Beyond the password

Ricardo Jorge Mendes de Almeida

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisors: Prof. José Manuel da Costa Alves Marques

Examination Committee

Chairperson: Prof. Daniel Jorge Viegas Gonçalves
Supervisor: Prof. José Manuel da Costa Alves Marques
Members of the Committee: Prof. José Manuel da Costa Alves Marques
Prof. Nuno Miguel Carvalho dos Santos

June 2018
Abstract

Nowadays, most of our devices are used to access business applications available on the web, such as online banking, e-commerce, etc. The common factor between all business applications is the need to use a secure authentication mechanism to perform some kind of operations, such as a login or a transaction. The most used authentication mechanism nowadays are passwords. However, passwords have problems of security and usability that can be solved by using biometrics as the authentication mechanism. The purpose of this work is to implement a system, called BioAuth, to perform second-level authentication requests using biometrics. Biometric authentication is performed on mobile devices and the information is processed by a central server. In this report, it is first carried out an investigation of already existing works that use biometrics and then an analysis of these same works, where it is concluded that the biometrics most suitable to implement in mobile devices are fingerprints, facial recognition and speech recognition. Finally, is explained the system architecture and implementation, how the system tests were performed and what can be added to the system in the future.

Keywords

Biometrics; Authentication; FIDO UAF; Security; Cryptography; Xamarin;
Resumo

Nos dias que correm, a maioria dos nossos dispositivos são utilizados para aceder a serviços de negócios disponíveis na web, tais como online banking, e-commerce, etc. O que todas estas aplicações de negócios tem em comum é o facto de para todas elas ser necessário um mecanismo de autenticação para realizar certo tipo de operações, tais como, entrar na nossa conta pessoal, realizar uma transação, etc. O mecanismo de autenticação mais utilizado atualmente, é a palavra-passe. No entanto, as palavras-passe trazem problemas a nível de segurança e de usabilidade, que podem ser resolvidos ao utilizar biométricas como mecanismo de autenticação. Por isso, o objetivo deste trabalho consiste na implementação de uma sistema, a que se deu o nome de BioAuth, para realizar pedidos de segundo-nível de autenticação utilizando biométricas. A autenticação por biometria é realizada em dispositivos moveis e a informação é processada por um servidor central. Neste relatório é primeiramente realizada uma investigação de trabalhos já existentes que utilizem biométricas e depois uma análise desses mesmos trabalhos, onde se conclui que as biométricas mais aptas para implementar em dispositivos moveis são as impressões digitais, reconhecimento facial e reconhecimento de voz. Por fim é explicada a arquitetura e implementação do sistema, como foram realizados os testes ao sistema e o que se pode acrescentar ao sistema no futuro.

Palavras Chave

Biométricas; Autenticação; FIDO UAF; Segurança; Criptografia, Xamarin;
## Contents

1 Introduction .................................................. 1
   1.1 Objectives of the Work ................................. 3

2 Related Work .................................................. 5
   2.1 Biometric Systems ....................................... 6
      2.1.1 Hand region ........................................ 7
      2.1.2 Facial region ....................................... 11
      2.1.3 Ocular region ....................................... 12
      2.1.4 Behavioral biometrics .............................. 13
         2.1.4.A Keystroke dynamic ....................... 13
         2.1.4.B Voice ........................................ 14
         2.1.4.C Signature .................................... 15
         2.1.4.D Gait .......................................... 15
      2.1.5 Medico-chemical biometrics .................... 16
      2.1.6 Multimodal biometrics .......................... 16
   2.2 Framework for secure authentication .................. 17
      2.2.1 Relation with other technologies ............ 18
   2.3 Application Services ................................... 19
   2.4 Conclusion ................................................ 19

3 Architecture .................................................. 21
   3.1 Architecture Design Requirements ................... 22
      3.1.1 Functional requirements ....................... 22
      3.1.2 Non Functional Requirements .................. 22
   3.2 Architectural Modules .................................. 24
   3.3 Data Flow ................................................ 25
      3.3.1 Registration data flow .......................... 26
         3.3.1.A Pre-Registration .......................... 26
         3.3.1.B Registration ............................... 27
3.3.2 Authentication data flow ........................................ 29
3.3.3 Deregistration data flow ........................................ 31
  3.3.3.A Pre-Deregistration ........................................ 31
  3.3.3.B Deregistration ........................................ 31
3.3.4 Security Considerations ........................................ 33
  3.3.4.A User Device - Authentication System .................... 33
  3.3.4.B Authentication System - Business Web Application .... 34
  3.3.4.C Mobile Device Security ................................ 34

4 Implementation ...................................................... 35
  4.1 Development Process ........................................... 36
  4.2 Development Environment ..................................... 36
  4.3 User Device ..................................................... 37
    4.3.1 User Interface ............................................ 37
    4.3.2 FIDO Client ............................................... 40
      4.3.2.A Registration .......................................... 40
      4.3.2.B Authentication ........................................ 43
      4.3.2.C Deregistration ......................................... 44
    4.3.3 Authenticator-Specific Module (ASM) ..................... 45
      4.3.3.A Android KeyStore and Fingerprint .................... 46
        A – Security on Android .................................... 48
      4.3.3.B iOS TouchID and Keychain ............................ 49
        A – Security on iOS ......................................... 52
  4.4 Authentication System .......................................... 53
    4.4.1 Communication Service ................................... 54
    4.4.2 Integration Service ....................................... 56
    4.4.3 FIDO Server ............................................... 56
      4.4.3.A Messages verification ................................ 56
      4.4.3.B Push notification ...................................... 58
      4.4.3.C Integration with a database ........................ 59
    4.4.4 FIDO Database ............................................. 61
  4.5 Business Web Application ..................................... 62
    4.5.1 Authentication Provider Plugin .......................... 62

5 Evaluation .......................................................... 63
  5.1 Usability Tests ................................................ 64
  5.2 Interoperability Tests ......................................... 66
List of Figures

1.1 Second-Factor-Authentication using biometrics ........................................... 3
2.1 Software components of biometric systems .................................................... 6
3.1 BioAuth’s Software Architecture ......................................................................... 23
3.2 Registration message flow .................................................................................. 28
3.3 Authentication message flow ............................................................................... 30
3.4 Deregistration message flow ............................................................................... 32
4.1 BioAuth’s schedule ............................................................................................. 36
4.2 Main Screen showing an error to the user asking to set a lock screen that uses fingerprint 38
4.3 Android and iOS Registration Screens ................................................................. 39
4.4 Registration screens on Android (left) and iOS (right) ........................................ 39
4.5 Option listing (left) and pop-up message (right) ................................................. 40
4.6 Android screen asking for user’s fingerprint ...................................................... 42
4.7 Authentication successful message ..................................................................... 45
4.8 iOS flow of an application interacting with the keychain ..................................... 53
List of Tables

2.1 Biometric's accuracy .................................................. 20

5.1 Performance test table for a time delay between messages of 1000 milliseconds .... 70
5.2 Performance test table for a time delay between messages of 100 milliseconds .... 71
Acronyms

ASM
Authenticator-Specific Module

BOPS
Biometric Open Protocol Standard

DoS
Denial of Service

ECDSA
Elliptic Curve Digital Signature Algorithm

FIDO
Fast Authentication IDentity Online

GUI
Graphical User Interface

LINQ
Language-Integrated Query

OTP
One Time Password

OS
Operating System

OSs
Operating Systems

PSS
Probabilistic Signature Scheme

RNG
Random Number Generator

SDK
software Development Kit

SE
Secure Element

SEP
Secure Enclave Processor

TEE
Trusted Execution Environment

TLV
Type-Length-Value

UAF
Universal Authentication Framework

UI
User Interface
<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WCF</td>
<td>Windows Communication Foundation</td>
</tr>
<tr>
<td>WSDL</td>
<td>Web Services Description Language</td>
</tr>
</tbody>
</table>
Introduction
Nowadays with the evolution of technology, it is possible access all kind of services such as e-commerce, online banking, e-payment and many others using our personal devices, like smartphones.

Such services require user authentication to allow users to access their accounts or to perform web transactions. Almost all of these services use passwords as the authentication and authorization mechanism.

However problems for the users emerge by using passwords as a security mechanism. As an example, if the user wants a strong protection, he has to memorize big passwords, that generally should not contain dictionary words, and are a combination of numbers, special characters, lowercase and uppercase letters [1]. It becomes hard for the user to remember such passwords.

Security and memorization are not the only problems associated with big passwords. It is important to notice that nowadays people are using smartphones for everything so user-friendliness authentication mechanisms are important too [1], since mobile devices are intended for quick and frequent access. Strong passwords are not appropriate for use in mobile devices due to the length of time required for their input, which can lead to password disclosure.

These two problems need to be solved: improve (i) security and (ii) usability of authentication and authorization mechanisms.

To achieve an higher level of security there exists alternative authentication mechanisms apart from knowledge (passwords), such as possession factor and biometrics.

Authentication based on a possession factor, although it is more secure than knowledge factor, it's still not a perfect solution since the user needs to possess and carry an additional object such as a smart card or an USB Pen, in order to authenticate. And it is the opposite of usability. With biometrics the problems of remembering text (passwords) and possess some kind of object (possession factor) to authenticate disappear [2] [3]. Biometrics are described as automatic recognition of individuals through their unique physiological (fingerprint, face, iris, etc.) or behavioural (voice, gait, signature, etc) attributes.

Biometrics offer several advantages over knowledge (passwords) since they are present on users all the time, can be input as quickly as a glance or a touch and they cannot be forgotten.

However, like every security mechanism, current biometrics system can be forged. For example, a fingerprint sensor can be forged by using a gummy fingerprint or a face recognition system by presenting to the system a picture of a face [3].

It means that currently biometric systems should not completely replace passwords, but rather complement those systems.
1.1 Objectives of the Work

The main objective of the proposed work is to study the current technology regarding biometric authentication systems and implement a second-factor-authentication framework as an alternative method for business applications available on the web, such as BankOnBox and eDocLink (both systems belong to Link Consulting company). Fig. 1.1 illustrates a second factor authentication flow, in an online banking system, using biometrics to perform second factor authentication (authorize the transaction), instead of certificates or One-Time-Passwords, like most of current system do.

Along with security, the framework needs to have an high level of usability, to allow users to authenticate with the minimum effort, and wherever the user is, it has to achieve high level of performance, independent of the device and the environment.

To reach the proposed objectives, information about biometrics technology, such as advantages, disadvantages and current implemented solutions need to be studied and analysed. Protocols for authentication and authorization are identified and investigated too.

Based on the information collected, biometrics are chose to implement the framework, taking into account security, performance and usability metrics.

In order to implement the framework a software architecture needs be defined. Different types of architectures are studied and the best one for this set of objectives is chosen.

To implement the system the information about different software tools are collected and analysed,

![Second-Factor-Authentication using biometrics](image)

**Figure 1.1:** Second-Factor-Authentication using biometrics
to be used. After the system implementation the system is tested based on security, usability, interoperability and performance.
## Contents

2.1 Biometric Systems .............................................. 6  
2.2 Framework for secure authentication .......................... 17  
2.3 Application Services ........................................... 19  
2.4 Conclusion ....................................................... 19
Most of the articles chosen are recent, from the last 4 years (2013-2017). However, there are some older relevant works that need to be mentioned due to their relevance in the biometrics field.

First are presented the general concepts of biometrics and how biometrics are evaluated. After that, biometrics are explained in detail and relevant works are mentioned. At the end of this section information regarding biometrics implementation approaches are covered.

2.1 Biometric Systems

A biometric system is defined as a system that recognize patterns and matches features from acquired signals with features stored in database [2].

Although the variety of applications and technologies for biometrics systems, all biometric systems have many software components in common [3]. The general components of a biometric system (Fig.2.1) are the following:

- **Signal acquisition module**: Acquires the images or signals of a subject's biometric characteristics. Then it outputs the image/signal to the system as a sample for further processing.

- **Feature extraction module**: Extracts features from the biometric sample. This module needs to have some quality, to ensure that the biometric sample acquired can be used for further processing, otherwise the system can reject the sample and collect other samples.

- **Data storage module**: Samples acquired at the enrolment stage are stored here. Those samples are often termed as templates. Those templates can be stored within a mobile device, a portable medium like a smart card or within a central server or personal computer.

- **Matching module**: Features are compared with those of data storage and from each comparison a similarity score is outputted.
• **Decision module**: The generated score from the matching module is used to provide a decision outcome for a verification or identification.

A biometric system operates in one of the following modes:

1. **Enrolment**: a sample is acquired and based on the quality of the sample, it is stored.
2. **Verification**: the user claims a particular identity and the system verifies a specific claim about the user enrolment. It is a one-to-one comparison. Verification will either accept or reject.
3. **Identification**: The system tries to find an identifier based on the submitted biometric sample, comparing the submitted sample to all the templates stored in the database. The difference between this mode and verification mode is that while the comparison in verification is one-to-one, in this mode is one-to-many.

ISO 19795 [4] standard provides definitions and explanations on general biometrics systems and those norms are used in almost all biometric systems [3]. The ISO 19795 norm proposes rates to evaluate biometric algorithms being the following the most commonly used in literature:

• **False rejection rate (FRR)**: The percentage measure of invalid matches, where the system recognizes an authorized user as an impostor.

• **False acceptance rate (FAR)**: The percentage measure of invalid matches, where the system recognizes unauthorized users as genuine users.

• **Receiver operating characteristics (ROC) curve**: Plot of the rate of false positives (x-axis) against the rate of true positives (y-axis).

• **Detection error trade-off (DET) curve**: A modified ROC that represents FAR on the x-axis and FRR on the y-axis.

• **Equal error rate (ERR)**: The point where FAR and FRR have the same value on DET curve. ERR is the measure of the accuracy of a biometric system. The accuracy of a system increases by decreasing FAR and FRR.

However, some researchers use other rates to evaluate their systems.

Different biometric modalities will first be explained and then some relevant works will be presented. Deeply explained characteristics of biometric systems and different biometric modalities can be found in many articles, books and state of the art reports [2] [3] [5].

### 2.1.1 Hand region

The human hand is the source of a number of unique physiological characteristics. Different types of attributes can be identified such as fingerprint, palmprint, hand geometry, hand vein pattern and finger
However most of these systems are an extension of fingerprint technology. Fingerprint recognition systems identify and compare the identity of a user based on the comparison between two fingerprints. Fingerprint systems use the texture of ridges and valleys present on the surface of a fingerprint to identify fingerprints. A summary of recognition techniques for fingerprint can be found in [2] [3]. The authors in [3] point the following advantages of using fingerprints in mobile phones:

- Fast authentication method.
- Good performance rates.
- Users accept fingerprints as an authentication method.

And the following disadvantages:

- Due to age and occupational issues, this technology does not work for all population.
- Present limitations to the environmental conditions.
- Require a contact sensor integrated into the mobile phone since approaches using mobile phone camera do not work properly.
- It is vulnerable to gummy finger.

One of the best methods to evaluate the accuracy of fingerprint systems is through the online platform FVC-onGoing [7]. FVC-onGoing is a web-based system with the purpose of evaluate fingerprint recognition systems. At the moment there are 5427 algorithms evaluated. The best algorithm (MM_FV) has an 0.042% EER.

