create a MasterKey and an EncryptedFile is AES-256 in Galois/Counter Mode (GCM) mode, since it is the recommended algorithm on the official Android developers webpage. Using AES with bigger key sizes, such as 256 bits, is more secure against brute force attacks and GCM mode provides authentication and is more parallelizable hence being faster than Cipher-Block Chaining (CBC) mode.

Since tickets are encrypted before being stored in internal storage, tickets that were stolen and used in a different device are not able to be used to display tickets in the form of QR codes.

### 4.1.3 Display tickets

The ticket validator reads tickets in the form of QR codes. From the home page of the mobile application, users can select their desired ticket to display a ticket in the form of a QR code.

For the generation of a QR code, the mobile application creates an encrypted JSON object which contains the ticket and TOTP. Initially, there was the idea of creating a QR code which included the authentication token as well as the ticket and TOTP. However, this token drastically increased the size of the QR code and made the process of validation much more difficult. Taking this into account, there were improvements to shorten the size of the QR code as much as possible (discussed below), with the creation of digital signatures and encryption keys used to encrypt the contents of the QR code also being taken in consideration, in order to reduce the key size without compromising the security of these keys (section 4.3).



**Figure 4.8:** Display ticket page

As stated above, TOTP's are only valid for a short period of time and, in the event that the current TOTP expires, the mobile application dynamically generates a new QR code which is valid for the duration of the new TOTP. In this version of the mobile application, each TOTP and QR code lasts for 60 seconds since TOTP's should only be valid for a very short period of time but also, it is taken into account that users might face delays in connection or in the creation of the QR code. Alternatively, this system could have been implemented using a different period of validity or timestep such as the current recommended timestep in the RFC6238 [31], which is 30 seconds.

With the dynamic generation of QR codes, the mobile application contains a feature which allows



**Figure 4.9:** Green progress bar



**Figure 4.10:** Yellow progress bar



**Figure 4.11:** Red progress bar

users to visualize when to use the current QR code or wait for the creation of a new one. This is to ensure that users are not met with the rejection of the presented ticket to the ticket validator. As seen in the figure 4.11, users should wait for the creation of a new QR code since the current one is due to expire soon. Presenting a valid ticket to a ticket validator on either a yellow or green progress bar indicates that the TOTP remains valid for a period of time which, in case of network delays, would not present a problem to the validation of tickets.

In the event that the OTP expires, the application dynamically creates a new QR code which includes the ticket and the newly calculated OTP value. However, in the event that neither the ticket nor the ticket signature are valid, the mobile application acts accordingly to avoid displaying an invalid QR code. In case the ticket signature expires, the mobile application is able to obtain tickets with a fresh signature by sending a recover signature request to the ticketing server. A recover signature request consists of a request to the ticketing server to renew the ticket signature as well as its expiration date to ensure authenticity. Despite the fact that the expiration date of the signature in the ticket is usually much shorter than the expiration date of the ticket (3), the ticketing server validates the request and verifies the ticket to avoid recovering expired tickets. In case the ticket expires and not the signature in the ticket, the mobile application does not need to inform the ticketing server and proceeds to delete the ticket from the internal storage and it is no longer displayed on the home page.

By having previously calculated the offset to the time server and having a valid authentication token, the mobile application is able to work offline for a brief period of time. This is not the focus of the system but there can be instances where the mobile application temporarily loses access to the internet and blocking users from being able to display tickets could pose as a threat to the functionality of this system.

The final step in the creation of the QR code consists of the encryption of the QR code contents, namely the ticket and TOTP. Before encrypting these contents, the mobile application compresses the message using the GZIP library, in order to reduce the contents of the QR code and making QR codes more readable. The message is encoded using Base64 to avoid losses in content of the message and to prevent decoding errors that might occur when creating a QR code. In order to encrypt the QR code contents, the mobile application makes use of the public key of the ticket validators which is provided by the ticketing server and shared in the network to all users. To generate tickets in the form of QR codes, the Android version of the mobile application uses the ZXing library (Core v3.4.0). ZXing ("Zebra

Crossing”) is an open-source barcode image processing library implemented in Java which provides an easy to implement interface to generate QR codes and show them in bitmap format. The iOS version uses UIImage which is equivalent to the ZXing library with the exception of being a built-in class in iOS.

Having the QR code displayed on the mobile application, users are only required to present the QR code to a ticket validator in order to validate the ticket and be able to use the system.

#### **4.1.3.A Trip history**

A feature that could have been implemented in the prototype of the mobile application is the ability to see expired tickets. This is a convenient feature since some users would appreciate being able to see a list of the previous tickets that were purchased by selecting an option to see the list of expired tickets as well as information regarding the expired ticket such as the date of expiration and other ticket details.

However, this feature was not considered to be an area of focus and thus, it was not implemented in this prototype version of the mobile application. As stated previously, the mobile application removes expired tickets from the internal storage and does not display expired tickets in the home page.

#### **4.1.4 Screenshot blocking**

Screenshots are methods of sharing information and applications that make use of sensitive information or copyrighted content such as banking applications, usually disable screenshots in the mobile application. In this system, double spending is one of the main attacks to the mobile application and one way to perform double spending attacks is by screenshotting the QR code with the ticket and sending the QR code, through the internet, to another user.

The Android version of the mobile application does not allow users to take screenshots, when displaying a ticket. Conversely, the iOS version of the mobile application does not permit the blocking of screenshots and, in order to prevent users from performing double spending attacks, the iOS version of the mobile application can make use of an external framework such as ScreenShieldKit. ScreenShieldKit blocks sensitive content from a screenshot that is taken to the application screen but, since ScreenShieldKit is not an open-source library, it was not implemented and tested.

#### **4.1.5 Root checking**

The mobile application implements root checking capabilities to prevent users from rooting the device and making use of the mobile application in unintended ways. Besides that, the mobile application contains sensitive information, e.g. tickets, which are stored in the internal storage of the mobile phone device and devices that have root capabilities, are able to bypass security checks that are imposed to prevent users from accessing private information.

To do so, the Android version of the mobile application uses the RootBeer library [45] which checks for indications of root and in case it finds that the device has been rooted, the mobile application does not allow the user to further use the mobile application. In the iOS version of the mobile application, it is implemented similarly to the Android implementation, without the use of a library and checks whether or not the system can execute root tasks, contains suspicious applications or paths and has Cydia installed, which is similar to the App Store with the exception of supporting Jailbroken devices.

#### **4.1.6 Code obfuscation**

Additionally, the mobile application tries to prevent reverse engineering attacks by using code obfuscation. Reverse Engineering attack consists of opening up an object to study its mechanisms which can be used in order to extract valuable information on how it works.

The Android version of the mobile application implements ProGuard, which is a Java open-source cross-platform tool. ProGuard is a command-line tool that shrinks, optimizes and obfuscates the code. ProGuard identifies unused classes, fields and methods attributes of applications and removes them as well as gives them short meaningless names. In addition, ProGuard analyses and optimizes the bytecode.

For the iOS version of the mobile application, due to the difficulty in using an open-source library that met our requirements, there is a simplistic implementation of code obfuscation, which is applied to sensitive data such as private credentials, stored in the application. These values were encrypted beforehand and stored as bytes and, when using these values, the mobile application performs the decryption of these values, to obtain the original data. The current method of obfuscating code is not fully secure since it lacks the use of a random salt value and therefore, the same key is used to cipher and decipher the contents of every sensitive field. Besides that, this implementation can only be applied to protecting sensitive fields whereas, in the case of ProGuard, code obfuscation is also applied to class names and methods, etc. To further implement code obfuscation in iOS and in order to meet the proposed requirements, it would require the use of commercial libraries such as iXGuard.

## 4.2 Ticket Validators

This section details the ticket validator implemented in this system as well as to apply the ideas previously discussed in the previous chapter (3). A ticket validator is responsible for the validation of tickets and emits visual feedback to the user, based on the status of the validation. In addition, the ticket validator controls the rotation of keys in the batch of keys, sent from the ticketing server, in order to remain synchronized with the system. Additionally, a ticket validator is responsible for the synchronization of time with a time server and therefore, remain synchronized with the rest of the system.

The ticket validator was developed in IntelliJ IDEA 2021.1 and using Java 8 and it is able to work offline for certain period of time. In case ticket validators do not have access to the internet for a long period of time, the batch of keys inside the ticket validator is not updated and the validator does not have the correct key, for the validation of tickets, on that specific day. This is considered to be the most extreme situation and it is assumed that ticket validators are going to obtain new information from the back-end servers, at least once, before every key in batch of keys becomes expired.

### 4.2.1 Authentication

In order to obtain information from the ticketing server, ticket validators are required to provide a valid access token which allows for the authentication of the ticket validator. With this, a ticket validator starts by requesting an access token to prove its authenticity to the authentication server, using their credentials. The authentication server returns an access token which is specific for ticket validators, with a validity period of 60 days. Since ticket validators are only required to access the ticketing server for a limited number of times, having to constantly obtain a new access token would lead to unnecessary calls to the authentication server and thus, this system implements ticket validators with long-lived access tokens. In addition, the access token contains permissions that are only specific to ticket validators in order to prevent other clients from accessing confidential information such as permissions to obtain the batch of keys from ticket validators, which are used to decrypt the contents of tickets in the form of QR codes.