Some authors have implemented fingerprint recognition systems using the mobile phone camera. The authors in [8] evaluate different mobile phones’ cameras (Nokia N8, iPhone 4, Samsung Galaxy I) for fingerprints recognition. The evaluation results showed an average EER of 45%. The results of experiments indicate it is very difficult to get a desirable performance of fingerprint recognition using mobile phone cameras in real-life scenario due to the complicated background, hand and camera shaking during taking photos. Another reason pointed by the authors was the fact that existing consumer mobile phone cameras are mostly optimized to capture human face or other more “attracting” objects in a frame instead of fingerprints.

Other approach to collect fingerprint samples in mobile devices is through incorporated contact sensors in mobile devices, instead of using their camera.

Contact sensors have performance issues because these kinds of sensors are prone to environmental conditions such as temperature and humidity. Other problems arise from users, such as dirty fingers.
The most important mobile phone manufacturer companies like Siemens, Fujitsu, HTC, Samsung and many others have tried to implement these kind of sensors since 1998. However most of these companies that have tried to implement contact sensors in mobile phones have seen their ideas being rejected by users for usability and security issues.

The author in [9] explains why HTC and some other companies failed the implementations of contact sensors into their mobile phones. The main reason the authors points is the bad implementation of the sensors that were difficult to use and the fact that the sensors were not well integrated with the software of the device so it created the perception of being uncomfortable to the users. Another reason is that HTC made people use their fingerprints for many actions they did not require such security.

On the opposite side, the authors in [10] [11] explain the success of Apple company with this technology.

Apple introduced the fingerprint contact sensor reader in their mobile phone, the iPhone5, in 2013. Along with the release of iPhone5 was released a new Operating System, the iOS7, that brought to Apple’s devices many advantages at a security level.

Older versions of this Operating System (OS) only allowed the use of a PIN (Personal Identification Number) or passwords as security measures. The OS iOS7 offered a significant improvement, with the inclusion of Touch ID (fingerprint recognition), a technology that Apple has acquired after they bought, in 2013, the AuthenTec company.

With Touch ID technology the user could unlock his device and authorize transactions at AppStore and iTunes Store.

It was possible to register more than one user fingerprint in one device (recognizing that some iOS devices will be used and shared by multiple users).

The contact sensor is implemented into the Home button and in order for the user to register his fingerprint the user just has to place his finger onto the Home button a number of times to build a template. This process does not take more than 30 seconds.

The sensor take a high resolution image from small sections of the fingerprint from the sub-epidermal layers of the skin.

Nevertheless, only 2 days after the device went on sale, the German group Chaos Computer Club have cracked the system using a fake fingerprints [12]. They showed that there was no liveness detection included in Apple’s fingerprint system. The group said that in reality, the Apple sensor only had a higher resolution than the others, so they only had to improve the fake fingerprint quality.

Although there are no official FAR and FRR in the iPhone fingerprint, the authors in [13] suggested a 0.05% FAR and a FRR between 2-10%.

But Apple was not the only company who had decided to implement this kind of sensors [10] explain. In February of 2014, Samsung released their mobile phone, the Samsung Galaxy S5. The user were
able to unlock the device and to authenticate towards enterprise systems through the Samsung’s Knox 2.0 service and authenticate payments in Paypal application.

Samsung’s fingerprint sensor technology on the Galaxy S5 uses Fast Authentication IDentity Online (FIDO)-compliant software that enables third party service providers, such as PayPal, to leverage the biometric services on the device. By being FIDO compliant, Samsung is enabling other service providers to leverage the biometric services on the device, like accessing the fingerprint sensor.

The same authors explain that both the Apple and Samsung cases, fingerprint data does not leave the device and stays within protected hardware environments. The Apple and Samsung fingerprint sensor environments uses hardware security components to securely store the biometric template and to support the use of a Trusted Execution Environment (TEE). Apple’s A7 chip uses a ‘Secure Enclave’ co-processor that is used to store all fingerprint data collected from Touch ID and the Galaxy S5 uses ARM’s TrustZone.

Other hand region modality is palmprint recognition. These systems consider lines and creases presented on the palmar region. A study accomplished by the authors in [14] demonstrate that even twin brothers have different palmprints.

The articles [15] [2] can be used as a state-of-the-art of palmprint technique.

The authors in [16] have created their own databases using a mobile phone. To normalize the hand’s posture variation, a hand-shaped guide window was used on the screen, and the embedded flashlight was turned on during the image acquisition. They reached an EER of 2.88%. Unfortunately, the database has not been made available to the public.

With this problem in mind, the authors in [15] created a database with 1,620 images from 15 subjects available to public. The database has been collected from users using four different smartphones (Samsung Galaxy S4, HTC Desire 610, Apple iPhone 5, Huawei P8). The results show it was possible to demonstrate close to 100% matching success on certain subjects, in unconstrained conditions. The unconstrained use case does pose several challenges like background colour similar to the skin colour.

Similar to palmprint is hand geometry biometrics that is based on the measurement of the shape of the contour of the hand, including finger widths at several points, finger lengths, palm shape, and deviation angles [6].

A hand shape biometric system uses a camera or scanner-based device to acquire the hand image of a person from which shape information is extracted and compared against the information stored in a database to establish identity [17]. Hand vein recognition systems utilize the vein bifurcations and endings beneath the skin of human hand [18]. Image acquisition requires specific cameras such as optical, thermal, silicon, or ultrasonic cameras [2]. Due to the difficulty in forging vein patterns, hand vein modality is becoming a great area of interest. Many more algorithms and implementations can be found in other state-of-the-art reports and books like [5] [2] [3].
2.1.2 Facial region

Face recognition is the oldest recognition technique used.

In the last 30 years, several facial recognition techniques have been proposed. Some of these techniques use either 2D or 3D face samples.

In mobile devices, the facial recognition was first introduced by Android systems in 2011, named “Android Face’s Unlock”. As the authors in explain [11] the original version of this recognition application had no liveness detection so it was vulnerable to attacks such as showing a photo to the camera instead of the real face. In order to avoid this kind of attacks, a later version of this system would ask people to blink. That was not a good solution since the attackers just had to modify the original picture of the user with the eyes closes.

Due to this problem, whenever the user was at screen lock menu looking at the available security options, the Android system had the description “Low security” under the option “Face unlock”.

In the field of face recognition in mobile devices, the main reference algorithm is the Viola-Jones algorithm [19]. It achieves high levels of Detection Rate (DR) with quickly processed images.

In [20] the authors propose a face verification system in mobile devices. The proposed algorithm, namely C-APCDA which is based on Viola-Jones algorithm is compared with another algorithm from the system Adaboost. The two algorithms are compared for three different databases (O2FN, AR, and CASPEAL).

The lowest EER achieved by this system is 1.90% on the AR database while the higher value is of 5.60% on CAS-PEAL database. The lowest value of accuracy for Face Detection Rate (FDR) is of 96.98% on O2FN database and the higher value is 98.73% on AR database.

All the presented results have better rates than the results presented to the Adaboost system.

In [21] authors point other problems with facial recognition systems in mobile devices such as the environmental conditions like lightning or background noise in public spaces.

In a recent work the authors [22] implemented a face recognition and face detection application in an Android mobile phone. The authors used software libraries they found in the literature, such as OpenCV Rekognition, CERT, FaceRecognizer, FaceRect, Face++, FaceReader, among others.

The application stores pictures in memory and all the processing mechanisms are performed within the cellphone. Some problems faced by the authors were that the Android API was not enough to make the application, and the other already mentioned was the background noise.

The results of their tests depend on light conditions and face position, and the authors say that face detection process works better than face recognition with more faces on camera. However, no quantitative results, such as FAR or FRR, were presented.
Other face recognition system will be presented in subsection 2.1.6.

Still in face region we have ear, tongue, and teeth, although for the latter two no mobile systems where found.

An ear recognition system identifies a subject by the ear’s shape. The ear shape can be obtained through a picture or pushing the ear against the cellphone surface.

The authors in [11] write about Descartes Biometrics, a company that created a solution for mobile devices using ear biometric authentication. The application, named "Ergo", is available to download in Google Play Store, since it is only available for Android mobile devices.

In order to be authenticated the user has just to press the ear to the cellphone surface (touchscreen). The application combines this authentication mechanism with behavioral biometrics such as the action of the subject’s arm to raise the smartphone to the ear.

The main advantages of using facial recognition modalities are:

• Non-intrusive mechanisms.
• Contactless image acquisition, as opposed to fingerprint.
• Not everyone has fingerprint sensors on their smartphones, but all smartphones have built-in cameras.

The advantages mentioned are the reason why people use face recognition as a secure authentication mechanism. We can find information that proves this evidence in the survey [23] that demonstrate people who use face recognition do it because of the security this mechanism provide.

### 2.1.3 Ocular region

The ocular region possesses the accurate, reliable and almost impossible to forge biometric signatures, such as retina, sclera and iris recognition. Retinal systems compare people based on blood vessels present on the retina. Patterns of blood vessels are always different for each person [24].

These kinds of systems need infrared light in order to obtain an image of the blood vessels. For this reason, to the knowledge of the author of this article, there are no systems with retinal recognition for mobile devices.

In other hand, some systems are implementing iris recognition in mobile devices. In [25] it is implemented an iris recognition system with a NIR camera. The system achieves a detection rate of 99.40% and its processing time is 17.64ms. The results show an EER of 0.1%, and the user can’t be up to 1.1 cm longer to the mobile phone camera, otherwise red-eye effect will occur.

Some related problems to this kind of technology (iris recognition) were not solved, such as the performance degradation on outdoor environments, the performance degradation on people that wear
eyeglasses and performance degradation of off angle-images.

Like all kind of systems, there are some advantages associated, such as:

- High performance rates compared to other biometric technologies.
- Almost impossible to forge ocular region biometric systems.

And disadvantages, such as:

- Usually requires additional hardware.
- Limitations to environment conditions.

More detailed information about ocular biometrics can be read in [5] [3] [2] [26].

### 2.1.4 Behavioral biometrics

This kind of biometrics are different from the already mentioned biometrics. Instead of identifying humans based on physiological characteristics, humans are identified based on how they do the things.

The most relevant behavioral biometrics are:

- Keystroke dynamics
- Voice
- Gait
- Signature

#### 2.1.4.A Keystroke dynamic

Keystroke dynamic systems distinguish people by typing rhythms (the timings of pressing and releasing buttons). This kind of systems are usually associated to a computer keyboard.

As can be read in [3] initial keystroke recognition systems implemented in mobile devices only recognized people when they were typing a PIN or a password. The problem is that whenever the user inserts his PIN or password, the mobile device is unlocked and if the device is then left unattended or stolen, the information on the device becomes freely available.

Although there exist more keystroke systems implemented in computer keyboards than in mobile devices, the implementation of that kind of systems have more advantages if implemented in mobile devices.

Computer keyboards are limited to its keys, while smartphones have different sensors. It means that a user can be authenticated by the strength he holds the smartphone, the orientation of the smartphone,
the speed he types and the finger size.

Authors in [27] studied the feasibility of keystroke dynamics over touch-screen devices. Four factors obtained from keystroke dynamics were obtained: the interval time, the interval timing ratio, the dwell time and the distance between two buttons. A Bayesian classifier was used in machine learning process.

The best results, obtained from the combination of the four factors, show an accuracy of 82.18%, a FAR of 0.02 and an FRR of 0.18.

Using lock patterns to unlock Android smartphones, authors in [28] using a Random Forest (RF) classifier obtained an EER of 10.39%. More works related to keystroke dynamics using lock pattern can be read in [3].

Most of the articles using this technique do not mention battery consumption, that is a disadvantage of this technique [3], since the smartphone’s CPU needs to be constantly processing information while user is typing.

2.1.4.B Voice

Voice recognition systems recognize the user by his vocal characteristics. The user is identified either by speaking words he wants or by reading and speaking words chosen by the system. Voice recognition systems solve some limitations of mobile phones such as storage and consumption. Most of voice recognition systems send the signal obtained to an external computer to be processed.

The biggest challenge with this systems is the environment noise.

The authors in [29] proposed a fusion of the 12 systems, presented in the competition “The 2013 speaker recognition evaluation in mobile environment”, and obtained an EER of 7% for females and 4.8% for males.

In [30] a voice recognition system implemented in Android OS is presented. It uses a text-independent speaker recognition system. The algorithms used are Mel Frequency Cepstral Coefficients (MFCC) and Vector Quantization (VQ). Results show an EER of 4.52%. In [31] the author presents some companies that already use voice recognition in mobile devices. Barclays Wealth and Investment Management, USAA and Banco Santander are banks that use voice recognition for secure entry in mobile device, for transactions and for account access.

Talktalk is a mobile phone company that provides this technology in their phone systems.

ValidSoft is another company that provides voice-based authentication, transaction verification, and mobile authentication for real-time transactions. Several UK banks have implemented these technologies.

As can be read in [3] the big advantages that make companies implement voice recognition systems in mobile phones are:

• All mobile phones have microphones
• Usability.

• Good performance results.

But voice recognition systems have some disadvantages such as:

• Noisy environments.

• Voice may vary depending on people healthiness.

• Voice is easy to replicate.

2.1.4.C Signature

Still in behavioral biometrics there exists signature recognition systems. Since centuries this is the main method to authenticate documents, so it has the advantage of people being familiarized with it.

But there some difficulties about those systems when implemented on mobile devices such as the small input area, the quality of touchscreen, the processing capacity and battery usage [3].

In [32] the authors evaluate the performance of handwritten signature recognition in four different mobile platforms (Samsung Galaxy Note, iPad, Samsung Galaxy Tab and Blackberry Playbook).

A database was created by the authors from 11 users with 3 sessions and 20 repetitions.

Results show an average EER of 2%. Samsung Galaxy Note obtained an EER of 0.29%.

However when tests were carried using skilled forgeries, EER increased an average 10%. As an example, Samsung Galaxy Note EER increased to 10.10%.

2.1.4.D Gait

The last behavioral biometric subcategory is gait. Gait recognition systems identify people by walking style, which is distinctive for each individual.

According to [3] this technique is not seen by the authors as an authentication technique. Some authors say the main purpose of this technique is to detect if a mobile device has been stolen, while other authors propose to use this technique to activate different profiles in mobile phone.

The main problem with this technique using a mobile phone is where to place the mobile phone to capture data, since the mobile phone needs to be always in the same position since signals captured in different parts of the body are different. Other problems are related to the use of clothes, since different shoes affect signal acquisition.

The most complete review for this technique may be found in [3].
2.1.5 Medico-chemical biometrics

Medico-chemical biometrics such as DNA, body odor, electrocardiogram (ECG) have a big advantage, the hardness to forge.

In [33] authors identify people with an ECG sensor. ECG signals, captured by an external sensor, are processed in mobile phones. Results show an accuracy of 99% and a Total Execution Time of 0.214s.

The company Bionym authenticates people based on heartbeat. The company uses a bracelet and the user just has to press the sensor that is built-in the bracelet. However, the big disadvantage of medico-chemical biometrics is that they require additional specialized sensors.

2.1.6 Multimodal biometrics

Previous sections demonstrated that no biometric system achieves 100% accuracy nor 0% ERR. Although quite hard to achieve such results, approximate values can be achieved if biometrics are combined into a single system. Systems fusing biometrics are called, in literature, multimodal systems.

FIRME [34] is an authentication system, for Android mobile devices, that combines face and iris recognition. Data is combined at score level.

FIRME has been used for mobile banking and the system provides two different operation modes: entire processing is performed on mobile device or the final step (decision step) is performed on an external computer.

The authors say that the biometrics don’t need strictly to be fused. One can use face recognition for authentication and iris recognition for a further transaction.

A survey showed that users (47 users) found the system secure and easy to use.

The time required by the application for registration is 4.82s and for authentication is 5.04s. Although FAR and FRR results are presented in linear charts, no EER value is presented.

Other multimodal system is Proteus [35]. Proteus is a multimodal system based on face and voice recognition built on a Samsung Galaxy S5.

An important aspect is that users are recognized independent of the phrases spoken.

Proteus training and testing are done by means of videos. The user just has to record himself in a video speaking. The face is detected using another already mentioned algorithm in section 2.1.2, Viola-Jones algorithm.

Data processing is done in a Trusted Execution Environment (TEE), hardware that is isolated from the rest of the device hardware and software.

Results show an EER of 25.70% for score-level fusion and 2.14% for feature-level fusion. Authors say they will improve the system.
2.2 Framework for secure authentication

Biometrics need to be securely integrated with the system. The author in [36] present three options to integrate biometric modalities into an application, which are (i) using native APIs from mobile platform and device vendors, (ii) using proprietary approaches specific to a particular vendor and (iii) using standards from FIDO consortium.

Regarding the first option, although it seems a quick path to implement the desired functionality, the developers have the responsibility to build the entire authentication system, from the end-user device to the back-end server and implement all transmission process and protocols.

But there’s two important security aspects that the developers need to take into consideration while building an authentication system: (i) the authentication has to occur first locally, that means, the user needs to be authenticated on the device and (ii) after local authentication comes remote authentication, the device needs to be authenticated by the authentication server.

However while native APIs support local authentication they do not perform authentication to the authentication server.

The developers need to build all the remote authentication system leading to the following risks:

- Insecure transmission: After local authentication, credentials sent over the network are vulnerable to attacks such as man-in-the-middle or phishing attacks.

- Insecure storage of credentials: It is a common mistake to store credentials using only application logic rather than using secure hardware storage.

- Comprised devices: Native APIs do not are not capable of verify device attestation, which can lead someone to exploit the API with comprised credentials.

Another problem is the support for multiple Operating Systems (OS). Native APIs come from specific mobile vendors for specific OSs, therefore whenever a new version of an OS is released, the developers have to accommodate the last versions of that OS. At the end code will need to be re-developed, re-tested, and re-deployed whenever a new version of an OS is released, that is a particular problem on Android devices, where new versions of OS are constantly being released.

The second option, the use of proprietary technology, avoid native API problems. However, they have its own costs and associated risks. An organization that implements a particular modality (fingerprint, eye, voice, face) from a specific proprietary is tied to that company. Using other biometric mechanism from other proprietary can’t be done due to interoperability issues. Some vendors provide a hybrid approach with vendor specific and standard approaches, but a good management will be need to differentiate the proprietary from the standard approaches. And of course, there’s a financial risk associated, since while tied to a specific vendor, the vendor can raise their prices over time.
The third option is cheaper, quicker, and more secure. The main emerging standards are FIDO Alliance and Biometric Open Protocol Standard (BOPS) [37].

FIDO Alliance is a consortium of more than 250 members, including companies like Google, MasterCard, Microsoft, Samsung and Visa.

The core ideas of FIDO Alliance are (i) ease of use, (ii) security and privacy, and (iii) standardization. FIDO Universal Authentication Framework (UAF) is a protocol that provides password-less and multi-factor security for online services. The users are able to register their devices to the server by selecting a local authentication mechanism such as fingerprint, voice or face. The UAF protocol allows the service to select which mechanisms are presented to the user. Once registered, the users simply need to repeat the local authentication mechanism, chosen at registration phase, whenever they need to authenticate to the server. The users no longer need to enter their password when authenticating from that device. UAF also allows the combination of multiple authentication mechanisms.

As described in the official FIDO UAF specification [38] the main goals of FIDO UAF are: "support a strong multimodal authentication, facilitate user authentication using only what the user’s device is capable of, enable selection of the authentication mechanism, simplify integration of new authentication capabilities and design extensible protocols and APIs in order to support the future emergence of additional types of authenticators".

Similar to FIDO, there exists BOPS developed by Hoyos Labs. It was recently published as a standard by IEEE.

Both protocols authenticate locally (user device) and the result is then sent to the specific server.

The main difference between FIDO and BOPS relies on biometric requirements, since BOPS specify several minimum requirements for the biometric recognition system, such as thresholds for EER, liveness detection, among others requirements, while FIDO is more abstract, in a way that there exist no requirements for the biometric recognition system.

### 2.2.1 Relation with other technologies

FIDO protocols are not meant to replace other existent protocols. They complement Federated Identity Management (FIM) frameworks, such as SAML, OpenID and OAuth.

FIDO is meant for remote authentication, using biometrics in mobile devices.

While other frameworks need passwords to authenticate and have access to a token, FIDO uses its local authentication to free up a private key stored internally on the user’s device to sign a challenge sent by the server. There is more privacy since there’s no passwords navigating on the network, with FIDO biometric data never leaves the user’s device.
2.3 Application Services

In this section will be described what is BankOnBox and its purpose and what is intended to be developed. BankOnBox is a Self Banking System that allows a bank to provide to their clients a complete Internet Banking Interface to manage their banking accounts online.

The user interface is a Web-based Application implemented using Microsoft .NET technologies. The system provides 2 levels of authentication for a user. A first level is required to login to the system allowing the users to perform queries about their banking information. A second level of authentication is required every time users want to perform an operation that involves transfers of money between accounts, perform changes to their accounts or changing of Internet Banking settings.

Currently the product supports a username/password combination for the first authentication level and each user can have one of the following second level authentication types: Confirmation Key (a second password), SMS One Time Password or SafeNet Digital OTP Tokens.

Each authentication type is a Plugin implementing a well-known interface, encapsulating all of the Provider’s authentication logic and integrations. New authentication mechanisms are added to the system by developing new Authentication Provider Plugins and adjust the user interface to accommodate the new required authentication use cases.

2.4 Conclusion

Table 2.1 sums up the accuracy (percentage of correct results) for studied biometrics that could be implemented using current smartphone’s hardware [2].

Based on all information collected and studied it can be concluded that the best biometrics to use are the fingerprint, face, and voice since they combine the fact that nowadays every smartphone has hardware and software that provide user authentication using those biometrics and have high accuracy levels. Other biometrics, due to hardware limitations of current smartphones, are not suited to use as secure authentication mechanisms.

Using biometrics is not enough to have a secure system. A secure system needs to be implemented using secure protocols for communication.

FIDO Alliance and BOPS are standard protocols for secure authentication. However there are more companies adopting FIDO UAF protocol than BOPS, making FIDO UAF protocol the best option to implement the communication. Aware and Hypr are some examples of companies that offer their services using FIDO UAF protocol.
<table>
<thead>
<tr>
<th>Biometric</th>
<th>Accuracy (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fingerprint</td>
<td>99.9%</td>
</tr>
<tr>
<td>Palmprint</td>
<td>&gt;95%</td>
</tr>
<tr>
<td>Hand Geometry</td>
<td>&gt;95%</td>
</tr>
<tr>
<td>Vein Pattern</td>
<td>99%</td>
</tr>
<tr>
<td>Face</td>
<td>95%</td>
</tr>
<tr>
<td>Ear</td>
<td>&gt;95%</td>
</tr>
<tr>
<td>Iris</td>
<td>99.9%</td>
</tr>
<tr>
<td>Retina</td>
<td>99.9%</td>
</tr>
<tr>
<td>Voice</td>
<td>&gt;90%</td>
</tr>
<tr>
<td>Keystroke dynamics</td>
<td>&gt;90%</td>
</tr>
<tr>
<td>Gait</td>
<td>&gt;90%</td>
</tr>
<tr>
<td>Signature</td>
<td>&gt;90%</td>
</tr>
</tbody>
</table>
3

Architecture

Contents

3.1 Architecture Design Requirements ........................................... 22
3.2 Architectural Modules ............................................................... 24
3.3 Data Flow .............................................................................. 25
In this section will be described the architecture of the system and justifications for decisions taken. Based on the information from the previous chapter, the biometrics suggested to be implemented are fingerprint, voice recognition and face recognition. Since the system aims to authenticate an user using biometrics the system will be called BioAuth.

The proposed BioAuth's architecture will be based on FIDO UAF Protocol, since FIDO UAF objectives of having multiple OS implementations, ease of use, security and privacy of user information, align well with those of the system to be implemented. Another important aspect that lead to chose FIDO instead of any other specification is the amount of information and open source software implementations available on the Internet. FIDO has a lot more information available on the Internet compared with BOPS.

3.1 Architecture Design Requirements

Together with the business sector from Link Consulting company, to clarify the system functionalities, were specified the architecture requirements.

The requirements are divided in functional and non functional requirements.

3.1.1 Functional requirements

• The system shall use mobile devices biometric capabilities for second-factor authentication.

• BioAuth application's registration screen must have three mandatory fields: username, password and user type.

• Before proceeding to registration in BioAuth system, the system shall verify the username, password, and user type fields, inserted by the user on the registration screen, on Business Web Application system, to check if the user is valid.

• If any error occurs during the process of user authentication, the user must be informed by a message in mobile device screen.

• The server shall have access to all present and past information of users, meaning that no user information data should be deleted.

3.1.2 Non Functional Requirements

Usability BioAuth application should be simple and easy for the user to understand.

Performance The system shall behave without any time delay under real-life circumstances and the response time for user operations should not last longer than a few seconds.
Security  All user confidential data must be protected both on the mobile device and during the exchange of messages between the device and the server.

Interoperability  BioAuth system shall be able to work together with other systems, such as BankOnBox and eDocLink from Link Consulting company.

Fig. 3.1 represents the BioAuth’s system architecture and it accommodates all architectural requirements mentioned above.
3.2 Architectural Modules

As can be seen in the architecture (Fig. 3.1), the architecture is composed of the following modules:

1. **User Device** This module represents the user’s mobile device (Android and iOS). For a regular user, the mobile device is the way to perform the registration, authentication, and deregistration in BioAuth system. The protocol is divided into 4 submodules, namely:

   1.1. **Authentication Application** The authentication application is simply the user interface, where the user interacts with the application, responsible for receiving inputs from the user, such as its username and password, and deliver them to the FIDO Client for further processing.

   1.2. **FIDO Client** The FIDO Client is the engine inside the User Device module. FIDO Client is responsible for processing all the information received, to process that information and act accordingly, and to send information to the Authentication System, such as the requests and responses for each action (registration, authentication, deregistration).

   1.3. **ASM** The Authenticator-Specific Module (ASM) is a software interface on top of FIDO Authenticators. The ASM gives a standardized way to the FIDO Client to detect and access the functionality of UAF authenticators and hides internal communication complexity from clients. The ASM is a software component that offers an API to the FIDO Client, enabling the FIDO Client to discover and communicate with one or more available authenticators. Requests like the creation of an asymmetric key and its storage or requesting to ask for fingerprint authentication to the user are sent by the FIDO Client to the ASM, but the FIDO Client does not need to know how it is implemented.

   1.4. **FIDO Authenticators** FIDO Authenticators are the biometrics used for authentication into the system. As an example, a fingerprint is an authenticator.

2. **Authentication System** The Authentication System is responsible for processing all the information regarding register, authentication, and deregistration in BioAuth system. This module contains four submodules:

   2.1. **Communication Service** Is the entry point of all communication with the Authentication System. Every message is first received by Communication Service that then forwards it to the FIDO Server. This way the system is able to accept information from multiple Business Web Application systems. The Communication Service just has to forward it to the FIDO Server with proper information.
2.2. Integration Service  This service is responsible for receiving requests from the Business Web Application, like an authentication request to authorize a transaction and to send the responses back to the Business Web Application. A new Integration Service needs to be created for each Business Web Application that wants to integrate with BioAuth system.

2.3. FIDO Server  FIDO Server is responsible for processing all the requests. It processes the registration, authentication and deregistration requests and acts in conformity with the operation result. It communicates with the database to store and access user data.

2.4. FIDO Database  FIDO Database is responsible to store information related to users registered in BioAuth system, such as username, userID (unique for each user), if the user is a valid user or not, among many other entries.

3. Business Web Application  Represents the system which the BioAuth will integrate.

3.1. Web App Engine  This module represents the business system such as BankOnBox.

3.2. Authentication Provider Plugin  This plugin is responsible to handle the communication between the Authentication System and Business Web Application. If the user is in his online banking account and requests a transaction, the Business Web Application uses the Authentication Provider Plugin to send the authentication request to the Authentication System.

3.3. Credential Verification Endpoint  Used by the Communication Service to verify the user credentials before proceeding to registration. This endpoint is only used for that purpose.

4. OneSignal  OneSignal service is used to send push notifications.

OneSignal is a push notification service for websites and mobile applications and it is totally free, which is unusual among push notification services. OneSignal service supports mobile platforms and even provide dedicated software Development Kit (SDK) for each platform such as native, hybrid and cross-platform such as Xamarin. A RESTful server API is available that enables to programmatically deliver notifications from our own server.

Along with all the features they provide, they have a good and clear documentation available, and the help support is fast and helpful.

All these options made me chose OneSignal as the push notification service to use.

The alternative to using an already existent push notification system was to implement a push notification system using .NET services from scratch, which would be very costly and time-consuming.

3.3  Data Flow

The system data flow is simple to understand for each of the three main functions, registration, authentication and deregistration. Briefly explaining, the user has a key pair, generated at registration time,
that is used to confirm challenges sent by the server, in order to confirm the user identity. The user's private key is securely stored inside the mobile device and the public key is sent to the server. In chapter 4 more details will be given about implementation of secure storage on the mobile device.

However to be able to use BioAuth’s system, the user needs to be registered on a Business Web Application, such as BankOnBox.

The moment the user registers at the Business Web Application, assuming the user is using BankOnBox system, he receives a password to access his user account.

Only after the user being registered in the Business Web Application, he is able to register in BioAuth System.

After registering in BioAuth system all second-factor authentication requests will be performed through biometrics.

The next sub-chapters will explain the data flow for registration, authentication and deregistration operations. Each action (ex. registration) has a figure that represents the data-flow. That figure only assumes the normal flow of responses and results by all parties in the system. However, in the descriptions, all scenarios with positive and negative responses, are described.

### 3.3.1 Registration data flow

#### 3.3.1.A Pre-Registration

Registration data flow is showed in Fig.3.2. Before a user being able to register in BioAuth system, using the mobile app, the FIDO Server must be informed that the user will request to register in the system. It’s like a pre-registration message that is sent to the FIDO Server saying that the user is able to register in BioAuth system. This request is performed by a bank operator.