### 4.2.2 Ticket validation

The process of validation of tickets starts upon reading the QR code contents, from the scanning of the QR code in the ticket validator. In order to read QR codes, the prototype version of the ticket validator application starts by creating a webcam, using the Webcam library by Sarxos, and creating a display which is used to display visual feedback.

The figure below 4.12 contains a comprehensive diagram of the extraction of tickets from QR codes. As stated above, to avoid losses in content of the message when generating QR codes, the mobile



application encodes the message to Base64 and the ticket validator decodes its value to obtain the original message. After decrypting the content of the QR code, the ticket validator proceeds to decrypt the encrypted message. The encryption phase occurs in the mobile application, using the public key associated with the ticket validators on a specific day, and the ticket validator uses the corresponding private key from the batch of keys, to decrypt the contents of the message.



**Figure 4.12:** Extraction of tickets from QR codes

Following the decryption of the message, the ticket validator proceeds to decompress the contents of the message using the GZIP library. The process of compression is executed in the mobile application, in order to reduce the size of QR codes.

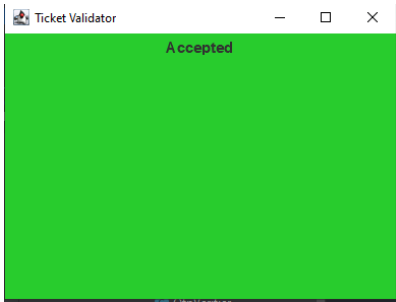
In the event that these steps are successful, the ticket validator obtains a JavaScript Object Notation (JSON) in a string format, and converts it to a class using the Gson library from Google. By converting the string into a class, the ticket validator can obtain the contents of the JSON from the class attributes such as the ticket and OTP.

The next step in the validation of tickets is the validation of the contents of the ticket. As seen in the figure 3.6, tickets contain three parameters which are used for the validation of tickets: the expiration date of the ticket, the expiration date of the signature and signature. The expiration date of the ticket is the date until the ticket is valid and upon expiration, users are required to purchase a new ticket. The expiration date of the signature is the date until the signature of the ticket is valid and requires a renewal of the ticket signature, from the ticketing server. The signature in the ticket is used to ensure the integrity and non-repudiation of the ticket, including the fact that the ticket was emitted by the ticketing server.

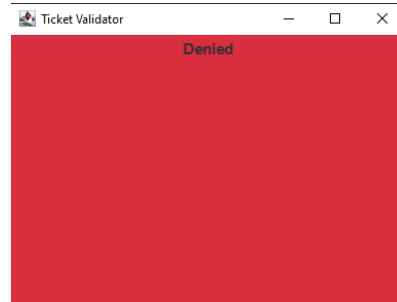
Following the verification of these parameters, the ticket validator proceeds to verify whether or not it had previously accepted that QR code, to prevent cloning attacks. Cloning attacks are attacks where the attacker makes a copy of the QR code such as taking a photo of the QR code, and attempts to use the ticket in order to make use of the system. Taking this into account, ticket validators store a history of validations, in memory (for simplicity reasons), and rejects QR codes that have been previously used.

In case these steps are successful, the ticket validator creates a OTP and compares it with the OTP of the ticket. The method of creating an OTP by the ticket validator is equal to the method used to create the OTP in the mobile application, and if these values are equal, the QR code is accepted by the ticket validator.

Ticket validators emit visual feedback to the user, based on the validation of tickets, since ticket validators contain an interface for users to easily identify whether or not the ticket presented was deemed



**Figure 4.13:** Ticket accepted



**Figure 4.14:** Ticket denied

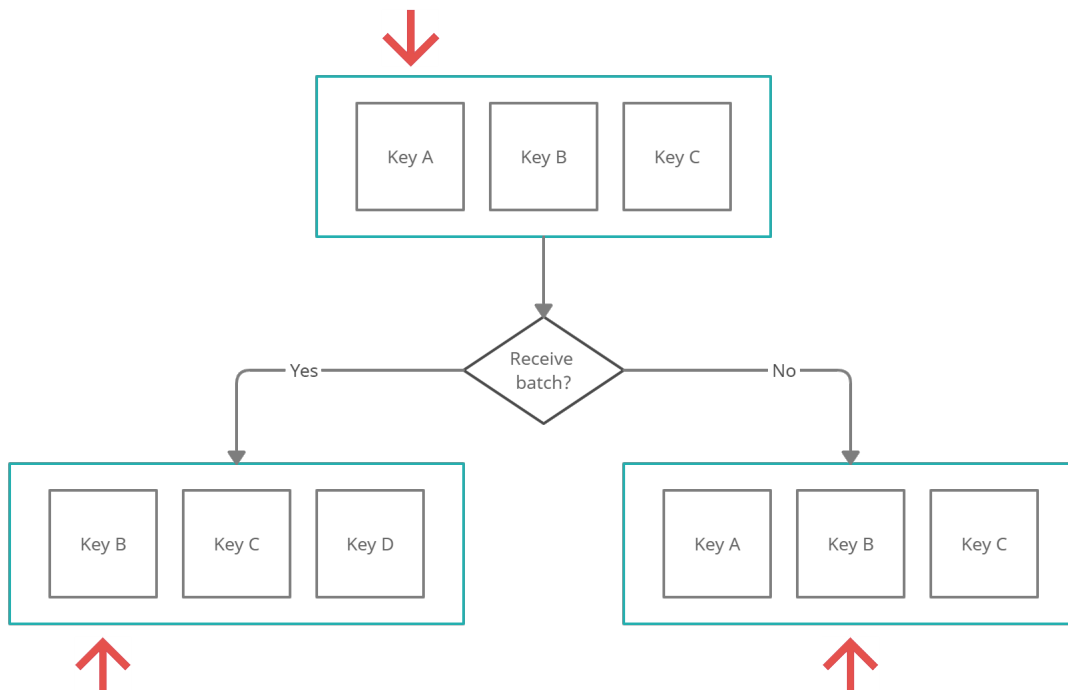
valid. As seen in the pictures below, the ticket validator accepts the ticket (in the figure 4.13) and rejects the ticket (in the figure 4.14).

### 4.2.3 Key rotation

Since ticket validators do not have constant access to the internet, the ticketing server emits a batch of keys instead of sending a single key to the validators. A batch of keys consists of a map of private keys, identified by the date in which each key is supposed to be used. Each key consists of a Base64 encoded KeysetHandle to avoid losses and prevent the corruption of messages, which contains a key generated in the ticketing server using the Google Tink library. In this version of the system, ticket validators use a ScheduledExecutorService to obtain a new batch of keys, everyday at midnight, and this request is executed once a day. Despite this, these values can be subject to change in another version of the system without compromising functionality.

The figure 4.15 contains a visual representation of the process of rotating keys in ticket validators. To begin with, the ticket validator obtained a batch of keys beforehand and proceeds to rotate the key which is currently in use (Key A), with the rotation of keys being based on whether or not the ticket validator was able to access the ticketing server to obtain a new batch of keys. Having determined that the upcoming key is Key B, the ticket validator proceeds to select Key B from either the new batch of keys or, in case it did not have access to the internet, from the old one. From the figure, we can see that a new and updated batch of keys, from the ticketing server, does not contain the now expired key (Key A) and contains a new key (Key D). On the other hand, the old batch of keys is not updated and the ticket validator is able to select the key that determined to be next in line (Key B).

Ticket validators determine which key is going to be used next based on the date associated with the key, and this process is synchronized across the system. Additionally, since the rotation of keys occurs once a day, the date associated with the ticket is the specific date on which the key is going to be used by the validator. Failures in the synchronization of dates may lead to ticket validators failing to synchronize with the system and eventually fail to properly decrypt and validate tickets.



**Figure 4.15:** Key rotation in ticket validators

Besides receiving the batch of keys, ticket validators receive the signing key of the ticketing server, and this key is also associated with an expiration date. Upon the expiration of the signing key, ticket validators fetch the new signing key from the ticketing server.

#### 4.2.4 Ticket transactions

Upon accepting a ticket from being valid, the ticket validator stores a recording of the validation of a ticket in storage. This is called a ticket transaction and it consists of the identifier of the ticket and the instant of validation. These are to later be forwarded to the ticketing server so that system administrators can obtain valuable information regarding the usage of tickets and prevent fraudulent behaviour.

The forwarding of ticket transactions is done asynchronously and, in the current version of ticket validators, at the same time that ticket validators obtain a new batch of keys (midnight of every day). Similarly to the obtaining updated batch of keys, the time and frequency of sending ticket transactions to the ticketing server can be subject to change. After receiving confirmation that ticket transactions were successfully received by the ticketing server, the ticket validator proceeds to delete them from storage.