In the particular case of BankOnBox, the choice of the authentication mechanism (One Time Password (OTP), biometrics, etc.) is controlled by the Business Web Application. When the user registers and signs up for a new online banking account using BankOnBox system, the user chooses the first and second-factor authentication mechanisms. If the user wants to change that agreement, he just has to inform that decision to the bank operator. Another reason is, this way only authorized people can register on BioAuth, otherwise, every user registered in BankOnBox could register in BioAuth.

When the bank operator sets the second-factor authentication of the user to biometrics, a message is sent to the FIDO Server (pre-registration message) that informs the server that a user is now able to proceed to registration in BioAuth system.
### 3.3.1.B Registration

BioAuth's registration itself starts on the user device. The user launches the BioAuth client-side application and selects the registration option. The user enters its credentials, namely, the username, password (the password to access his Web Business Application account) and user type (singular or enterprise) and press the submit button.

First, a message is sent to the Communication Service asking to initiate registration.

The Communication Service needs to confirm if the user is already registered in the Business Web Application and to confirm his credentials are correct.

If the user is not yet registered Business Web Application, a message is sent to the user device informing about the situation. Otherwise, if the user is already registered and if the username, password, and user type, submitted by the user are correct, the protocol continues and the Communication Service informs the FIDO Server that a registration was required.

The FIDO Server creates a registration request with information regarding that specific request message and that specific user, sends it back to the Communication Service that forwards the request message to the Authentication Application. The request message contains a challenge to be signed by the user device.

When the Authentication Application receives the registration message, the local registration starts. The user selects which biometric wants to use (at the moment only fingerprint biometric is available). As in the example of the diagram (Fig. 3.2) the user wants to register the fingerprint in the system, so he swipes the finger on the fingerprint reader.

In order to be authenticated locally on the mobile device, the user needs to previously have that biometric registered on the mobile device, meaning that in order to register that specific finger on the system, that finger needs to be already registered in the mobile device. The reason for this is that, in fact, the fingerprint is a way of authorizing the mobile device Operating System (OS) to perform some privilege actions. Any application that needs to store private information inside the mobile device, in secure hardware that prevents data from being extracted (secure hardware will be explained in detail in section 4), that user must have the smartphone secured "at the eyes" of the OS, using a secure method available in the mobile device. The OS considers the smartphone secure, if the user has settled a "lock screen", such that every time the user has to unlock the phone he has to prove that he has permission. The user can set the lock screen to be unlocked by a PIN, a password, a pattern or by fingerprint. This is why the user needs to previously have the fingerprint already registered in the mobile device, because, first, because the OS requires the mobile device to have a secure lock screen, with one of the secure methods mentioned before, like fingerprint, to allow and authorize privilege actions such as accessing data stored inside secure hardware and, second, because BioAuth application requires that the user has the lock screen settled and that lock screen must be defined to be unlock by fingerprint, so at least
one fingerprint must enrolled in the system. Summing up, the fingerprint data never leaves the device and FIDO Server do not have access to fingerprint data at any moment. The fingerprint is used locally to authorize the OS to perform privileged actions. Those actions will be explained now.

After the user used the fingerprint, OS verifies if the presented fingerprint matches the one already stored internally. If so, the user is locally verified and the process of local registration continues.

After local registration succeeds, a pair of asymmetric keys is created by the FIDO Client. The private
key is stored internally in the mobile device, in secure hardware (it is possible since the user has authorized this action by using his fingerprint). That private key is bounded to the BioAuth application and can only be used in future actions by presenting the fingerprint registered. This way even if the mobile device is stolen, the attacker can not access the key. Also, cryptographic operation, like signing and verifying, run inside the secure hardware.

The FIDO Client then creates the registration response. That message contains an hash of the challenge, the authenticator certificate, the public key created and a signature composed of those three fields signed with the client’s private key.

The registration response is sent to the Communication Service, that forwards it to the FIDO Server. The FIDO Server processes the response and after verifying the information received, it stores a tuple containing the public key, user identifier and the registered authenticator identifier in the database. The FIDO Server informs the Communication Service that the process of registration is completed with success and the Communication Service informs the Authentication Application that the registration process is completed with success.

If the same user wants to register another device, the server just adds a new entry to the database at the time of registration and both devices will be valid for the same user.

3.3.2 Authentication data flow

Fig.3.3 shows the use case when a user needs to be authenticated in order to authorize the transaction (second-factor authentication).

The process begins with the user, in its online banking account, submitting a transaction to the Business Web Application.

After the transaction was submitted, the Business Web Application informs the Communication Service about the operation desired.

The Communication Service informs the FIDO Server that a transaction by that specific user was submitted and needs to be authenticated. So, the FIDO Server creates an authentication request. This authentication request contains a challenge, to be signed by the user, along with some other additional information for communication purposes.

The FIDO Server creates a push notification request using OneSignal API. Inside that push notification request is the authentication request (to be sent to the FIDO Client) and information about which device to send the data (a unique device ID per-device).

The notification request is then sent to the OneSignal push notification service that creates a notification and sends it to the user specified by the server.

The device receives the notification and shows it. By clicking on the notification message the app will open and ask the user to authenticate its registered biometric.
The user authenticates using the biometric registered in the system, fingerprint following the scenario in Fig.3.3. When the device authenticates the user biometric, the private key is able to be used by the FIDO Client, that will sign the challenge sent by the server.

The FIDO Client starts creating the authentication response containing the signature of the challenge along with other additional information for communication purposes.

The authentication response is then sent to the Communication Service that forwards it to the FIDO Server for verification.
Finally, the FIDO Server sends the result information to the Communication Service to be forwarded to the Authentication Application. At the time the result information arrives the Authentication Application, a pop-up message will display a message informing the user about the operation result.

The user knows the transaction was authorized, since he received a message saying that the authentication operation has succeeded.

In BankOnBox, instead of the system having an open connection waiting for the authentication response from the FIDO Server, the user has a button that shows the user the status of the authorization action.

So at the end, the user knows the operation was authorized by BioAuth, but BankOnBox system will only show that information on the user BankOnBox account when the user presses the button to confirm the transaction authorization status.

### 3.3.3 Deregistration data flow

The fig.3.4 shows the data flow to deregister the user.

#### 3.3.3.A Pre-Deregistration

The user the user needs first to have a pre-deregistration call holding on the database, for the same reason as the pre-registration message, it is BankOnBox policy and to avoid non authorized BankOnBox users to deregister in the system. This way, if some user loses his mobile device, the bank operator can send a pre-deregistration request and the server will not process future authentication requests.

That pre-deregistration call is performed by some manager of the system, in this particular scenario, following the example of fig.3.4, it’s the bank operator who sends the pre-deregistration information to the FIDO Server.

Only an authorized person can ask the bank operator to send a pre-deregistration request. Of course, a regular user does not say to the bank operator to specifically send a pre-deregistration request. Instead, a regular user tells the bank operator that wants to use a different authentication mechanism instead of biometrics for second-factor authentication, and when the bank operator changes the authentication mechanism to, for example, one-time-password, a pre-deregistration request is sent to the FIDO Server and only after this is the user able to proceed to deregistration.

#### 3.3.3.B Deregistration

The deregistration is performed locally on mobile device.

The user just needs to press the deregistration button shown on the app’s main screen and confirm that wants to deregister.
A deregistration request will then be sent to the Communication Service to be processed by the FIDO Server. Assuming that a pre-deregistration was already created and the user is still a valid user, the FIDO Server will create a deregistration request and send it to the FIDO Client. If none of the mentioned conditions are valid, namely, if no pre-deregistration exists or if the user is no longer a valid user, the FIDO Client will be properly informed and a pop-up notification will show that information to the user on the screen of the mobile device.

Following the scenario where all the conditions are valid, the Authentication Application will receive the deregistration request sent by the server and the user will need to authenticate the deregistration request, locally on mobile device by using the biometric selected on the system. Following the example of fig.3.4 the user uses fingerprint.
When the user is successfully authenticated locally on the mobile device, the private key is used to sign the challenge created by the server.

The FIDO Client will then create the deregistration response that contains the signature of the challenge along with additional information for communication purposes and sends it to the Authentication System to be verified by the FIDO Server.

The FIDO Server verifies the response and sends the response back to the user’s mobile device. A pop-up message will show the user the result of the deregistration operation.

After the process being successfully completed, the user entry is not deleted from database. Instead a flag saying that the user is no longer active is set to ‘true’, so an administrator will be able, in the future, to see information about previously registered users.

3.3.4 Security Considerations

The communication in the system can be divided in two parts. The communication between the User Device and the Authentication System, and second the communication between the Business Web Application and the Authentication System.

3.3.4.A User Device - Authentication System

The communication protocol is documented on FIDO Specification, where all messages are described, each parameter for every message and security measures to take into consideration for registration, authentication and deregistration.

As documented in the specification the communication is performed via HTTPS, to prevent Man-In-The-Middle attacks.

The communication web service used is REST, where all messages are document in FIDO Specification as JSON. The first message of registration, when the user sends his credentials (username and password) is the only message not specified by FIDO Specification. That message is sent as a JSON object and through HTTPS (as said before) like all other messages.

All messages are sent using HTTPS POST where data is contained inside HTTP request body.

The FIDO Server also relies on OneSignal service to send the push notifications. As stated by OneSignal documentation, for iOS OneSignal uses the TLS encryption for sending notifications as required by the Apple APNS specifications and for Android uses Google’s GCM/FCM service which uses the HTTPS protocol.

User authentication is performed by the user mobile device signing a challenge sent by the server, using the user’s cryptographic private key. In section 3.3.4.C more details will be given about mobile device security using keys.
3.3.4.B Authentication System - Business Web Application

The communication between the Authentication System and the Business Web Application is handled by the Communication Service (Authentication System) and Authentication Provider Plugin (Business Web Application).

The communication is through HTTPS using SOAP Web Services, which prevents from external attacks.

To prevent from internal attacks, from people inside BankOnBox network, on top of that communication, there is an external security layer that uses Microsoft WS-Security [39]. This extra security layer uses mutual authentication of the server and client with X.509 certificates [40].

3.3.4.C Mobile Device Security

User sensitive data must be securely stored inside the mobile device in order to prevent data leakage.

The security of data user is one system requirement specified, and it is achieved by storing all user sensitive data inside secure hardware.

Secure hardware is a separate hardware module where data can be stored with an extra level of security. In iOS is a co-processor called Secure Enclave Processor (SEP) [41], a processor that run separately from the main processor.

In Android it can be stored in two places, depending on the device manufacturer.

First the Secure Element (SE), a tamper-resistant secret store, like a microchip or SIM-card. Its main purpose is to store cryptographic secrets in a way that illicit use is hard or impossible to do [42] [43].

Second, using using the TEE. The Android hardware can be divided into two different OS [44]: the normal OS where usually all applications run and the secure world OS. This secure world is usually called TEE. The Android key storage providers, like the KeyStore, use this environment to securely store the keys and cryptographic operation, like signing, are ran inside this TEE.

Each OS (Android and iOS) have its own keystorage providers. Keystorage providers are API to manage cryptographic keys. Then, each keystorage provider will store the keys depending on the device hardware capabilities. The Android system keystorage provider is called KeyStore, and iOS system is called Keychain. In both OS the cryptographic key pair is bound to that specific app and only BioAuth application can access its own cryptographic keys.
In this chapter it will be explained BioAuth’s implementation. Will be described used tools and decision taken at each step of the process.

First will be described the system development process, where will be explained the methodology approach and the schedule.

Later will be described the main difficulties while implementing the system.

### 4.1 Development Process

The system was implemented following a waterfall model (Fig.4.1) where each module was implemented at a time.

At the end of implementing a module, I just had to integrate the current module with the others already implemented.

This way integration of all modules was easy.

![Figure 4.1: BioAuth’s schedule](image)

### 4.2 Development Environment

The system was implemented using Microsoft technology, for mobile and for server implementation. Mobile device application was implemented using Xamarin Forms. Xamarin is a cross-platform software from Microsoft that makes programmer’s life easier, since a single application can be built into multiple Operating Systems (OSs).

This way there is no need to learn another language, like Swift for iOS. In Xamarin Forms, using C# programming language, an application is written for both OSs, Android and iOS.

The Authentication System, where the FIDO Server is located, were implemented using ASP.NET Web API Technology. ASP.NET Web API is a framework for building web APIs on top of the .NET Framework.
4.3 User Device

The User Device is composed by the Authentication Application, FIDO Client and ASM submodules. The Authentication Application submodule is in fact the user interface of the application, the FIDO Client is the submodule responsible for data processing in the mobile device and the ASM submodule is responsible to access and make use of biometrics and other specific-device features such as accessing keystore system.

The user device module, was implemented using Xamarin. A Xamarin solution can be composed of different types of projects. Some of these projects are platform-specific projects and the others are shared project types or modules that make it possible to reuse code across platforms.

The authentication application (the user interface) and the FIDO Client were implemented using a shared library. This way it was just needed to be implemented once to be deployed for Android and iOS.

Before starting to explain the user interface, the first thing the application does is to verify if:

- Fingerprint sensor is available on that mobile device;
- The device has fingerprint activated as screen lock
- If the application has permissions to use fingerprint sensor;
- If no user is already registered on the BioAuth’s system;

If all conditions are met the device is able to register the user in BioAuth’s system. However if any of these conditions fails, a message is presented to the user (Fig. 4.2). After checking all conditions the device, using OneSignal API, registers his device OneSignal ID, in OneSignal system. This OneSignal ID is obtained by using OneSignal API, that gives a unique identifier to each device. The code listing 4.1 shows how the application starts the OneSignal service and how it gets the OneSignal ID.

Listing 4.1: Example of Refit call

```csharp
1. OneSignal.Current.StartInit("Here-is-the-OneSignal-project-ID")
2. .HandleNotificationOpened(HandleNotificationOpened)
3. .EndInit();
4. string OneSignalID = OneSignal.Current.IdsAvailable();
```

4.3.1 User Interface

The User Interface (UI) was designed using XAML. XAML stands for eXtensible Application Markup Language, is an XML-based language created by Microsoft as an alternative to programming code for instantiating and initializing objects to define the layout of UI. A single XAML builds the same screen for two different OSs. Fig. 4.3(a) and Fig. 4.3(b) represent the application main screen. The former represents the screen for Android, while the later represents the screen for iOS devices. When the user
presses the "Register" button, the registration page is presented. The registration page for both Android and iOS can be seen in Fig. 4.4(a) and Fig. 4.4(b).

A form is presented to the user, where the user has to fill the fields with the credentials he uses to sign in into his Web Application Business account. The application on the screen was implemented with BankOnBox system in mind, so there's an extra field, the third one, where the user select which type of user he is ("Singular" or "Enterprise") (Fig. 4.5(a)). Since a user may left an empty field, after pressing the register button, an information message is shown to the user informing to fill all the fields (Fig. 4.5(b)). When the user presses the registration button the information on the fields is gathered in a class called "Credentials" and sent to the FIDO Client.

When the user presses the "Deregister" button, in Android a request is immediately sent to the FIDO Client requesting deregistration, while on iOS an extra step is needed.

On iOS, the user needs to type the username he wants to deregister. This problem is due to the implementation of APIs in Xamarin. The Keychain API in Xamarin does not have all iOS native Keychain methods implemented. One of these not implemented methods is the one to the list objects stored in the keychain. Due to this problem, it is only possible to access one item at a time on Keychain. The same does not happen with Android devices since the Keystore API has all native methods implemented.

Since only one user is registered per-device, if the user enters a wrong username, the application will throw a message informing that the username entered is invalid.
Figure 4.3: Android and iOS Registration Screens

(a) Android Main Screen  (b) iOS Main Screen

Figure 4.4: Registration screens on Android (left) and iOS (right)
4.3.2 FIDO Client

The FIDO Client processes all the messages that come from the user or from the server.

As said earlier, FIDO Client was implemented using a shared library project, so a single implementation was needed for both OSs. The FIDO Client processes the three functionalities of the system (register, authenticate and deregister). Each functionality implementation will be explained in detail next.

4.3.2.A Registration

Registration is triggered by the user when he presses the register button. As said at the end of the last chapter, the info of the user is gathered in a class named “credentials” and is sent to the FIDO Client.

The first thing the FIDO Client does is to send the credentials to the Authentication System to verify if:

- User is registered in the Business Web Application system
- User’s credentials are correct in Business Web Application
- A pre-registration request already exists in the Authentication System database
If all three conditions are met, the user can proceed with the registration and the FIDO Server will send a registration request message containing a challenge.

That message sent from the FIDO Client to the server, are sent via HTTPS using REST Web Services. But instead of implementing the communication from scratch, I used a plugin called Refit.

Refit [45] is a plugin, that simplifies the creation of HTTP REST requests. First, an interface is defined, as can be seen in the code listing 4.2.

The endpoints are defined inside of an interface using special Refit annotations to encode details about the parameters and request method and each endpoint specifies an annotation of the HTTP method (GET, POST). In brackets can be seen that the method is an HTTP POST, and the endpoint is specified after that. Below the endpoint and HTTP method specification is the method call interface. In the code example the register method is showed, it receives an object of type "Credentials", that is sent in the body of the HTTP request.

Listing 4.2: Example of Refit interface

```java
[Post("/api/fido/register/")]
Task<RegistrationRequest> Register([Body] Credentials credentials);
```

Then the "RestService" uses HttpClient to make its calls and generates an implementation of the specified interface as can be seen in the code listing 4.3.

The RefitService is first called and the base URL is passed as a parameter. "GeneralVariables" is a class that holds variables shared in the system. The base URL is one of them.

Listing 4.3: Example of Refit call

```java
RefitInterface refitInterfaceObject = RestService.For<RefitInterface>(GeneralVariables.urlLink);
RegistrationRequest registrationRequest = await refitInterfaceObject.Register(credentials);
```

So at the end, the requesting URL will be the base url, from "GeneralVariables", plus the path specified in the Refit interface. Example, base url = "https://172.16.20.58:443/Servidor/" and the path is "/api/fido/register/" as presented in the code listing 4.2. So the final URL will be "https://172.16.20.58:443/Servidor/api/fido/register".

However the most interesting thing about Refit plugin is that he converts objects to JSON and vice-versa. The "Credentials" object in code listing 4.2 is converted in runtime to JSON and in code listing 4.3 a JSON object is received and converted back to a "RegistrationRequest" object.

After sending the credentials and receiving the registration request from the server, the FIDO Client will process the information sent by the server. That information contains:

- Information about the version of the FIDO UAF protocol used. At the moment only version one is used.
- The challenge created by the FIDO Server
• A policy that specifies which biometrics and which cryptography algorithms are supported and accepted.

Based on the information received, the FIDO Client will first ask the ASM to create an RSA key pair with the information regarding which cryptography algorithms are supported and accepted by the server.

Since the ASM is implemented in a different project (a platform-specific project), the FIDO Client needs to use a "DependencyService". DependencyService allows applications to call into platform-specific functionality from shared code. An example of a DependencyService can be seen in the code listing 4.4. In the code, "Fingerprint" is an interface, that the ASM implements and, "createAsymmetricKeyPair_IOS(keyID)" is a method of that interface.

Listing 4.4: Example of DependencyService call

```csharp
bool result_KeyPairCreated = await DependencyService.Get<Fingerprint>().createAsymmetricKeyPair_IOS(keyID);
```

While creating the RSA key pair, the user will be prompt to use it's fingerprint in order to authorize the creation of an RSA key pair and its secure storage on mobile device. Fig. 4.6 show the application asking the user to use it’s fingerprint.

More details about fingerprint and key storage will be given at ASM section.

![Figure 4.6: Android screen asking for user's fingerprint](image)

The FIDO Client will then begin the creation of the registration response message. This message will contain information regarding this user, such as:
• KeyID, which is the identification of the key pair in the mobile system, like an alias of this particular key pair. This identifier will be used later in authentication.

• The user’s username, to be stored in the database.

• A signature of the challenge sent by the server.

• The user’s public key.

• The OneSignal device identifier.

This information is sent in a byte array in a Type-Length-Value (TLV) encoding scheme. The “Value” is the object converted into its representation in bytes, the “Length” is the numeric “Value” length converted into bytes, and the “Type” is the identifier of what is being sent. The “Type” value is specified in the FIDO Specification document.

All these values are transformed into a TLV byte array, and all arrays representing the different objects are concatenated into a single byte array and added to the registration response message.

The registration response message is then sent to the server to be verified. A result of the server verification will be received and the FIDO Client will return that result to the user interface to be shown to the user.

4.3.2.B Authentication

The ASM, uses platform-specific APIs for each mobile OS. The authentication process starts at the FIDO Server that sends a push notification to the User Device.

When the user device receives and opens the notification, the application, if not running yet, will automatically starts.

In Android some applications, even after being closed, will still run in background. But some applications stop running.

If BioAuth application is stopped, the device will not receive notifications. This is bug reported and documented by OneSignal, and no solution exists to this problem.

However, in general, the most common scenario is that after closing BioAuth application, the app will still run in background. Back to the authentication implementation, when the application opens, the FIDO Client will export the data in the notification. That data is an “Authentication Request” message sent by the server and encapsulated into the notification message body. The authentication request is serialized and sent as a JSON object from the server and the FIDO Client just deserializes it back to an “Authentication Request” object.

The authentication request message comes with a challenge created by the server. The FIDO Client
extracts the challenge and asks the ASM to sign it. However, the ASM needs some additional information in order to know which key to use. Here, is where the KeyID is needed. The KeyID was sent back in registration phase to the server.

To sign the challenge, the private key needs to be accessed so a message will be showed asking the user to authorize that action using his fingerprint.

After signing the challenge, the FIDO Client sends the authentication response message to the server that contains the signature and the transaction identifier along with some additional information such as the version of the FIDO protocol, the operation type, etc.

The server will verify the signature and send a result. The FIDO Specification document specifies the first two messages, the "Authentication Request" and the "Authentication Response". However, the last message where the FIDO Server must inform the FIDO Client about the result of the signature verification, no specification exists in FIDO Specification document.

Due to this, I created a message called "Authentication Result" that contains a boolean value to know if there exist any errors and a list of strings where the server can write all the errors, if an error occurred, and send it to the FIDO Client. This way, instead of just sending a boolean value to the FIDO Client informing about the result, if the operation succeeded or not, I thought it was important to know that if the operation didn’t succeed, what caused that error and to inform the user about the error (code listing 4.5).

### Listing 4.5: AuthenticationResult object

```csharp
public class AuthenticationResult
{
    public bool hasErrors { get; set; }
    public List<string> errorList { get; set; }
}
```

A similar message object was implemented in registration ("Registration Result") and in deregistration ("Deregistration Result").

The final step in the authentication is the user receiving the operation’s information as in Fig 4.7.

#### 4.3.2.C Deregistration

Similarly to the registration process, the deregistration only succeeds if first, there exists a pre-deregistration request on the FIDO Server.

Imagining a scenario of online Banking, the user tells the bank operator that doesn’t want biometry as a second factor authentication. The bank operator edits that option and immediately a pre-deregistration request is sent to FIDO Server, informing that a deregistration will occur for that specific user. That pre-deregistration process occurs between the Authentication System and the Business Web Application, so it will be explained in more details later on this chapter. A pre-deregistration request is needed to protect
the user in the worst scenario where the mobile device is stolen and the thief can fake the fingerprint. To prevent this scenario it is not possible to deregister without having a pre-deregistration request.

The FIDO Client knows the user wants to deregister when the user presses the deregister button and immediately informs the server that this user wants to deregister by sending the username. The server sends a “Deregistration Request” containing a challenge, and the FIDO Clients asks the ASM to sign it. As said earlier, every time the ASM signs data the user is prompted to authorize the action by using fingerprint.

After having the signature, a “Deregistration Response” message is sent to the FIDO Server containing the signature, along with other data.

The FIDO Client then receives the “Deregistration Result” message, informs the user about the result of the operation and if the result of the operation is successful the FIDO Client asks the ASM to delete the key pair entry.

### 4.3.3 Authenticator-Specific Module (ASM)

The Authenticator-Specific Module, as the name suggests, is the module specific to interact with the authenticators. The authenticators are the biometrics, the fingerprint is an authenticator while voice is another authenticator.

In this project only fingerprint authenticator was implemented since Android and Apple do not provide APIs to face and voice authentication to store and access keys.
In Android fingerprint authentication was implemented using Fingerprint API while in iOS was using TouchID API.

To store the user’s RSA key pair in Android it is used the Android Keystore, in iOS is used the Keychain.

4.3.3.A Android KeyStore and Fingerprint

Android has many key storage providers, such as Android Keychain, Android Keystore and third-party providers such as Bouncy Castle, or Spongy Castle (android version of Bouncy Castle). The authors in [44] [41] compare Android Keystore with BouncyCastle. Android Keystore results to be more secure since keys are stored inside secure hardware. Using Android KeyStore only my application has access to the keys it generates. Since Android 6.0 it the KeyStore is hardware-backed. This means a dedicated cryptography chip with internal memory is being used to secure the key material.

At the moment of user’s registration, a key pair is created when the FIDO Client asks the ASM to register the user locally and create the key pair. The key pair is created using following code (4.6).

```java
Listing 4.6: RSA key pair creation
1 KeyPairGenerator kpg = KeyPairGenerator.GetInstance(KeyProperties.KeyAlgorithmRsa,
    "AndroidKeystore");
2     kpg.Initialize(
3             new KeyGenParameterSpec.Builder(keyID,
4                KeyStorePurpose.Sign || KeyStorePurpose.Verify)
5                .SetSignaturePaddings(KeyProperties.SignaturePaddingRsaPkcs1)
6                .SetDigests(KeyProperties.DigestSha256)
7                .SetUserAuthenticationRequired(true)
8                .SetUserAuthenticationValidityDurationSeconds(1 * 30)
9                .Build()
10          );
11
12 KeyPair keyPair = kpg.GenerateKeyPair();
```

The first thing to do is to create an instance of “KeyPairGenerator”. The algorithm and the provider are specified (line 1). The algorithm used is RSA and the provider to store the keys is “AndroidKeystore”, the default provider.

The “KeyGenParameterSpec” class, introduced in Android 6.0, ensures the the correct key usage in the application, since it allows to specify specific key storage attributes.

The “keyID” is the alias given to the key. That alias is specified in FIDO Specification and it is build by concatenating the username with the hash of the application. An example of a key alias is "username.android:apk-key-hash:Id3S/Valp2nTnUhqiAgYNfD8M".

Next, the purpose to which the key is being created. In this system, is to sign and verify data. Further is specified the padding (Pkcs1), the signature digest (Sha256). The padding and the digest are specified in FIDO Specification.

The attribute "SetUserAuthenticationRequired" sets whether this key is authorized to be used only if the user has been authenticated. By default, the key is authorized to be used regardless of whether the
user has been authenticated. The use of the key must be authorized by the user by authenticating to
this Android device using fingerprint. "SetUserAuthenticationValidityDurationSeconds" sets the duration
of time for which this key is authorized to be used after the user is successfully authenticated.

Key size was not specified since the default RSA key size using "AndroidKeyStore" provider is 2048-
bit, the size used for RSA keys in this project. The size is specified in FIDO Specification.

Summing up, the key is stored in "AndroidKeyStore" provider. In order to access the key, the user
must be authenticated to authorized the use of the key. For 30 seconds the key is able to be used by the
system after user’s authorization. The key algorithm is RSA, its purpose is to sign and verify data, uses
“PKCS#1” padding, the digest is “Sha256” and the key size is “2048-bit”.

Now that the key pair was created and stored, the user must authorize the use of key by using fin-
gerprint. The first thing to do is to give access to the application to use the fingerprint hardware (code
4.7).

Listing 4.7: Give application permission to access fingerprint hardware

```java
<uses-permission android:name="android.permission.USE_FINGERPRINT" />
```

There are two ways to implement fingerprint authentication. The first one is through the "Fingerprint-
Manager" class, by instantiating a "FingerprintManager" and calling its "authenticate()" method. The
second method is by using the "KeyguardManager.createConfirmDeviceCredentialIntent" flow.

While implementing the first method, a bug was found. That bug is already known and is documented
in many Android forums. When calling the "authenticate()" method, callback methods are registered to
handle possible outcomes of the authentication process (i.e. success, failure, or error).

What happens is that after calling the "authenticate()" method, it goes to the callback method "onAu-
thenticationSucceeds", which is the expected result, since the fingerprint authentication succeed. Oth-
ewise the callback method "onAuthenticationError" would be called instead. So, inside "onAuthentica-
tionSucceeds" callback method is where the signing task should be performed since the authentication
succeeded (this is all documented in Android developers web site). But, when I’m trying to sign the data,
the system says that the user is not yet authenticated, even though the signing task is being performed
after a successful fingerprint authentication and inside the "onAuthenticationSucceeds" callback method.
Even though the user authenticates successfully using fingerprint and the callback method "onAuthen-
ticationSucceeds" is called too, the system does not assume user authentication. It is a known and
documented bug that happens in many devices, such as Nexus, LG, and many others, and at the mo-
ment the reason for this problem was not found yet and no solution exists too.

Due to this problem I had to go the second way. This way works as follows, a "KeyguardManager" ob-
ject is instantiated and an intent is created by calling the "keyGuardManager.CreateConfirmDeviceCredentialIntent()"
method. When the intent is started the user is prompted to present his fingerprint.

It goes to the "OnActivityResult()", the result is analyzed, and if the authentication was successful the
data is signed. The code listing 4.9 demonstrates its implementation.

Listing 4.8: Ask fingerprint authentication and data signing

```csharp
public void askFingerprint()
{
    KeyguardManager keyGuardManager =
        (KeyguardManager)Xamarin.Forms.Forms.Context.GetSystemService(Context.KeyguardService);
    Intent i = keyGuardManager.CreateConfirmDeviceCredentialIntent("Registration
        confirmation", "Please swipe fingerprint to authenticate registration");
    Activity act = Xamarin.Forms.Forms.Context as Activity;
    act.StartActivityForResult(i, 1);
    await Task.Delay(5000);
}
```

In "onActivityResult" method, the KeyStore is used to get access to the key pair. Is important to note
that, the key material of the private key is never returned, what is returned is a pointer to the private key.