## 4.3 Back-end Servers

This section details the back-end servers and their specifications in this system since these are used to manage back-end processes such as the communication with the client, more specifically, the mobile application. In addition, the main goal of a server is to provide functionality and information for a lot of users since servers are equipped with more storage and processing capabilities. Taking this into account, there are some processes which can be managed by the back-end servers instead of the mobile application.

As stated above, in chapter 3, users do not interact with the back-end servers physically. These servers are located in the back-office and they provide information such as keys and tickets to the mobile application.

This system contains two different back-end servers: the authentication server and the ticketing server. The former is used for authentication in the system and the latter is used to the distribution of keys and tickets to the client.

### 4.3.1 Authentication Server

This version of the system makes use of a server from Keycloak [46] which is an open source identity and access management solution, and it takes care of the authentication services such as the registration and login of clients in the system. This system includes a token-based authentication protocol which consists of an authentication system which allows clients to verify their identity and emits a unique access token to a successfully authenticated client.

An access token is a JWT which contains several fields of information that are required for the correct functioning of the system, including user roles and user permissions, which can either be applied to a specific user or instead, all users. The figure below 4.16, contains a table with relevant fields to the implementation of this system.

To start off, this system contains realm roles or roles that identify the different types of clients: user and validator, which refer to the user which makes use of the mobile application and ticket validators, respectively. With this, the ticketing server is able to implement client-specific methods and reject requests from access tokens which do not have the required roles. Additionally, the authentication server allows for more fine-grained permissions such as the ability to obtain the catalog of products or the permission to purchase tickets. In this system, client roles are implemented with relation to the only audience member (ticketing server) and an example of a situation where one would put this into practice could be that a certain user is deemed of fraudulent behaviour and the system would, consequently, prohibit the purchase of new tickets from the ticketing server.

This system allows users to self-register from the mobile application and thus, the process of regis-

Field in the access token	Definition	Default value
Expiration time (exp)	Expiration time of the JWT	60 minutes
Issuer (iss)	Who created the JWT	Authentication Server
Audience (aud)	Who the token is intended for	Ticketing Server
Subject (sub)	Identifier of the user	-
Realm roles	Roles that dictate the type of client	user
Ticketing Server Client Roles	Roles associated with the client, for the ticketing server	view-information, buy_ticket

**Figure 4.16:** Relevant fields in the access token

tration in this system is made solely by the user and it does not require the intervention of an authority which would take care of the registration process, on behalf of the user. Instead, in order to allow users to self-register from the mobile application, the authentication server requires users to be authorized and thus, the mobile application needs to request a registration token beforehand. The registration token is an access token with permissions to perform the action of registration in the server and, in this version of the system, we use the default registration token from Keycloak, for simplicity reasons, which contains permissions to manage users in the system, it is valid for 10 minutes and solely used for the registration process. Upon receiving this token, the mobile application makes a registration request which includes the registration token along with user credentials and information that is specified in the register fields. The registration request is processed by Keycloak and creates a response based on the status of whether or not the user was registered in the server.

Users that have already registered in the system can proceed to use their credentials to login in the system. In case their credentials are valid, the authentication server emits an authentication token which is an access token that can be used on the mobile application and ticket validators, to make requests to the ticketing server. This token is only valid for 60 minutes and, to avoid having to re-introduce their credentials every 60 minutes, the authentication server includes a refresh token in the login response, with a period of expiration of 30 days. In addition, the access token in ticket validators is valid for 1 day since access tokens are used to authenticate in the ticketing server, on a daily basis, and the period of expiration of the refresh token is 30 days. The period of expiration of the refresh token is much longer than the access token since its only purpose is to avoid the inconvenient task of having to re-introduce user credentials and instead, use the refresh token to obtain a new access token. When requesting a new access token using a refresh token, the authentication server emits a new refresh token which can later be used for the same process.

## **4.3.2 Ticketing Server**

This server was developed using Spring Boot and Java 11, and it is considered to be a resource server since it handles requests upon the verification of the access token, emitted by the authentication server. Using Spring Boot Security and the Java-JWT library from Auth0, the ticketing server is able to read and validate the access token, created in the authentication server.

### **4.3.2.A Access token validation**

The ticketing server verifies if the access token was created by the authentication server and rejects requests that contain forged access tokens. In order to verify the authenticity of access tokens, the Java-JWT library obtains the list of certificates from the authentication server and compares the signing key id with the signing key id found in the header parameter of the JWT. Upon receiving a JWT that was created in the authentication server, the ticketing server proceeds to validate the parameters in the token such as the header, signature and payload contents. The validation of the header consists on the validation of the type of token, which is required to be a JWT token, and the validation of the signature consists on the verification of the authenticity of the signature, with the public key of the authentication server. Since the header parameter contains the identifier of the signing key, the ticketing server is able to identify the public key that was used for the signature and obtain its value, from the authentication server, which is required in case the authentication server uses multiple signing key pairs. Following the verification of the header and signature parameters, the ticketing server proceeds to validate the contents of the payload parameter such as the expiration date, audience and roles.

Upon verification of these parameters, the ticketing server proceeds to execute the request of the client and, depending on the request, the ticketing server rejects requests based on the roles in the access token. This ensures that unauthorized parties are not allowed to obtain confidential information and if it were allowed, the reliability of this system could become compromised. As an example, users are not able to request batches of private keys, from the ticketing server, since this service is only provided to ticket validators. In addition, as stated above, the authentication server allows for including more specific roles in the access token to prevent clients from executing a certain action that would be allowed previously such as preventing users with fraudulent behaviour from obtaining new tickets and keys.

### **4.3.2.B Services**

After validating the access token, the ticketing server is able to execute requests for users and ticket validators. In light of this, the services provided by the ticketing server are split into three categories: services provided to users only, services provided to ticket validators only and mutual services.

User services	Validator services	Mutual services
Purchase ticket	Obtain batch of keys	Obtain public key of the ticketing server
List available tickets	Send ticket transactions	-
Obtain public key of the validator	-	-
Recover ticket signature	-	-

**Figure 4.17:** Table with the services provided by the ticketing server

In order to purchase a new ticket from the ticketing server, the mobile application makes a purchase request to this server which includes the product listed in the available tickets list. The ticketing server creates a new identifier for the ticket using a new id created by the UUID class, uses the parameters of the product to fill the rest of the fields in the ticket and creates a signature of the ticket which is inserted in the signature field of the ticket. Before sending the ticket to the client, the ticketing server updates a HashMap of tickets which contains all the tickets for every user so that it is able to obtain the details of a particular ticket, for recovery purposes.

Each signature in the ticket is valid for 60 minutes so that users are required to provide a form of authentication to the ticketing server on a regular basis. This operation is called the recovery of tickets and the ticketing server re-signs tickets with an expired signature and forwards them back to the owner of the ticket. This process starts with the validation of the ticket fields such as verification of the signature and that the ticket is not expired, since failures in the verification of these parameters would mean that the ticket was either forged or expired and no longer usable, respectively. Besides that, the ticketing server validates if the ticket belongs to the user that is making the current request and that the ticket is located in the tickets map. In case these conditions are met, the ticketing server creates a new signature for the ticket, updates its entry in the tickets map and sends the ticket back to the user.

The ticketing server is also able to receive ticket transactions, from ticket validators, and these are to be stored in storage. In this prototype of the ticketing server, ticket transactions are stored as a JSON file, for simplicity reasons, in the storage of the ticketing server. However, a different version of the ticketing server could implement a database schema or a data warehouse with a view to later perform analysis over ticket transactions. Upon having successfully registered these ticket transactions and stored, the ticketing server emits an OK response to the ticket validator which sent the transactions.

The obtaining of a batch of keys and the public key of the validator as well as the public key of the ticketing server are services which include encryption keys and signing keys, respectively, and these are mentioned in the sections below.

#### **4.3.2.C Creation of signing keys**

Signatures are created using ECDSA [47] which is a digital signature algorithm that uses Elliptic Curve Cryptography. Elliptic Curve Cryptography or ECC provides similar security to using RSA keys whilst having much smaller keys in size comparison, which leads to the having tickets with smaller signatures and consequently, tickets have smaller size. To generate signing keys, the ticketing server makes use of the Tink library by Google, and uses the NIST curve P-256. Even though this curve is the only curve supported by Tink at this moment in time, later iterations of this system could have different curves supported, and their use would not compromise the correct functioning of this system.

In this system, signing keys are not subject to change and do not expire. These keys are associated with the ticketing server and their purpose is for the ticket validators to be able to verify the integrity of tickets, emitted by the ticketing server. For simplicity reasons, the rotation of these signing keys was not implemented, and the ticketing server makes use of a single signing key. In case the ticketing server would make use of multiple signing keys, it would be required to include an additional field in each ticket, with the identifier of the signing key.

#### **4.3.2.D Creation of encryption keys**

Besides having signing keys, the ticketing server is also responsible for creating and securely distributing encryption keys which are used in the encryption of the contents of QR codes, generated by the mobile application. These keys are also ECC keys but instead of using asymmetric cryptography, the ticketing server creates hybrid encryption keys. By using hybrid cryptography, when a message is ciphered in the mobile application, the combination of symmetric and asymmetric keys removes some of the drawbacks of encrypting large contents with asymmetric keys, since these are limited to a maximum size of the message and the slow speed of encryption. Taking this into account, the ticketing server also uses the Tink library by Google, to create these encryption keys which are to be forwarded to the mobile application (Tink is available for Java and Objective-C). These keys are generated using ECIES, from a combination of the curve P-256 from NIST for the asymmetric key and AES 128 for the symmetric key. These keys include Authenticated Encryption with Associated Data (AEAD) for confidentiality and authenticity of the key, and HMAC-based Key Derivation Function (HKDF) for extracting a pseudo-random key from HMAC-SHA256.

Encryption keys are changed on a regular basis, to prevent brute-force attacks and attacks against QR codes. Despite being changed on a daily basis (every 24 hours), this system can be adapted to work with a different key rotation period which does not make the system susceptible to the attacks aforementioned. The purpose of encryption keys is to ensure that the only entity with the ability to read tickets are ticket validators and in order to do so, ticket validators receive batches of keys, from the ticketing server.



The ticketing server updates these batches of keys similarly to using First In, First Out (FIFO) on a stack and therefore, removing an expired encryption key and adding a new one that goes to the end of the batch. An encryption key is associated with the synchronized date from the system, which acts as an identifier of the key in the batch of keys. When changing to the next day, the ticketing server removes expired keys and adds the missing number of keys to the batch. In addition, since it knows the current date of the system, the ticketing server uses it to fetch the public key of the encryption key pair and shares it with the network.

In this system, the size of the batch of keys is 5 and it allows for ticket validators to remain usable for 5 days without receiving another batch of keys from the ticketing server. Despite this, it is assumed that ticket validators are able to obtain a new batch of keys regularly and should receive a new batch of keys much prior to becoming unusable. In addition, the ticketing server can change the size of the batch of keys without compromising the functionality of the system.

#### **4.3.2.E Anti-fraud System**

The ticketing server receives ticket transactions from ticket validators with a view of keeping a centralized registry of the validation of tickets and to prevent fraud. The ticketing server could contain an anti-fraud system which analyses these ticket transactions, by taking a look at the identifier of the ticket used to validate in the ticket validator and the timestamp in the ticket transaction or the time that the ticket was validated.

The anti-fraud system was not implemented for simplicity reasons and since it is out of the scope of the system. This system is able to detect fraudulent behaviour in cases where it is possible to detect multiple usages of the same ticket, through comparison of the validation timestamps.

In the case of a public transport system, the anti-fraud system can detect suspicious cases of ticket validations in instances where the time of validating tickets does not match the minimum required time to validate a certain ticket. Using the metro as an example, in case a ticket is validated in two different stations in a shorter period of time than the minimum time it takes to complete the journey from one station to the other, the anti-fraud system recognizes this as fraudulent behaviour and takes preventive action.

To prevent users with fraudulent behaviour from using tickets, the anti-fraud system notifies the ticketing server and this server does not renew the signature in the ticket used. Even though this preventive measure can successfully prevent users from further committing fraud, it does not act instantaneously since the ticketing server is not able to notify ticket validators that a certain ticket is being used to commit fraud, in real time.

## 4.4 Clock Synchronization

Due to the fact that this system works offline, this system is able to achieve synchronicity within its components due to the synchronization of clocks with every entity of the system, with the exception of the authentication server. This means that the ticketing server and ticket validators as well as the mobile application make use of the same time reference. The authentication server does not require the same method of synchronization of clocks, with the other components of the system, since access tokens emitted by the authentication server can be validated without having to obtain its time from the same time server as the other components. An access token contains the issuing and expiration date of the access token and in order to verify if the access token is valid, the access token uses the Unix time representation which can be obtained using the Instant class in Java.

The rest of the entities of the system have their clocks synchronized using the Network Time Protocol or NTP protocol, which is used to synchronize the times in two different computers or, in this case, between a client and a time server. A client requests the current time of the time server and calculates the difference between the clock in the time server with its own clock. This difference is called the offset and when it is added to the current time in the device of the client, the client obtains the current time, with precision, in the time server.

The implementation of the NTP protocol in the entities of the system, with the exception of the authentication server, are equal and they are used on the same time server. The time server used by this system is the Google Public NTP which is the time server provided by Google, and its time comes in UTC which is the time standard commonly used across the world. In order to calculate the offset, an entity such as the mobile application, makes use of the Apache Commons Net library. This library contains the NTPUDPClient which contains methods to calculating the offset to the time server.

# 5

## Evaluation

### Contents

---

5.1 Performance Results . . . . .	61
5.2 Battery usage . . . . .	65
5.3 Security Issues . . . . .	66

---



Following the implementation of the system, the various components that make up the system were subject to testing, namely measuring times as to obtain performance metrics over important services provided by each component such as the time to create a QR code or the time to validate QR codes. This chapter contains the metrics collected through the use of the system and contains a deliberation over the results obtained.

It is important to notice that these values were obtained in a simulation environment and not in a real public transport vehicle or station. Only by testing in a real environment, with the interference of vehicles or temporary losses of internet connection, could the results obtained have been more precise. In a scenario where there are no slowdowns in internet connection, the results obtained are approximate to the real-time operation of this system.

The simulation scenario used for the evaluation of this system was a scenario similar to the scenario found in the ticket validation use case figure 3.2. As stated above, the user of this system uses the mobile application on a physical device, namely a mobile phone. A ticket validator is located in front of the user so that the ticket validator is able to scan a QR code with the ticket, shown by the user. After reading the QR code, the ticket validator emits visual feedback based on the validation of the ticket. The back-end servers are located in the backoffice and, in order to simulate the use of back-end servers, these servers were located in a separate room, different from the one where the user was showing tickets to the ticket validator.

As previously mentioned, there are two different version of the mobile application: one for Android devices and the other version for iOS devices. For the Android version, the devices used for testing were OnePlus 7T Pro and Samsung S7 Edge whereas for the iOS version, the mobile phone device was iPhone 7 Plus. In addition, ticket validators were created on a computing platform consisting of an Intel i7-3520M (3.6GHz) with 16GB of RAM. Furthermore, the back-end servers were created on a computing platform consisting of an Intel i5-6200U (2.8GHz) with 8GB of RAM.

In order to obtain the test results, each component tested creates a record with the timestamp of the time prior to executing the service as well as the timestamp of the time after the execution. A log file is created with these times and the difference between both times which are then stored.

## **5.1 Performance Results**

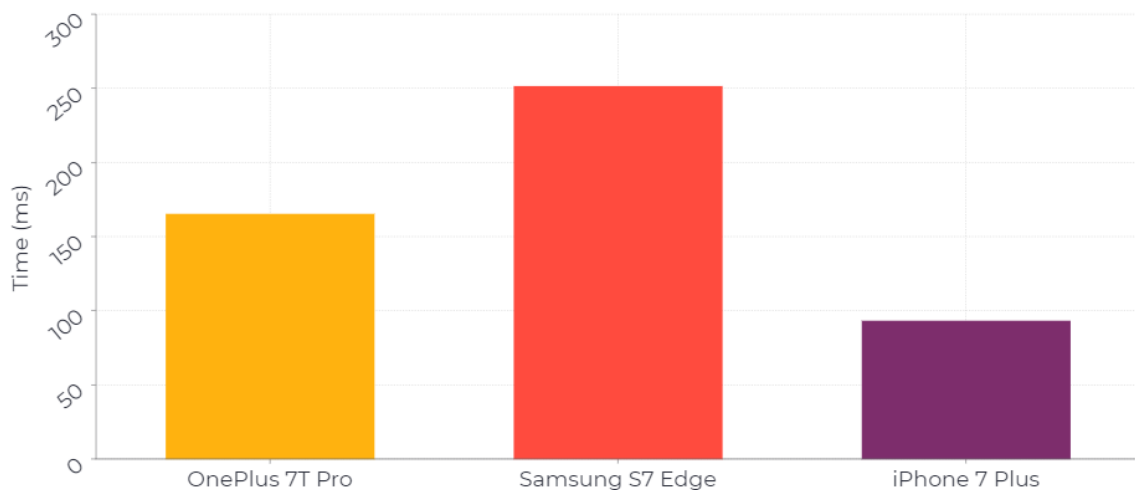
This section contains the results from the various tests made to the mobile application and ticket validator. Besides containing a detailed discussion of the obtained results, this section also provides insights on situations where these values can be different from real-world results.

### 5.1.1 Creation of QR codes

Starting off with the ticket validation use case 3.2.2, a user has already purchased a ticket and intends to display the ticket to a ticket validator. As previously mentioned, the mobile application creates a QR code based on the encryption of the ticket and OTP values, and this section presents the results of creating QR codes.