After checking that the authentication succeeded ("resultCode == Result.Ok") the data is signed by
creating an instance "Signature" object, using SHA256 padding for RSA algorithm. The data is then
signed and verified.

The data to sign was stored inside the static class "GeneralVariables", by the FIDO Client. The sig-
nature and the public key is then saved on "GeneralVariables" class too. The public key is stored using
DER encoding scheme.

A – Security on Android When hardware-backed KeyStore is available, keys are generated and
used in a Trusted Execution Environment (TEE) or a Secure Element (SE), and the operating system
can’t access them directly [41].

But, not all modern devices have a built-in chip hardware-backed KeyStore. Since those devices are
not able to store keys inside secure hardware, a software approach is used. The keys of a software-only implementation are encrypted with a per-user encryption master key. However, an attacker can access all keys stored on rooted devices that have this implementation [44] [41].

Older devices do not even include KeyStore, however, they have other key storage providers. But since it is not entirely secure [41] I chose to not support Android versions prior to Android 7.0, by setting in Android manifest file the minimum target platform to SDK version 24 (Android 7.0 Nougat).

Listing 4.9: Setting Android minimum and target SDK version
```
<uses-sdk android:minSdkVersion="24" android:targetSdkVersion="26"/>
```

About the cipher algorithm, the FIDO Specification leaves at the discretion of the programmer to chose between RSA and Elliptic Curve Digital Signature Algorithm (ECDSA) for asymmetric keys. I chose RSA algorithm because RSA is widely deployed, so better industry support is available, in contrast to ECDSA [46] [47].

Since I am using RSA, the size of the key must be 2048-bit long, the padding could be Probabilistic Signature Scheme (PSS) or PKCS#1. I had to chose PKCS#1 padding since on server side PSS padding isn’t supported. The signature digest is SHA256. Those parameters were already explained before, however here I am emphasizing that I chose them because I followed FIDO Specification and they specify these parameters for keys.

4.3.3.B iOS TouchID and Keychain

The first thing to do in iOS is to verify if the system has fingerprint hardware (TouchID) which is simply performed by using the following code 4.10. The device lock screen, similar to Android, must have the TouchID authentication available. After a successful verification of the device TouchID, follows the storage of keys.

Listing 4.10: Verify if iOS device has TouchID
```
bool result = context.CanEvaluatePolicy(LAPolicy.DeviceOwnerAuthenticationWithBiometrics, out AuthError);
```

To store and access sensitive information on iOS, the Keychain API is used. Keychain is built on top of the SEP, a coprocessor that provides cryptographic operations for data protection and key management. A device-specific hardware key - the device UID (Unique ID) - is embedded into the secure enclave, ensuring the integrity of data protection even if the operating system kernel is compromised.

Keychain is a database providing secure storage for passwords, keys, certificates and notes for one individual Apple ID. An application always has access to its own unique keychain items, and cannot access any keychain items of other applications, similar to Android.

To create an RSA key pair in iOS a dictionary containing the attributes types and its respective values
has to be created as showed in code 4.11.

Listing 4.11: Creation of a dictionary containing RSA key attributes

```csharp
private NSDictionary BuildKeyPairAttributes(int keySize, string KeyAlias)
{
    IList<object> keyBuilder = new List<object>();
    IList<object> valueBuilder = new List<object>();
    keyBuilder.Add(IOSConstants.Preloaded.constKSecAttrIsPermanent);
    keyBuilder.Add(IOSConstants.Preloaded.constKSecAttrApplicationTag);
    keyBuilder.Add(IOSConstants.Preloaded.constKSecAttrAccessControl);
    valueBuilder.Add(NSNumber.FromBoolean(true));
    valueBuilder.Add(KeyAlias);
    valueBuilder.Add(new SecAccessControl(SecAccessible.WhenPasscodeSetThisDeviceOnly,
                                           SecAccessControlCreateFlags.UserPresence));
    NSDictionary privateKeyAttr = NSDictionary.FromObjectsAndKeys(valueBuilder.ToArray(),
                                                                  keyBuilder.ToArray());
    keyBuilder.Clear();
    valueBuilder.Clear();
    keyBuilder.Add(IOSConstants.Preloaded.constKSecAttrKeyType);
    keyBuilder.Add(IOSConstants.Preloaded.constKSecAttrKeySize);
    keyBuilder.Add(IOSConstants.Preloaded.constKSecPrivateKeyAttrs);
    valueBuilder.Add(IOSConstants.Preloaded.constKSecAttrKeyTypeRSA);
    valueBuilder.Add(keySize);
    valueBuilder.Add(privateKeyAttr);
    return NSDictionary.FromObjectsAndKeys(valueBuilder.ToArray(), keyBuilder.ToArray());
}
```

The parameters of the dictionary are security system constants. The class "IOSConstants" initializes the constants as showed in code 4.12.
Listing 4.12: Accessing security constants in iOS system using Xamarin

```csharp
public IOSConstants()
{
    var handle = Dlfcn.dlopen(Constants.SecurityLibrary, 0);
    try
    {
        constKSecAttrApplicationTag = Dlfcn.GetStringConstant(handle, "kSecAttrApplicationTag");
        constKSecAttrKeyType = Dlfcn.GetStringConstant(handle, "kSecAttrKeyType");
        constKSecAttrKeyTypeRSA = Dlfcn.GetStringConstant(handle, "kSecAttrKeyTypeRSA");
        constKSecAttrKeySize = Dlfcn.GetStringConstant(handle, "kSecAttrKeySizeInBits");
        constKSecAttrIsPermanent = Dlfcn.GetStringConstant(handle, "kSecAttrIsPermanent");
        constKSecPrivateKeyAttrs = Dlfcn.GetStringConstant(handle, "kSecPrivateKeyAttrs");
        constKSecClass = Dlfcn.GetStringConstant(handle, "kSecClass");
        constKSecClassKey = Dlfcn.GetStringConstant(handle, "kSecClassKey");
        constKSecPaddingPKCS1 = Dlfcn.GetStringConstant(handle, "kSecPaddingPKCS1");
        constKSecAccessibleWhenPasscodeSetThisDeviceOnly = Dlfcn.GetStringConstant(handle, "kSecAttrAccessibleWhenPasscodeSetThisDeviceOnly");
        constKSecAttrAccessible = Dlfcn.GetStringConstant(handle, "kSecAttrAccessible");
        constKSecAttrAccessControl = Dlfcn.GetStringConstant(handle, "kSecAttrAccessControl");
    }
    finally
    {
        Dlfcn.dlclose(handle);
    }
}
```

There are some particular constants, aside from key attributes such as the alias, the size, etc. The constant "kSecAttrIsPermanent", set to the boolean value "true", indicates that the key pair should be stored in the Keychain at creation time. In code 4.11, at line 12, it is specified that every time the the key pair needs to be accessed, the user should authorize its use by presenting his fingerprint.

After creating the dictionary, its time to create the key pair (code 4.13).

Listing 4.13: Creation of RSA key pair in iOS

```csharp
SecKey.CreateRandomKey(keyGenerationAttributes, out NSError errCode);
```

The moment the key is created it is immediately stored inside Keychain.

Now, every time the key needs to be accessed, the user will be prompted to use his fingerprint. However there is a difference between Android and iOS. In Android I had to force that authentication screen every time I wanted to authenticate the user. In iOS this is automatic. When I try to access the key pair and use it, as the code 4.14 demonstrates, it automatically asks the user for local authentication.
private SecKey GetKeyFromKeyChain(string KeyAlias)
{
    var foundKey = SecKeyChain.QueryAsConcreteType(
        new SecRecord(SecKind.Key)
        {
            ApplicationTag = KeyAlias
        }, out SecStatusCode errCode);
    return foundKey as SecKey;
}

public async Task<bool> askFingerprint()
{
    SecKey key = GetKeyFromKeyChain(GeneralVariables.keyID);
    //************* CHECK IF THE KEY IS ABLE TO SIGN AND VERIFY **************
    bool canSign = key.IsAlgorithmSupported(SecKeyOperationType.Sign,
        SecKeyAlgorithm.RsaSignatureMessagePkcs1v15Sha256);
    bool canVerify = key.GetPublic().IsAlgorithmSupported(SecKeyOperationType.Verify,
        SecKeyAlgorithm.RsaSignatureMessagePkcs1v15Sha256);
    if (canSign && canVerify)
    {
        //************************************* SIGN DATA*****************
        NSError errorSign;
        NSData dataSigned = key.CreateSignature(SecKeyAlgorithm.RsaSignatureMessagePkcs1v15Sha256,
            NSData.FromString(GeneralVariables.dataToSign), out errorSign);
        byte[] signature = dataSigned.ToArray();
        //********* VERIFY DATA AND STORE ON 'GeneralVariables' ************
        if (signature != null)
        {
            GeneralVariables.signatureBytes = dataSigned.ToArray();
            NSError errorVerifyMethod1;
            bool result = key.GetPublic().VerifySignature(SecKeyAlgorithm.RsaSignatureMessagePkcs1v15Sha256,
                NSData.FromString(GeneralVariables.dataToSign), dataSigned, out errorVerifyMethod1);
            GeneralVariables.signatureBytes = signature;
            GeneralVariables.signatureString = Convert.ToBase64String(signature);
            GeneralVariables.opSuccess = true;
            GeneralVariables.publicKeyBytes = key.GetPublic().GetExternalRepresentation().ToArray();
        }
    }
    else
    {
        await App.Current.MainPage.DisplayAlert("Error", "Your personal key is unable
to sign or verify data", "Ok");
    }
    return GeneralVariables.opSuccess;
}

The data is signed, then verified locally, and is stored, such as the public key, in “GeneralVariables”
static class. The public key is stored using DER encoding scheme, as FIDO Specification specifies.

A – Security on iOS  
iOS Keychain is similar to a database, where personal user items are stored. Each item stored is described by attributes and the values are stored encrypted. In iOS the SEP stores a master password. Every time an item in Keychain needs to be accessed it is first decrypted on SEP. The diagram below (Fig. 4.8) illustrates how an application interacts with the Keychain.

At the moment the SEP only supports NIST-P-256 elliptic curves, that unfortunately, the Cryptogra-
Figure 4.8: iOS flow of an application interacting with the keychain

The cryptographic community does not consider this curve safe to use [48]. Using RSA algorithm, the keys still need to be stored in the Keychain.

When the RSA private key is used, first the secure enclave is responsible for determining a successful match from fingerprint data from the Touch ID sensor against a registered fingerprint. The fingerprint sensor (Touch ID) is operated by the SEP and fingerprint data is never exposed to any other parts of the system. After a successful fingerprint match, the SEP will then decrypt the private key and return it to the keychain.

Although the key pair is stored in the Keychain, an application can access only its own keychain items [49] [41], so no other application can access the keys stored by BioAuth application.

4.4 Authentication System

This module is composed by four submodules:

- The Communication Service analyses all HTTP message requests and forwards them to the FIDO Server. It receives the HTTP messages directly from the User Device.

- The Integration Service handles the messages from the Business Web Application and forwards the messages to the Communication Service to be analyzed.

- The FIDO Server, the engine of the system, responsible for processing all information.

- The FIDO Database where user data is stored.
The Authentication System was implemented using ASP.NET Web API, a framework to build web APIs on top of the .NET Framework. It is an object-oriented framework that simplifies the implementation of RESTful Web APIs and the relation with databases. The Authentication System, it is composed by a "Controller" and "Models", as specified in ASP.NET Web API documentation.

The "Controller" is the object that handles HTTP requests and creates the HTTP response, in my system is the "Communication Service".

A model is an object that represents the data in the application. Models are represented as C# classes. In this system, the models are the messages specified in FIDO Specification.

The next sub-chapters will explain in detail the implementation of each submodule.

### 4.4.1 Communication Service

Communication Service is a "Controller". The public methods of a controller are called action methods.

When the Web API framework receives an HTTP request, it routes the request to an action. To determine which action to invoke, the framework tries to match the URI against the route template in the routing table.

The code below (4.15) shows the defined route. That route is defined in a file called "WebApiConfig.cs" where all routes are configured.

**Listing 4.15: Communication Service routing**

```csharp
config.Routes.MapHttpRoute(
    name: "DefaultApi",
    routeTemplate: "api/{controller}/",
    defaults: new { id = RouteParameter.Optional }
);
```

The Communication Service, responsible for communication, has the endpoints defined as attributes, as shown in code listing 4.16.

**Listing 4.16: Communication Service class showing the endpoints**

```csharp
[RoutePrefix("api/fido")]
public class AuthenticationServiceController : ApiController
{
    (...)
    [Route("authResponse/")]
    [HttpPost]
    {
        AuthenticationResult authenticationResult = fidoServer.processClientAuthResponse(authResponse);
        return Ok(authenticationResult);
    }
    (...)
}
```
The method "ProcessClientAuthResponse" (code 4.16) would be accessed by the following URL "http://IP_ADDRESS:PORT/api/fido/authResponse".

The message content is sent from the User Device and serialized as JSON. The Web API automatically deserializes the JSON object from the body of HTTP message and creates the correct corresponding class object.

The Communication Service does not only processes the requests from the User Device, it does process the requests from Business Web Application too.

The Business Web Application may use SOAP or REST Web Services. The Integration Service handles the communication with the Business Web Application from the Authentication System side and will use the same Web Service as the Business Web Application does. In BankOnBox scenario, the Integration Service uses SOAP and forwards all requests to the Communication Service.

The Communication Service class is implemented as follows:

**Registration** Three methods.

The first method that receives the pre-registration request from Business Web Application.

The second method, to inform that a registration is requested. That method receives the user credentials (username, password and user type) from the client and verifies the user credentials on Business Web Application. Credential verification request is sent to Credential Verification Endpoint (fig.3.1 in the chapter 3). This endpoint is only used for this purpose, to verify the user credentials in BankOnBox system.

If the user is valid, ask FIDO Server to create the registration request and send back the registration request to the client, otherwise inform the user the credentials are incorrect.

The third method to receive the registration response message, sent from the client. The message is forwarded to the FIDO Server for verification and is sent back a registration result message to the client informing about the registration result operation.

**Authentication** Two methods.

The first method to receive the request from Business Web Application that the user must authenticate an action (second factor authentication), such as a transaction or access to a protected document. Then ask the FIDO Server to send a push notification to the client with the authentication request attached.

The second method which receives the authentication response message from the client. The message is forwarded to the FIDO Server for verification and is sent back an authentication result message to the client informing about the authentication result operation.

**Deregistration** Three methods, similar to registration. The first to receive a pre-deregistration from SOAP Business Web Application.
The second method, to inform that a deregistration is requested. That method receives the user ID, that was stored on the mobile device. The Communication Service will then inform the FIDO Server that a deregistration is requested, the FIDO Server creates a deregistration request and the Communication Service sends the deregistration request message to the client.

The third method to receive the deregistration response message, sent from the client. The message is forwarded to the FIDO Server for verification and is sent back a deregistration result message to the client informing about the deregistration result operation.