By evaluating the performance of the mobile application, in different mobile phones, it is possible to gain insights over time amount of time a mobile phone device takes to cipher the message and evaluate whether this time is appropriate for the dynamic generation of QR codes.

Since older mobile phones may lack the processing capabilities of newer mobile phones, this document presents the obtained results across multiple mobile phones with different Operating Systems (OSs), and even with different OS versions, so that it would be possible to perform an evaluation of the difference in performance and study the impact of using mobile phones, with different capabilities, on this system. These results are also taken into account to ensure that users are less likely to be limited to using newer devices, when using this system.



**Figure 5.1:** Performance results from the creation of QR codes in different devices bar chart

The figure above 5.1 contains the results of the average time to perform the creation of QR codes, over 150 runs. As stated above, in order to obtain these results, the mobile application obtains the timestamp before creating a QR code and calculates the difference to the timestamp obtained when the QR code is set to display on screen. Besides that, the mobile application calculates the difference of these times and stores it in a file.

In a real environment, the results obtained would be similar to the results in the figure, since the mobile application was installed in everyday devices and thus, they are not specific to this test case.

From the results obtained, it is possible to infer that the time taken in the creation of QR codes is

acceptable and also suitable for the dynamic approach to QR codes, since users are not required to wait for the process of generating QR codes as it occurs almost instantly after selecting the desired ticket to present as a QR code.

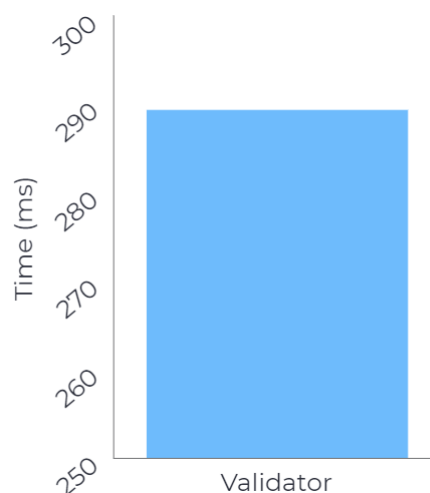
Additionally, older devices such as Samsung S7 Edge, do not fall behind in comparison with newer models such as the OnePlus 7T Pro. In iOS devices, the evaluation was only possible to be done using one mobile phone and thus, it was not possible to create a more detailed comparison between newer and older models or even across different operating systems.

Despite this, the results show that there is a noticeable difference in the times of the creation of QR codes in iOS devices, in comparison with Android devices. One reason for this time disparity between iOS and Android devices, could be the fact that iOS has a native class for the generation of QR codes whereas the Android application required the use of an external library.

### 5.1.2 Validation of QR codes

Following the creation of QR codes in the mobile phone application, a ticket validator must ensure the correct validation of tickets, through the scan of QR codes. This section presents the evaluation over the validation of QR codes and the comparison between the usage of two different hybrid encryption mechanisms, and how these results impact validation machines.

The process of validation of QR codes starts with the decryption of the message in the QR code and, since this could be the most intensive step in the validation of QR codes, this test case is also focused on various methods of decryption of the message. Furthermore, the focus on the validation of tickets is done to ensure that ticket validators do not take up much time validating tickets and execute the validation of tickets within an acceptable time limit, as stated in the research objectives section 1.2.



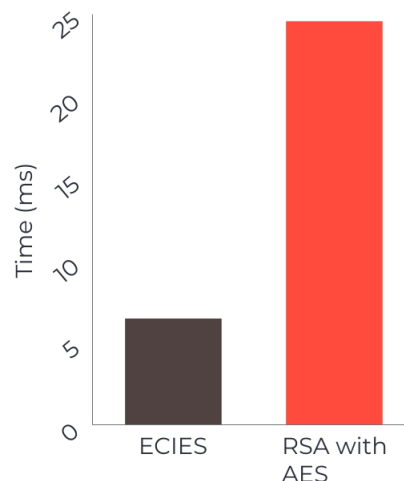
**Figure 5.2:** Performance results from the validation of QR codes bar chart

The figure 5.2 contains the average time that the ticket validator took, when executing 175 runs in a computing platform, to validate each QR code that was presented to the reader. Since it was not possible to test and obtain performance results from a ticket validator machine that is used in a real environment, these results do not represent the results that would be obtained in a real environment, since ticket validator machines have much smaller processing capabilities, in comparison with the computer used to obtain these results.

By taking a look at the results in the figure, the ticket validator performs the validation of tickets with an average time of 289.4ms. Despite the fact that ticket validator machines in public transports have smaller processing capabilities than the machine used to obtain these results, the implementation of this system with the former ticket validators should not be too time consuming and cause users to find the process of validating tickets, inconvenient.

### 5.1.2.A Comparison between ECIES with RSA-AES

In addition, it was tested the use of alternative hybrid encryption methods such as RSA-AES and these results were compared with the results obtained from using ECIES. In the figure 5.3, it is presented a graph with the average time across 75 runs, of the process of decryption of the message in the QR code. This is solely for the process of decryption of the message, and these times do not include the time from other methods for validating the ticket as well as emitting visual feedback to the user.



**Figure 5.3:** Performance results from the comparison between ECIES and RSA-AES bar chart

From the figure, the implementation of ECIES finishes the process of decryption much quicker, with an average time of deciphering of less than 7ms whereas the implementation of RSA with AES has an average time of over 24ms. Despite the fact that using RSA with AES is not as fast as using ECIES, these



results indicate that either approach would be suitable for the current security levels, since the validation times are not very significant when used in the current prototype of the ticket validator. However, in a real environment, due to the fact that ticket validator machines do not have the same processing capabilities as the computing machine used to represent the ticket validator, the impact of using RSA with AES instead of ECIES could become more noticeable and therefore, the use of ECIES would be more suitable for the validation of tickets.

The process of hybrid encryption using ECIES consists of using ECC with symmetric encryption, namely EC keys generated from the curve NIST P-256 combined with AES-128 symmetric keys. In order to be able to perform a fair comparison, the implementation of RSA with AES provides similar security levels to the ones found in ECIES. When using a combination of RSA with AES, the approach used RSA 3072-bit keys with AES-128 since RSA 3072-bit keys have the same level of security as ECC P-256 keys.

In the event that this system required higher security levels such as using EC keys from NIST P-512 for 256-bits of security, instead of having 128-bits of security from using ECIES, with EC keys from NIST P-256, the average time of using the implementation with RSA with AES could rise exponentially and cause a major impact in the overall performance of ticket validators [48].

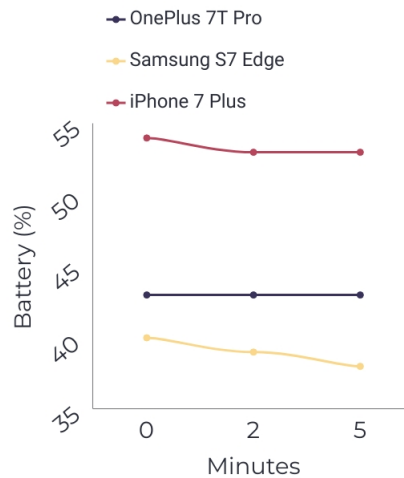
## 5.2 Battery usage

Nowadays, it is important that mobile phones are able to remain usable for extensive periods of time, namely for the duration of a whole day. This section details the battery consumption over time of the mobile phones and an explanation of how these results could diverge from usages in a real environment.

Since the generation of QR codes in the mobile application is dynamic, the mobile application creates and displays a new QR code every 60 seconds and as such, it is considered to be the most expensive action in regards to battery usage. For the current use case of the application, users would use the mobile application for a few minutes and close the application as the main purpose of the mobile application is allow users to present a ticket from the mobile application to the ticket validator.

The figure 5.4 contains a graph with the variation of battery over time, in different mobile phone devices, that were tested using the mobile application, performing the dynamic generation of QR codes, in order to put stress over the mobile phone and maximize its battery consumption. These results were taken using a refresh rate of 60Hz and without the use of the power saving mode across all devices.

From the results in the figure, presenting the QR code over a short period of time such as 2 minutes, may not even affect the battery percentage as seen in the OnePlus 7T Pro. Despite this, the mobile application does not lead to draining huge percentages of battery resources, with an average of less than 1%. Therefore, these values are acceptable for real environment usage since these values do not



**Figure 5.4:** Results from battery usage testing results line chart

cause almost any impact to the battery percentage of the mobile phone.

An exception to the rule is the Samsung S7 Edge which has a battery with years of extensive use and charges, and since the health of its battery is really low, the results on the Samsung S7 Edge may diverge from other devices of the same model.

Alternatively, it would be possible to test the battery consumption of the mobile application in case it was left open, for instance, on the home page, for a long period of time such as 1-2 hours. It would provide insights on the different processes that run in the background of the mobile application impact the battery performance of the mobile phone.

### 5.3 Security Issues

Besides obtaining performance metrics of the usability of the components of this system, specifically the mobile application and ticket validator, it is necessary to take into consideration that this system can be susceptible to attacks.

As previously mentioned, this system could be implemented in a public transports system and thus, some users with malicious intents could attempt to circumvent the security measures imposed for the correct functioning of this system, in order to avoid paying for tickets and commit fraud. In addition, other types of fraudulent behaviour may include the attempt of taking advantage of inexperienced users, with techniques such as social engineering or phishing.

This section details the possible attacks and methods which attackers can attempt to exploit the vulnerabilities of this system, and countermeasures implemented to protect this system against those attacks and prevent fraud. Taking this into account, the following attacks are the attacks that are most

likely to be used against this system:

- Reverse Engineering
- Cloning
- Double Spending
- Phishing

### **5.3.1 Reverse Engineering**

A reverse engineering attack consists of opening up an object to study its mechanisms. This system contains two methods to prevent reverse engineering attacks: code obfuscation of the application code and the encryption of the contents of QR codes.

#### **5.3.1.A Code obfuscation**

In this system, attackers may attempt to reverse engineer the application code with a view to extracting valuable information on how the application works, and to do so, attackers have at their disposal several mechanisms such as IDA Pro, which allow to convert the binary application code, to source code. The intention behind dissecting this application could be to gain insights over the creation of QR codes such as which algorithms are being used or even, the methods of storing tickets, to facilitate access to tickets.

In order to protect against these types of reverse engineering attacks, the obfuscation of the application code reduces the possibility of execution of this attack since its purpose is to difficult the process of extraction of valuable information from the mobile application.

The Android version of the mobile application uses ProGuard which is a Java open-source cross-platform tool whereas the iOS version of the mobile application contains a simplistic approach to code obfuscation where sensitive data is stored encrypted, beforehand, and decrypted at the moment of use. This approach is not fully secure and only protects sensitive fields while ProGuard can also provide obfuscation to class names and methods. A possible implementation of code obfuscation in the iOS version of the mobile application, which would suit our needs better, is using iXGuard.

#### **5.3.1.B Encrypted QR codes**

Alternatively, this system can be subject to reverse engineering attacks to the QR code. Since tickets are displayed in the form of QR codes, attackers may attempt to gain insights over the creation of tickets and find vulnerabilities in the methods of creating tickets. Besides that, attackers may also attempt to find faulty methods which the mobile application uses to display QR codes such as in the creation of the one-time password.

Hence, in order to prevent this specific case, the mobile application implements the encryption of the contents of QR codes so that only authorized entities, namely ticket validators, can be able to read the QR code. In addition, the rotation of the encryption key is done on a regular basis, to prevent brute-force attacks against the key and consequently, make reverse engineering attacks more difficult to perform.

### **5.3.2 Cloning**

Cloning attacks are attacks where an attacker steals information from another user, with the intention of using this information for their benefit and, in the case of this system, cloning attacks could be performed by copying or taking a photo of the QR code from another user. The window of attack is very short-lived since this attack could only be executed within the timestep or period of validity of the TOTP, which, as aforementioned, usually lasts less than one minute.

Despite this, cloning attacks can not be fully prevented with the use of one-time passwords due to the fact that attackers could still make multiple usages of the stolen QR code, as long as it had not yet expired. In order to prevent this, ticket validators are required to have a validation history so that the ticket validator can detect and reject QR codes that had been previously used. Time-based OTPs remain unchanged for the duration of the timestep and therefore, ticket validators are able to detect different uses of the same QR code.

As previously mentioned, ticket validators could be connected through a physical centralized server and therefore, these ticket validators would have access to validations which were performed by other ticket validators. With this, even though ticket validators operate in a near offline state, they would be able to reject tickets which had been previously accepted, in a different ticket validator.

### **5.3.3 Double Spending**

A double spending attack is an attack which can be executed by reproducing the ticket and having two users, using the same ticket. To execute this attack, an attacker is required to obtain a ticket from another user, and both users could be cooperating to commit fraud.

There are multiple ways of executing double spending attacks such as mobile phone sharing, the sharing of QR codes over the internet and the extraction of tickets from the mobile application. This section also covers the reasoning behind the methods used to prevent these attacks.

#### **5.3.3.A Mobile phone sharing**

To start off, one user could share their mobile phone with another user, after validating their QR code, and the other user can use the same ticket, as long as the mobile application has created a new QR code. This is due to the fact that ticket validators create a record of the ticket information and OTP

value. Since this system makes use of time-based tickets, the owner of the ticket should be able to use the ticket more than once. Hence, ticket validators do not prohibit multiple uses of the same ticket but instead, they prohibit multiple uses of the same QR code.

For simplicity reasons, the current prototype of this system assumes that the sharing of the mobile phone would not be possible and therefore, the current implementation of this system does not prohibit against this specific scenario. A possible implementation of this feature could make use of an anti-passback control system, which would prevent subsequent usages of the same ticket, and it is further developed in the Future Work section (6.1).

In addition, the ticketing server contains an anti-fraud system which detects fraudulent behaviour by analysing the ticket transactions sent from the ticket validators. Users that bypass the proposed anti-passback control mechanism, are going to be detected by the anti-fraud system which takes preventive action.

#### **5.3.3.B Sharing QR codes over the internet**

Another case of double spending is the sharing of QR codes over the internet. For instance, one user could take a screenshot of a QR code, created in their mobile application, and send it to another user so that they would use the same QR code in a different ticket validator. This scenario can only be possible due to the fact that ticket validators operate in a near offline state and do not receive information from other entities such as other ticket validators or a backend server, to inform that a specific ticket has been used.

In order to prevent users from sharing sensitive information, the mobile application implements security methods such as screenshot blocking. Despite the fact that users can still take a photo of the QR code of another user, by blocking the possibility of taking a screenshot of the QR code, users are less likely to attempt the execution of this attack.

The Android version of the mobile application contains the implementation of this method since it is supported natively by Android. However, as aforementioned, the iOS version of the mobile application does not implement methods to block screenshots since these are not supported natively by iOS.

Hence, the current prototype of the iOS version of the mobile application contains a small window of attack which is equivalent to the period of validity of the OTP in the QR code. In addition, tickets contain specific parameters that can be used to limit the target audience of the ticket such as valid zones and valid operators. Tickets that are not presented to ticket validators that meet these requirements, are going to be rejected. With this, in the current prototype of iOS version of the mobile application, the combination of these factors does not fully prevent this specific case of double spending attacks. However, it makes the attack more difficult and reduces the possibility of successfully executing this attack.

Another possible implementation of screenshot blocking in the iOS version of the mobile application is through the use of ScreenShieldKit which blocks sensitive content in screenshots, as mentioned in the Future Work section 6.1.

Similarly to mobile phone sharing, users that bypass this security mechanism are going to be detected by the anti-fraud system which takes preventive action against fraudulent behaviour.

### **5.3.3.C Ticket extraction**

Users may attempt to forcefully move tickets from one application to another, in a different mobile phone. In case tickets were stored as plaintext, other applications could read the contents of tickets and generate new QR codes. Hence, the mobile application performs the storage of the encrypted contents of tickets, in the internal storage, of a mobile phone and therefore, these tickets can only be decrypted using the keystore from that specific device. Other devices are not able to read the contents of the ticket and thus, they cannot create new QR codes.

Besides that, in order to move tickets from one application to another, the attacker must bypass the security mechanisms imposed by the Android OS, in order to execute methods that require special access or root. For this case, the mobile application implements root checking and disables the ability of using the application in rooted devices or, in the case of iOS devices, jailbroken devices.

### **5.3.4 Phishing**

Phishing attacks are attacks that can be performed by deceiving users to access websites or install applications which are controlled by the attacker. The intent behind phishing is to gain personal information of the users of this system so that attackers can exploit this system and potentially commit fraud through the use of personal information from other users. For instance, in the event that users have installed a malicious fake application, there is high likelihood that their account credentials, tokens and tickets, have been stolen and will be used by attackers. Thus, the application needs to provide a robust security mechanism that protects users against phishing.

Ticket validators send ticket validations to the ticketing server and, one of its purposes is to allow for the analysis of ticket validations and detect fraudulent behaviour. In the event that users share their tickets with other users, ticket validations would allow for the detection of information sharing. Through analysis of ticket validations, it would be possible to identify instances where multiple users are making use of the same ticket. In these situations, system administrators would be responsible for taking preventive action.

Furthermore, there is the possibility of implementing multi-factor authentication in the mobile application. With multi-factor authentication, users would be required to authenticate through another method

besides their account credentials such as using one-time passwords. In this case, users would be required to confirm the one-time password sent by the authentication server to the mobile phone of the owner of the account.

Another possible implementation to prevent against phishing attacks would be that the authentication server could disable the ability of having more than one active session per account. Each account and session would be associated with a specific mobile phone device, for instance, through the recording of the International Mobile Equipment Identity (IMEI). In this scenario, users that had an active account session in their mobile phone device would be protected against this case of phishing attacks since the authentication server would detect and reject logins made in different mobile phones.