4.4.2 Integration Service

The Integration Service is responsible for communication between the Authentication System and the Business Web Application. The communication may use RESTful or SOAP Web Services.

In this system, the Integration Service communicates with BankOnBOx system using SOAP. The Integration Service is a SOAP Web Service created using Windows Communication Foundation (WCF) framework (a framework to build service-oriented applications) from .NET.

4.4.3 FIDO Server

FIDO Specification does not specify FIDO Server's architecture and database integration only specifies its functionalities.

The main functionalities of the FIDO Server are the verification of the messages received from outside modules, such as User Device and the Business Web Application messages, the ability to store data and access the FIDO Database and the ability to send push notifications.

4.4.3.A Messages verification

One of the most important tasks of the FIDO Server is the verification of the client signature. The signature is performed by the client by signing a challenge sent by the FIDO Server. That challenge is sent on registration, authentication, and deregistration request messages. That challenge is created using a random number generator (code 4.17). The class "RNGCryptoServiceProvider", from .NET framework, is used to create a cryptographic Random Number Generator (RNG) and the method "Get-NonZeroBytes" to create a sequence of random nonzero values. FIDO Specification only mentions that the minimum challenge length is 8 bytes. The challenge length from FIDO Server in this system is 32 bytes long.
Each time the FIDO Server creates a challenge for a user, that challenge, along with the user ID, in the database, to verify the signature from the user.

The three response messages contain the signature that are verified by the server.

The public key representation [50] is sent with a different encoding from an Android device and from an iOS device. Android devices send the public key encoded in ASN1-X.509-DER [51] while iOS encode as ASN1.PKCS#1-DER [52]. Due to a different public key encoding, the methods used to reconstruct the public key and verify the signatures had to be different for Android and iOS.

The code 4.18 illustrates how the public key is reconstructed and the signature verification is performed when the signature comes from an Android device.

The code 4.19 illustrates how the public key is reconstructed and the verification of a signature is performed when the signature comes from a iOS device.

Both codes, 4.18 and 4.19, use BouncyCastle cryptography library. BouncyCastle cryptography library was used in this project since it was stated by the users in many online forums to be the best library for cryptography.

Listing 4.17: Generation of the challenge

```java
private static int challengeLength = 32;

public static byte[] GetChallenge()
{
    var challenge = new byte[challengeLength];
    using (var random = new RNGCryptoServiceProvider())
    {
        random.GetNonZeroBytes(challenge);
    }
    return challenge;
}
```

Listing 4.18: Verification of signature by the FIDO Server for Android devices

```java
 AssemblyVersion GENERATION
AsymmetricKeyParameter asymmetricKeyParameter = PublicKeyFactory.CreateKey(clientPubKey);
RsaKeyParameters rsaKeyParameters = (RsaKeyParameters)asymmetricKeyParameter;
RSAParameters rsaParameters = new RSAParameters();
rsaParameters.Modulus = rsaKeyParameters.Modulus.ToByteArray();
rsaParameters.Exponent = rsaKeyParameters.Exponent.ToByteArray();
RSACng rsaCng = new RSACng();
rsaCng.ImportParameters(rsaParameters);

SHA256Managed hash = new SHA256Managed();
hashedData = hash.ComputeHash(challenge_Sent);

bool rsaCngDataOk1 = rsaCng.VerifyData(challenge_Sent, signatureFromClient, HashAlgorithmName.SHA256, RSASignaturePadding.Pkcs1);
bool rsaCngHashOk = rsaCng.VerifyHash(hashedData, signatureFromClient, HashAlgorithmName.SHA256, RSASignaturePadding.Pkcs1);
```

Listing 4.19: Verification of signature by the FIDO Server for iOS devices

```java
 AssemblyVersion GENERATION
AsymmetricKeyParameter asymmetricKeyParameter = PublicKeyFactory.CreateKey(clientPubKey);
RsaKeyParameters rsaKeyParameters = (RsaKeyParameters)asymmetricKeyParameter;
RSAParameters rsaParameters = new RSAParameters();
rsaParameters.Modulus = rsaKeyParameters.Modulus.ToByteArray();
rsaParameters.Exponent = rsaKeyParameters.Exponent.ToByteArray();
RSACng rsaCng = new RSACng();
rsaCng.ImportParameters(rsaParameters);

SHA256Managed hash = new SHA256Managed();
hashedData = hash.ComputeHash(challenge_Sent);

bool rsaCngDataOk1 = rsaCng.VerifyData(challenge_Sent, signatureFromClient, HashAlgorithmName.SHA256, RSASignaturePadding.Pkcs1);
bool rsaCngHashOk = rsaCng.VerifyHash(hashedData, signatureFromClient, HashAlgorithmName.SHA256, RSASignaturePadding.Pkcs1);
```
4.4.3.B Push notification

Push notifications were implemented using OneSignal API. The FIDO Server creates a push notification request and sends it to the OneSignal service.

To notification is sent via HTTPS, and inside the body of the HTTP request is the notification content. That content is encoded as JSON, as specified in OneSignal documentation. It could be done by creating a string object and inserting all the tags and values on the string. However it was too costly to create all message fields. Fortunately there exists a library, called "OneSignal.CSharp", that makes the task of creating the notification much easier, by providing methods for each field of the notification. The code 4.20 illustrates how it is done.

Listing 4.20: Verification of signature by the FIDO Server for iOS devices

```csharp
OneSignalClient oneSignalClient = new OneSignalClient(APP_KEY);
NotificationCreateOptions notificationCreateOptions = new NotificationCreateOptions();

// APP ID
notificationCreateOptions.AppId = new Guid(“16c79cae-5cf3-48be-b5ec-5daa96ab9211”);

// Content
notificationCreateOptions.Headings.Add(LanguageCodes.English, "BioAuth");
notificationCreateOptions.Contents.Add(LanguageCodes.English, transactionText);

// Player ID
List<String> userList = new List<String>(){};
userList.Add(OneSignalUserID);
notificationCreateOptions.IncludePlayerIds = userList;

// Data
Dictionary<string, string> dix = new Dictionary<string, string>();
string jsonAuthReq = JsonConvert.SerializeObject(authenticationRequest);
dix.Add("AuthenticationRequest", jsonAuthReq);
notificationCreateOptions.Data = dix;

// *** SEND NOTIFICATION
NotificationCreateResult notificationCreateResult =
    oneSignalClient.Notifications.Create(notificationCreateOptions);
```

In OneSignal a user can create multiple notification apps, one for each system the user wants to implement. Each app has an ID and a key for app-specific operations.

The notification message is defined in the "header" and "content"parameters.

The list of users receive the OneSignal User Ids unique for each device. At the moment it is send just to a single user. That OneSignal user ID is stored in database along with other user information.

The authentication request message, that contains the challenge, is added as "additional data".
4.4.3.C Integration with a database

To integrate the FIDO Server with a database it was used Microsoft Visual Studio 2017 with Entity Framework. I decided to use Entity Framework since it is an object-relational mapper that let me work with the database using .NET objects. This way I had no need to use data-access code (SQL) to access and store data. I only had to focus on using objects of my domain and not on the underlying database tables and columns where this data is stored.

Entity Framework development can happen in either of two ways: by building an object model from an existing database, or by building a database from a newly created object model. This latter approach is known as Code-First. With Code-First approach I created the database based on the objects I wanted. To understand it better, I will start explaining how it was implemented.

Listing 4.21: Database class that inherits from DbContext and DbSets defined

```csharp
public class DatabaseController : DbContext
{
    public DatabaseController(): base("name=DatabaseController")
    {
    }

    public DbSet<UserInformation> userInformation { get; set; }
    public DbSet<AuthReqToDatabase> authenticationRequest { get; set; }
    public DbSet<DeregistrationRequestsDB> deregistrationRequests { get; set; }
    public DbSet<PreRegistrations> preregistrations { get; set; }
    public DbSet<PreDeregistrations> prederegistrations { get; set; }
}
```

The code 4.21 illustrates the creation of the class "DatabaseController" responsible for database management such as adding or searching for data. That class inherits from DbContext. DbContext is the class responsible for interacting with data as objects, such as when a search is performed it populates objects with data from a database, or when data is added to database it persists data to the database.

The constructor of the DatabaseController class receives a string. That string indicates the name of the entry in the configuration file indicating where to read details about the connection string and the data provider. The file were the connection details are written is the "web.config" file. In listing 4.22 is illustrated the database configuration string in "web.config" file.

Listing 4.22: Connection string in "web.config" file where is specified the database connection configuration for FIDO Server

```csharp
<connectionStrings>
  <add name="DatabaseController" connectionString="data source=DESKTOP-8I1U6SC;initial catalog=FidoUafDb; Persist Security Info=True;User ID=sa;Password=SomePassword" providerName="System.Data.SqlClient" />
</connectionStrings>
```

The first tag is the entry of the connection string configuration ("DatabaseController") that must matches the one passed in the "DatabaseController" constructor. Then is the "connectionString" tag
where the database connection configurations for FIDO Server are specified. The "data source" is the
name of the server, in the example is the name of my laptop. The "initial catalog" is the name of the
database, that I gave the name of "FidoUafDb". The following tags "Persist Security" set to "True" means
that every time a connection is established the "username" and "password" fields should be used. If it
was set to "False", it would only use the password for the first connection. The last tag "providerName"
specifies the driver to use.

After the connection configuration being specified what is left is to define the database tables. Since
I followed the Code-First approach, I had to specify the table contents in code. It was easy using entity
framework since I only had to create my object models and create "DbSet" objects that specify the tables.
The code listing 4.21 shows, below the class constructor, multiple "DbSet" specified. Each "DbSet" has
a class type. As an example, in line 6, the "DbSet<UserInformation>" is a "DbSet" of "UserInformation"
type. The "UserInformation" type is a class I defined 4.24, like all other types.

Every time a DbSet object is defined, the database needs to be updated. Every update to the
database is stored as a log in a folder called "Migrations". The first thing to do is to add a "migration",
that is an update to the database, to the list of updates needed to be performed. It is performed by using
the command line of visual studio with the command "Add-Migration optional_description_text". Multiple
"migrations" can be added before the final database update. After all changes being done and all "mi-
grations" being added, the database needs to be updated with all the changes ("migrations") that were
performed. It is done in the Visual Studio command line too, with the command "Update-Database".
After this command, the database is updated.

An important thing to do when creating the class that will represent a table in database is to define
the primary key of the table, defined with the tag ":[Key]" as is showed in code listing 4.24. More tags are
available if needed, however in this system no more tags were needed.

After specifying the database connection, creating and update the database tables, it is possible to
interact with the database using Language-Integrated Query (LINQ) syntax. Code listing 4.23 illustrates
to query the database. The first one, line 1, uses LINQ syntax to find any user in "UserInformation" that
has the username value equals to "username_example" value. The second example, line 2, shows how
information is added to a table. In this example, the table to add information is the "UserInformation"
table, represented by the "DbSet<UserInformation>userInformation" (code 4.21, line 6), using the "Add"
method and passing as an argument an "UserInformation" object.

Listing 4.23: LINQ query example

```csharp
1 bool userExists = databaseController.userInformation.Any(o =>
    o.Username.Equals("username_example"));
2 databaseController.userInformation.Add(userInformation_object);
```
4.4.4 FIDO Database

FIDO Specifications do not specify what needs to be stored in the database or how it should be implemented, it only says to store the user information and the public key. The rest is up to the developer.

The FIDO Database was named “FidoUafDb” and is composed by six tables:

Pre-Registrations Requests and Pre-Deregistration Requests Both tables are used to store information about pre-registration and pre-deregistration messages. Both have three fields: "TokenID" the unique user ID in the system, the username and a field that says if the pre-registration or pre-deregistration actions is still holding to be verified, "isStillPending".

Registration Requests, Authentication Request and Deregistration Requests These three tables contain the ID of each message, or action. The ID of the user, the challenge sent and a field, named "verified", to check if the action was already verified by the FIDO Server. Example, when the FIDO Server creates a registration request, that registration action is given an ID, a challenge is created, and these information is stored in database along with the target action user ID and the "verified" field set to "false". When the Registration Request arrives to the FIDO Server, it checks the action ID to know which row of the table to get the action information. It verifies the signature with the challenge stored and after a successful verification he changes the "verify" field to true.

UserInformation This table contains the information about the user. The "keyID" is the primary key. It is used to identify the user and when a push notification is sent this value is sent to the user, so he knows the key alias in his keystore system. That value is unique, and is specified in FIDO Specification as the user identifier. "tokenID" is the identifier of the user in the Business Web Application. The "Username" and "PublicKey" are, as the names suggest, the username and the public key respectively. The public key is represented as a string, encoded in DER encoding scheme. "OneSignalUserID" is the user device identifier in OneSignal system. "isValidUser", as the name suggests, is the field that represents the state of the user in the system. When the user successfully deregisters, this value is set to "False", otherwise it stays "True".

The code listing 4.24 shows the "UserInformation" class.

Listing 4.24: UserInformation class

```csharp
public class UserInformation
{
    [Key]
    public string KeyID { get; set; }
    public string tokenID { get; set; }
    public string Username { get; set; }
    public string PublicKey { get; set; }
    public string OneSignalUserID { get; set; }
    public bool isValidUser { get; set; }
}
```
In the system, no user entry is ever deleted. When the user deregisters his respective “isValidUser” value is set to “False” and his information is still kept in the system, such as user information and all authentication requests. This way, in the future, it may be useful to get information about any issue regarding this user.

4.5 Business Web Application

The Business Web Application represents a system like a home banking system or a file management system. It was not implemented by me. I just integrate my BioAuth system with this already existing system.

To perform this integration there is needed a module to perform the communication with my system. This module is the “Authentication Provider Plugin” that will communicate with the Authentication System through the “Integration Service”.

4.5.1 Authentication Provider Plugin

BankOnBox uses the concept of providers to authenticate with external systems. I developed my own provider to authenticate with the BioAuth framework.

Authentication Provider Plugin handles the requests (pre-registration, authentication, authentication verification, and pre-deregistration) from Web Business Application to the Authentication System.

The current implementation uses SOAP Web Services since all BankOnBox communication structure uses WCF with SOAP. It accesses the Integration Service (Authentication System) with SOAP web services (consumes) using the WCF framework.
## Contents

5.1 Usability Tests ................................................. 64  
5.2 Interoperability Tests ......................................... 66  
5.3 Security Tests .................................................. 67  
5.4 Performance Tests .............................................. 69
After the system implementation, tests were carried out with the intent of finding errors or other defects and to verify if the system was fit for use.

The tests were performed to guarantee that the system fulfilled all the requirements, functional and nonfunctional, specified in section 3. With that in mind, the following types of tests were performed: usability tests, interoperability tests, security tests and performance tests. The next sections will describe the tests in detail.

5.1 Usability Tests

Usability tests were mandatory in this system to verify if users understood and could easily interact with the Graphical User Interface (GUI) in the application. Another important thing to test was to verify if all messages, during the process of authentication, showed to the users were clear for the user to understand.