# 6

## Conclusion

### Contents

---

6.1 System Limitations and Future Work . . . . .	76
--	----

---



With the recent advances in mobile ticketing, several research groups have been trying to implement a ticketing system which makes use of barcodes such as QR codes, some with relevant results. However, after performing the analysis of these systems, we came to the conclusion that these implementations still did not meet all the required criteria and that there is room for improvement. For instance, one approach contained the implementation of a ticketing system that provided the offline validation of tickets through the use of digital signatures. The approach to the system would be able to correctly validate tickets with a single validator but, in case of a distributed system that involves the offline validation of tickets over multiple ticket validators, this system would not provide the reliable validation of tickets and prevent fraud.

Taking this into account, the proposed solution consists of a distributed system, in order to achieve scalability, and it contains the development of a mobile application, a ticket validator and back-end servers. One of the main innovations in our approach is that the mobile application is able to generate dynamic QR codes. As previously mentioned, through the generation of dynamic QR codes and taking advantage of one-time passwords, the mobile application is able to ensure the security and freshness of tickets. However, it is required to take into account that mobile devices face other issues such as battery consumption and that ticket validator machines in public transports have limited processing capabilities in comparison with computing platforms. Besides that, in order to maximize the number of users that are able to use this system, other factors were taken into account such as interoperability and thus, there was the development of two version of the mobile application, one for Android devices and the other for iOS devices.

The evaluation of the performance of the system consists of the evaluation of the mobile application and ticket validators such as the creation of tickets and the validation of tickets, respectively. In addition, there were specific requirements which were imposed as research objectives such as the time limit for the validation of tickets in order to ensure that results would be acceptable in a real environment. The results obtained were mostly satisfactory and the minority of outliers, i.e. values that deviated most from the average time, did not compromise the correct behaviour of the system.

In conclusion, the proposed solution focuses on a ticketing system which provides the accurate validation of tickets by ticket validators and allows users to perform the generation of QR codes, within their mobile phone application. By providing an integration of mobile tickets with one time passwords, the proposed solution implements a system based on the dynamic generation of QR codes. Even though this system aims to focus on the general concept of mobile ticketing, it can be adapted to a specific field such as the public transport industry. Some of the characteristics of system are innovative and involve the implementation of QR codes with one-time passwords in order to provide tickets which can be used in a mobile ticketing system. Our solution meets the requirements and research objectives which were initially proposed, with an innovative approach and margin for improvements.

## 6.1 System Limitations and Future Work

Following the development of the system, there are some additional features that could be added to later iterations of this system and some features in the current version that, for simplicity reasons, can be even further developed.

This section consists of the explanation of these features and how they could improve the system. These are some of the main ideas that can be explored in the future:

- **Implementation of the anti-passback control system** As stated above, ticket validators record the information of tickets and the OTP value, in a validation history, so that validators can use these values to reject uses of the same ticket by unauthorized parties. This issue is specified in the Double Spending section (5.3.3) and the intention behind using a validation timeout would account for the prevention of users sharing their mobile phone device, with another user, after one user has used the ticket.

As an example, a possible implementation of this feature would have ticket validators reject validations for the same ticket, for a period of 10-15 minutes, in case the validation of the ticket occurred within the timeout period. By implementing a timeout period, users are less likely to take advantage of this exploit.

- **Implementation of ticket recovery feature** In the current prototype of the mobile application, upon signing in using their account credentials in a different mobile phone, users would not have any tickets. This is due to the fact that the process of recovering tickets, in a different mobile phone, was not implemented.

This process would consist of making a recover request to the ticketing server so that it would forward the tickets which the user purchased, to the mobile application. Since this process would allow users to store their tickets in multiple devices, it is necessary to take into account and prevent multiple users from making use of the same tickets, as this would be considered a double spending attack.

One possible implementation to prevent this is that the ticketing server would prevent users from recovering tickets within a specific time frame such as the expiration time of the signature. In this scenario, the ticketing server would not allow for the recovery of tickets whose signature had not yet expire. Upon the expiration of the signature, the ticketing server is able to send these tickets to the mobile application.

Alternatively, the aforementioned time frame could be extended over a longer period such as the period of several hours or days and in case this period had not yet finished, users would need to contact customer support to recover their tickets.

- **Integration of a payment gateway** The current prototype of the mobile application does not integrate a payment gateway for the purchase of tickets. The implementation of this feature would include a payment request to a financial company which would in turn, notify the ticketing server of the purchase of a certain ticket and, in case the payment of the ticket was successful, the ticketing server would forward the ticket to the user.
- **Improvement of code obfuscation in iOS devices** Code obfuscation in iOS devices contain a simplistic method of obfuscating sensitive information. This method, as aforementioned, works by encrypting sensitive fields in the mobile application beforehand, and the application decrypts these fields at the moment of using them. This method is insecure since the mobile application does not perform the rotation of the encryption key. In addition, there is the lack of use of salt when generating this key and thus, the current prototype of the mobile application in iOS, makes use of same key for every sensitive field.

A future version of the mobile application should make use of a code obfuscation library, which would be more secure and would better meet our needs, such as iXGuard. iXGuard works similarly to ProGuard, an open-source code obfuscation library for Java, by encrypting the sensitive fields of the application as well as class names and methods.

- **Implementation of protection against screenshots in iOS devices** Users that take screenshots of tickets as QR codes, are able to share the ticket with other users, through the internet. In this system, the current prototype of the iOS version of the mobile application does not contain protection against the screenshotting of tickets, as previously mentioned.

Despite the fact that the one-time password creates a very limited window of attack, ticket validators which are located in different locations, can accept QR codes shared over the internet, as long as the OTP in the QR code has not expired and the parameters in the ticket are valid for the specific validator. With this, a future version of the iOS version of the mobile application should contain methods for preventing users from taking screenshots or blocking the contents of the screenshot taken such as using ScreenShieldKit.

- **Implementation of the anti-fraud system** As aforementioned, the anti-fraud system located in the ticketing server was not implemented since it was out of the scope of this project. However, a possible implementation would require the training of a system which would be able to detect fraudulent behaviour in instances where it could be detected that multiple users were making use of a ticket.

A future iteration of this system should have this anti-fraud system implemented as the anti-fraud system allows for the detection of fraud and contributes to prevent monetary loss for the company responsible for this system.

- **Implementation in a real environment** Following the development of a prototype of this system, a future implementation of this system could be applied in a real environment such as the public transports industry. Therefore, this system could be proposed for implementation to a public transports company such as Transportes Metropolitanos de Lisboa.

Nowadays, recent versions of ticket validators used in public transports, also support the scanning of QR codes and therefore, the implementation of this system using these types of ticket validators would be possible. In addition, the support of QR code scanning in newer ticket validators leads to the tendency of using QR codes in the future and, as a result, these validators are not at risk of becoming outdated in the near future. However, older versions of ticket validators that do not support the scanning of QR codes, are not able to be used in the implementation of this system since this system requires the use of QR code technology to display tickets.

# Bibliography

- [1] M. Mezghani, "Study on electronic ticketing in public transport. Final Report, European Metropolitan Transport Authorities. May 2008, <http://www.emta.com/IMG/pdf/EMTA-Ticketing.pdf> (date of access 25.04.2015)." [Online]. Available: <http://www.emta.com/IMG/pdf/EMTA-Ticketing.pdf>
- [2] H. L. H. S. Warnars, Y. Lanita, A. Prasetyo, and R. Randriatoamanana, "Smart integrated payment system for public transportation in jakarta," *Bulletin of Electrical Engineering and Informatics*, vol. 6, no. 3, pp. 241–249, 9 2017.
- [3] I. Yatskiv, M. Savrasovs, D. Udre, and R. Ruggeri, "Review of intelligent transport solutions in Latvia," *Transportation Research Procedia*, vol. 24, pp. 33–40, 12 2017.
- [4] B. Guirao, A. García, M. Lopez-Lambas, C. Acha, and J. Comendador, "New qr survey methodologies to analyze user perception of service quality in public transport: The experience of madrid," *Journal of Public Transportation*, vol. 18, pp. 71–88, 09 2015.
- [5] J. Neefs, F. Schrooyen, J. Doggen, and K. Renckens, "Paper Ticketing vs. Electronic Ticketing Based on Off-Line System 'Tapango'," 05 2010, pp. 3 – 8.
- [6] M. Makita, "NFC-Based Mobile Ticketing. July 2017, <https://nfc-forum.org/nfc-based-mobile-ticketing/>." [Online]. Available: <https://nfc-forum.org/nfc-based-mobile-ticketing/>
- [7] R. Couto, J. Leal, P. M. Costa, and T. Galvão, "Exploring ticketing approaches using mobile technologies: Qr codes, nfc and ble," in *2015 IEEE 18th International Conference on Intelligent Transportation Systems*, 2015, pp. 7–12.
- [8] D. Škarica, H. Belani, and S. Illes, "Implementation and evaluation of mobile ticket validation systems for value-added services," 10 2009, pp. 260 – 264.
- [9] "Information technology — Automatic identification and data capture techniques — QR Code bar code symbology specification," Standard, Feb. 2015.
- [10] S. Dey, "SD-EQR: A New Technique To Use QR Codes™ in Cryptography," 05 2012.

- [11] R. Aulya, H. Hindersah, A. Prihatmanto, and K. H. Rhee, "An authenticated passengers based on dynamic QR Code for Bandung Smart Transportation Systems," 08 2016, pp. 23–27.
- [12] C. U. Reddy, D. V. P. Reddy, N. Srinivasan, and J. A. Mayan, "Bus ticket system for public transport using QR code," *IOP Conference Series: Materials Science and Engineering*, vol. 590, p. 012036, oct 2019. [Online]. Available: <https://doi.org/10.1088/1757-899x/590/1/012036>
- [13] L. Finzgar and M. Trebar, "Use of NFC and QR code identification in an electronic ticket system for public transport," *SoftCOM 2011, 19th International Conference on Software, Telecommunications and Computer Networks*, pp. 1–6, 2011.
- [14] P. Reddy, "Real Life Applications of CRYPTOGRAPHY. Nov 2019, <https://medium.com/@prashanthreddyt1234/real-life-applications-of-cryptography-162ddf2e917d>." [Online]. Available: <https://medium.com/@prashanthreddyt1234/real-life-applications-of-cryptography-162ddf2e917d>
- [15] S. Duggal, V. Mohindru, P. Vadiya, and S. Sharma, "A Comparative Analysis of Private Key Cryptography Algorithms: DES, AES and Triple DES," *International Journal of Advanced Research in Computer Science and Software Engineering*, vol. 6, p. 1373, 06 2014.
- [16] D. Singh and A. Singh, "A Secure Private Key Encryption Technique For Data Security in Modern Cryptosystem," *BVICAM's International Journal of Information Technology*, vol. 2, 01 2010.
- [17] D. J. Bernstein, J. Breitner, D. Genkin, L. Groot Bruinderink, N. Heninger, T. Lange, C. van Vredendaal, and Y. Yarom, "Sliding right into disaster: Left-to-right sliding windows leak," in *Cryptographic Hardware and Embedded Systems – CHES 2017*, W. Fischer and N. Homma, Eds. Cham: Springer International Publishing, 2017, pp. 555–576.
- [18] D. Pocock, "RSA Key Sizes: 2048 or 4096 bits? - <https://danielpocock.com/rsa-key-sizes-2048-or-4096-bits/>." [Online]. Available: <https://danielpocock.com/rsa-key-sizes-2048-or-4096-bits/>
- [19] V. Mainanwal, M. Gupta, and S. K. Upadhayay, "Zero knowledge protocol with rsa cryptography algorithm for authentication in web browser login system (z-rsa)," in *2015 Fifth International Conference on Communication Systems and Network Technologies*, 2015, pp. 776–780.
- [20] V. Gayoso Martínez, L. Hernandez Encinas, and A. Queiruga-Dios, "Security and practical considerations when implementing the elliptic curve integrated encryption scheme," *Cryptologia*, vol. 39, pp. 1–26, 05 2015.
- [21] A. A. P. Ratna, P. Purnamasari, A. Shaugi, and M. Salman, "Analysis and comparison of md5 and sha-1 algorithm implementation in simple-o authentication based security system," 06 2013, pp. 99–104.



- [22] L. Latinov, "MD5, SHA-1, SHA-256 and SHA-512 speed performance, <https://automationrhapsody.com/md5-sha-1-sha-256-sha-512-speed-performance/>," July 2018. [Online]. Available: <https://automationrhapsody.com/md5-sha-1-sha-256-sha-512-speed-performance/>
- [23] N. I. of Standards and Technology, "The Keyed-Hash Message Authentication Code (HMAC)," *Soft-COM 2011, 19th International Conference on Software, Telecommunications and Computer Networks*, pp. 1–13, 07 2008.
- [24] K. R. Glenn and C. R. Madson, "The Use of HMAC-SHA-1-96 within ESP and AH," RFC 2404, Nov. 1998. [Online]. Available: <https://rfc-editor.org/rfc/rfc2404.txt>
- [25] M. View, D. M'Raihi, F. Hoornaert, D. Naccache, M. Bellare, and O. Ranen, "HOTP: An HMAC-Based One-Time Password Algorithm," RFC 4226, Dec. 2005. [Online]. Available: <https://rfc-editor.org/rfc/rfc4226.txt>
- [26] L. Lamport, "Password authentication with insecure communication," *Commun. ACM*, vol. 24, no. 11, p. 770–772, Nov. 1981. [Online]. Available: <https://doi.org/10.1145/358790.358797>
- [27] R. SecurID, "Two Factor Authentication, Security Token - EMC." [Online]. Available: <http://www.rsa.com/node.aspx?id=1156>
- [28] M. Karuppiah, G. D. H. Mehta, A. Rajan, and B. Perumal, "A novel way of integrating voice recognition and one time passwords to prevent password phishing attacks," *International Journal of Distributed and Parallel systems*, vol. 5, pp. 11–20, 07 2014.
- [29] Y. Huang, Z. Huang, H. Zhao, and X. Lai, "A new one-time password method," *IERI Procedia*, vol. 4, pp. 32 – 37, 2013, 2013 International Conference on Electronic Engineering and Computer Science (EECS 2013). [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S2212667813000099>
- [30] U. Abdurrahman, M. Kaiiali, and J. Muhammad, "Enhancing totp protocol by embedding current gps location," 10 2014.
- [31] M. View, J. Rydell, M. Pei, and S. Machani, "TOTP: Time-Based One-Time Password Algorithm," RFC 6238, May 2011. [Online]. Available: <https://rfc-editor.org/rfc/rfc6238.txt>
- [32] C.-Y. Huang, S.-P. Ma, and K.-T. Chen, "Using one-time passwords to prevent password phishing attacks," *J. Network and Computer Applications*, vol. 34, pp. 1292–1301, 07 2011.
- [33] K. Liao, W. Lee, M. Sung, and T. Lin, "A one-time password scheme with qr-code based on mobile phone," in *2009 Fifth International Joint Conference on INC, IMS and IDC*, 2009, pp. 2069–2071.

- [34] T. Hwang, Y.-P. Luo, and Z.-R. Liu, "Forward/Backward Unforgeable Digital Signature Scheme Using Symmetric-Key Crypto-System," 12 2016, pp. 244–247.
- [35] S. Jarusombat and S. Kittitornkun, "Digital Signature on Mobile Devices based on Location," 10 2006, pp. 866 – 870.
- [36] C. Tian-huang and X. Xiao-guang, "Digital signature in the application of e-commerce security," *2010 International Conference on E-Health Networking Digital Ecosystems and Technologies (EDT)*, vol. 1, pp. 366–369, 2010.
- [37] S. Alam, A. Jamil, A. Saldhi, and M. Ahmad, "Digital image authentication and encryption using digital signature," in *2015 International Conference on Advances in Computer Engineering and Applications*, 2015, pp. 332–336.
- [38] O. Foundation, "OpenID Connect: <https://openid.net/connect/>." [Online]. Available: <https://openid.net/connect/>
- [39] "Access Tokens: <https://www.oauth.com/oauth2-servers/access-tokens/>." [Online]. Available: <https://www.oauth.com/oauth2-servers/access-tokens/>
- [40] K. Dodanduwa and I. Kaluthanthri, "Role of Trust in OAuth 2.0 and OpenID Connect," in *2018 IEEE International Conference on Information and Automation for Sustainability (ICIAfS)*, 2018, pp. 1–4.
- [41] Y. KAKIZAKI and H. TSUJI, "A decentralized attribute management method and its implementation," *International Journal of Information Processing and Management*, vol. 3, 01 2012.
- [42] "Network Time Protocol." [Online]. Available: [https://en.wikipedia.org/wiki/Network\\_Time\\_Protocol](https://en.wikipedia.org/wiki/Network_Time_Protocol)
- [43] StatCounter, "Mobile Android operating system market share by version worldwide from January 2018 to June 2021," June 2021. [Online]. Available: <https://www.statista.com/statistics/921152/mobile-android-version-share-worldwide/>
- [44] AppBrain, "Android OS version market share over time," August 2021. [Online]. Available: <https://www.appbrain.com/stats/top-android-sdk-versions>
- [45] S. Alexander-Bown, "RootBeer library." [Online]. Available: <https://github.com/scottyab/rootbeer>
- [46] "Keycloak." [Online]. Available: <https://www.keycloak.org/>
- [47] "ECDSA." [Online]. Available: [https://en.wikipedia.org/wiki/Elliptic\\_Curve\\_Digital\\_Signature\\_Algorithm](https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm)

- [48] V. Gayoso Martínez, L. Hernandez Encinas, and A. Queiruga-Dios, “Security and practical considerations when implementing the elliptic curve integrated encryption scheme,” *Cryptologia*, vol. 39, pp. 1–26, 05 2015.