Usability tests were performed without the BankOnBox component fully integrated (Business Web Application). It was missing the webpage from BankOnBox, where the user performs the transactions, and the webpage where the bank operator sends the pre-registration and pre-deregistration requests. Those messages were sent by me using SoapUI, to simulate BankOnBox behavior.

The tests were carried out by me and tested on eight users (seven using Android devices, one using an iOS device).

The tests were conducted as follows:

• Before starting the test tasks I asked the users, after opening the application on mobile device, that before I explain the application purpose if they could explain beforehand what they thought the application would do. All users answered that the application was to register in some system. It meant that at first sight, all users would understand the purpose of the application. Then I explained to half of them the application purpose, but not to the other half, so I could test if those users, who did not explain the system purpose, knew what to do when the fingerprint screen appeared.

• After the introductory question, I gave tasks for users perform. The first task was to know if the user could register in the system without explaining any step of the process.

All opened the application and in the home screen all users pressed the “Register” button, successfully.

When the register screen appeared, seven out of eight users asked which credentials should they use. I explained they could invent their own credentials. One user missed the “User Type” field and was successfully prompted by the application to fill all the fields, which that user said that warning
was useful.

After all fields filled, all users pressed the “Register” button to submit the registration. The next screen to appear was the fingerprint authentication screen where the user had to use the fingerprint to authorize the creation and use of cryptographic keys. This detail is hidden from the user. The user only needs to know it is required to register in the system.

Three out of the four users that I didn’t explain the purpose of the application, asked what to do when that screen appeared, while the other user understood immediately what to do. I told them to read the message that was showing on the screen. The message said “Please swipe fingerprint to authenticate registration” and those users understood immediately what to do (some users did not know where the fingerprint scanner were). The other half of users that I explained the purpose of the system, acted correctly with no doubts of what to do.

After all users used the fingerprint, the result message was showed saying “Authentication successful”. All users asked after that message, if the process of registration was done, since they thought it was too fast compared with similar systems.

I explained them the process of registration was completed and that we were moving for the next task, the authorization of a transaction by fingerprint authentication.

- The task of authentication was performed easily. They received a push notification message on the mobile device. All users opened the notification message and read the transaction text. After reading the transaction text some users pressed the notification screen balloon instead of the button saying “Authenticate” under the transaction notification text. I explained them that in mobile devices they have to press the button “Authenticate” to close the message. When the notification balloon closed, the fingerprint authentication screen appeared. They were already familiar with that screen since they already have seen that screen while performing registration. All users knew what to do, and all used fingerprint correctly. The last step was the authentication status message. They read the message saying “Authentication successful” and understood the transaction was confirmed.

The last task was to deregister.

- The deregistration task is in a point different from Android to iOS, but all users performed this last task without any problem. Android users just pressed the “Deregistration” button on home screen, authenticated successfully with fingerprint when fingerprint screen showed up and understood the process of deregistration was completed after reading the message of deregistration completed. The iOS user, knew he had to press the “Deregistration” button on home screen too and did it successfully. Then the deregistration page showed up where the user had to fill the username field with the already registered username and submit the deregistration request. The user did
it just fine, without problems like with Android users, used the fingerprint when fingerprint screen showed up and understood the process of deregistration was completed after reading the message of deregistration completed.

At the end of the tests, I asked for the users give an opinion about the experience and what could be done to improve the usability of the application. All users liked the experience and said it was easy to perform the tasks. However all users said, the GUI had to be improved by giving it a better look instead of just buttons in a white background.

5.2 Interoperability Tests

Since the system was implemented following a waterfall model, it was crucial to perform interoperability tests to ensure that each module would communicate with other modules without any compatibility issues.

The tests were conducted as follows:

- Verifying that the User Device was sending messages to the Authentication System using RESTful Web Services communication. It was carried out by sending registration messages from the User Device, with Android and iOS emulators and with real devices, to the Authentication System registration endpoint and receive response messages on User Device again. FIDO Specification is well specified, so no errors occurred.

- Verifying that the Authentication System and the Business Web Application were successfully sending messages to each other. Since I had no access to the Web App Engine, like BankOnBox web site, I had to perform this tests using an external tool. I used SoapUI, an Open Source and free tool for testing Web Services. In this case to test SOAP Web Services between the "SOAP Plugin" (Authentication System side) and the "Authentication Plugin" (Business Web Application side).

  I sent messages using SoapUI simulating the "Authentication Plugin" behaviour. Everything went fine, just added the Web Services Description Language (WSDL) URL to SoapUI and the exchange of messages, between the two modules, occurred without errors.

  The result was that the integration of all modules was easy and no errors occurred.
5.3 Security Tests

The security tests, in BioAuth system, could be divided in three parts: the security of cryptographic key storage in mobile devices, the security of the communication and the security of the FIDO Server.

5.3.1 Cryptographic Key Storage Security

While investigating about ways of testing the security of keys storage for iOS and Android [41] [44], what I found that the only way to retrieve information from key storage providers is by rooting (Android) or jailbreaking (iOS) the mobile device. Jailbreaking (iOS) removes the restrictions Apple puts in place and allows to install third-party apps that can access confidential information stored on the mobile device since Apple’s restrictions are no longer valid on a jailbroken device. Rooting basically gives an user access the entire operating system, allowing the user to access system parts, or directories, that were protected before by the Android OS.

The only thing that could be done in BioAuth system is to prevent the application to be installed on a rooted or jailbroken device, by adding extra code to the application [41]. Although new ways of rooting are always being used by malicious users it still is something to add in the code in the future.

5.3.2 Communication Security

The entire system communication is based on HTTPS protocol, so the messages are cyphered.

Although I couldn’t do more, to my knowledge after some online investigation, I tried to sniff the message content exchanged between the User Device and the Authentication System using Wireshark tool. Wireshark is a network packet analyser that captures network packets and displays the packets data as detailed as possible.

I could see the HTTP messages being exchanged, however I could not any useful data from the messages since the data was ciphered. The verification of message exchange between the Authentication System and the Business Web Application was performed too and the result was the same, no useful data could be read from human eyes.

However I just did a security verification, since I just verified that the content of messages was unreadable. I did not attack the message content, to decipher the data, neither performed any automated test with any testing tool.

5.3.3 FIDO Server Security

To test FIDO Server security I investigated online for an automated tool to perform automated tests. However, I could not find any free automated test tool. Even so, I performed a static analysis of the FIDO
Server, by analyzing the code and functionality, based on a STRIDE threat model [53].

**Spoofing Identity** Spoofing identity means a malicious user becomes any other user or assume the attributes of another user.

This threat cannot be avoided [53], it may happen in BioAuth system, by a thief stealing the user's smartphone and using a fake fingerprint. However, since the authentication is performed through biometrics, this threat is less likely to occur than in a system with password, since counterfeit fingerprint is more difficult to perform [3] (depending on the password).

FIDO Specification leaves the biometric implementation to the developer, but they also mention this threat as a problem that cannot be avoided but is mitigated by the use of biometrics.

**Tampering with Data** Tempering with data means that a malicious user can change or manipulate the data.

Even though the communication is performed by HTTPS and any information can be extracted, a malicious user simply randomly manipulate the information of the message to create entropy in the system.

To mitigate this problem, there is a field in any message sent by the server named “serverData”, specified in FIDO Specification, which is a random value, created by the FIDO Server and that the client can't change and must send it as it was received to the client. This value is used to identify the communication connection and to identify if the data in the message was randomly manipulated, since it is encrypted (HTTPS). If so, the FIDO Server discards that message.

**Repudiation** Repudiation means that a user may reject an action he performed by telling he did not performed that action.

In this system all transactions are logged and all messages contain a signature, signed by the user private key, that only the user has access to, by using his fingerprint to unlock the private key usage.

This way non-repudiation is granted in this system.

**Information Disclosure** Information disclosure enables an attacker to gain valuable information about a system. But in BioAuth’s system all message exchange is performed through HTTPS so all HTTP message data, even the headers, are encrypted. The attacker cannot access any privileged information about the user.

**Denial of Service** A malicious user might send a large amount of messages to make the system runs slowly or even stop working.

The FIDO Server verifies the HTTP header of every message received, and discard all messages that comes with an incorrect “Content-Type” HTTP header, without even processing the message. This measure is not the most secure and does not prevent completely this threat, but at least
prevents some malicious messages to be accepted and slow the system. According to [54] there are not many effective ways to prevent Denial of Service (DoS) attacks. However some additional methods can be implement in the future, such as deploy a firewall or third party services to prevent some DoS attacks [55].

**Elevation of Privilege** In BioAuth system all users have the same account type, so there is no elevation of privilege threat possible since there are no special permissions for any particular users. The users just authorize transactions as a second factor authentication mechanism and the only permission they have is to use the BioAuth system as their biometric authorization system.

### 5.4 Performance Tests

Performance tests were carried out to check how the Authentication System behaves and performs under a different number of concurrent virtual users performing transactions over a certain period of time using SoapUI tool.

To message chosen to be sent was the first message request, when the user informs the Authentication System he wants to register in BioAuth and sends the username and password. In this test the message response from the Authentication System is always a message informing that the registration is not possible since the username sent is always the same, and a single user can only register once. But I choose this message because it requires the Communication Service to receive the request, analyze the data received in the database and send a proper answer.

The other types of messages would be immediately discarded by the Authentication System, since they require an ID that would not exist in database.

Tests were performed in a computer with the following specifications:

- **CPU** Intel Core i7-4710HQ 2.50GHz
- **Memory** 12.0 Gigabyte
- **Hard-Drive** Solid-State-Drive 50 Gigabyte free memory
- **Operating System** Windows 10 Pro 64-bit

The following variables were used to simulate different user behaviors:

- Number of virtual users (2, 10, 100 and 1000)
- Test time duration (seconds) (always 60 seconds)
- Time delay between each message is sent (milliseconds) (1000 ms and 100 ms)
The following statistics were collected [56]:

- Total number of messages sent
- The number of bytes processed
- Requests per second
- Error rate (percentage of requests that failed)

Every message sent had to meet the following assertions, otherwise they would be considered faulty:

- The message must contain the information that an error has occurred, by having the following text in the body ("hasErrors":true).
- There are two types of errors. The message had to contain the text "User is not pre-registered or does not exists in the system!". It would mean that the Authentication System verified the username and password on the database and analyzed this user is not able to proceed to registration. Otherwise the system would jump to a "catch" code block and would inform that an exception have occurred when trying to read data from database. In this case it would mean the system was unable to access the database and that is considered an error.
- The HTTP code response had to be a valid one (HTTP code 200 means "Ok").
- The response from the server had to be with the time limit of 500 milliseconds. This time was choose based on the observation of response times when developing the system and sending messages to test communication. The average response time was 500 milliseconds.

Tables 5.1 and 5.2 show the results of the tests. Table 5.1 shows the results for a fixed time delay between messages of 1000 milliseconds, a variable number of users and a fixed duration of 60 seconds. Table 5.2 shows the results for a fixed time delay between messages of 100 milliseconds, a variable number of users and a fixed duration of 60 seconds.

Table 5.1: Performance test table for a time delay between messages of 1000 milliseconds

<table>
<thead>
<tr>
<th>Statistics Nr. Users</th>
<th>Total Messages</th>
<th>Average Response Time (ms)</th>
<th>Requests per Second</th>
<th>Error Rate (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>160</td>
<td>3.25</td>
<td>2.65</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>402</td>
<td>2.68</td>
<td>6.69</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>782</td>
<td>14.58</td>
<td>13.02</td>
<td>4</td>
</tr>
<tr>
<td>100</td>
<td>7903</td>
<td>3.16</td>
<td>131.55</td>
<td>21</td>
</tr>
</tbody>
</table>
Table 5.2: Performance test table for a time delay between messages of 100 milliseconds

<table>
<thead>
<tr>
<th>Statistics Nr. Users</th>
<th>Total Messages</th>
<th>Average Response Time (ms)</th>
<th>Requests per Second</th>
<th>Error Rate (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1533</td>
<td>2.65</td>
<td>25.53</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>3798</td>
<td>2.7</td>
<td>63.24</td>
<td>4</td>
</tr>
<tr>
<td>10</td>
<td>7481</td>
<td>4.2</td>
<td>124.6</td>
<td>12</td>
</tr>
<tr>
<td>100</td>
<td>65920</td>
<td>12.48</td>
<td>1097.14</td>
<td>99</td>
</tr>
</tbody>
</table>

The results in the tables demonstrate that by increasing the number of users for a constant time delay between messages, the error rate increases too. And for the same number of users, the number of error increases when the time delay between messages is decreased from 1000ms to 100ms.

The assertion that failed to verify with all faulty messages was always the missing sentence “User is not pre-registered or does not exists in the system!” meaning that the system was unable to access the database with that request. However the system returned an answer. No messages were discarded with an invalid HTTP error, which means that the system accepted and tried to analyze all requests.

It can be concluded that the system can handle an average of 6.69 requests per second without any errors and 25.53 with an error rate of 1%.

It must be noticed that in a real-life scenario the system performs cryptographic operations to verify signatures which takes longer to process the requests. Unfortunately these operations could not be tested. However in a real-life scenario the system hardware specification is better than the one used to perform this test which can result in a better performance even performing cryptographic operations.
6
Conclusion

Contents

6.1 Conclusions ................................................. 73
6.2 System Limitations and Future Work ....................... 74
6.1 Conclusions

BioAuth is a framework developed for second factor authentication, such as authorize online banking transaction. The authentication is performed using biometrics. At the moment only fingerprint biometric is used.

The framework is composed by an application for mobile devices and a server. The mobile application is used to perform second-factor authentications using biometrics. It was implemented using Xamarin Forms. The server was implemented using ASP.NET Web API technology.

The communication between the mobile application and the server are based on FIDO UAF Specification. User authentication is performed by mean of asymmetric cryptographic keys. The communication is performed using REST Web Services, through HTTPS. The BioAuth framework is integrated with BankOnBox, an online home banking system developed by Link Consulting company.

The framework communicates with BankOnBox using SOAP Web Services, through HTTPS. The framework was evaluated in terms of usability, performance, interoperability and security. The evaluation results were satisfactory, since no negative result was obtained.

The technical contributions of this project were:

- The use FIDO UAF protocol for remote authentication using biometrics;
- Using Xamarin Forms to implement the mobile application for Android and iOS;
- Using native secure APIs, from Xamarin, to implement biometric authentication and cryptography operations on Android and iOS;
- Implementation of a server that securely communicates with outside modules using REST and SOAP web services over HTTPS;
- Implementation of a server that stores the client’s cryptographic public key and verifies signatures from the client using that same cryptographic public key with BouncyCastle cryptography library;

This framework comes in an era where mobile devices are being used more than laptops or desktops. It is a different idea, that allows the user to use biometrics, and innovative technology introduced in recent smartphones that provide a high level of security too.

Biometrics makes this system different from other authentication systems since alternative systems authentication is based on passwords or an external authentication device, which are authentication methods easily to be forgotten.

The framework meets all the requirements established and can be integrated into more system in the future since it is a viable authentication alternative.
6.2 System Limitations and Future Work

Currently, BioAuth is only integrated with BankOnBox. In the future BioAuth can integrated other business systems.

The current framework only uses fingerprint as biometric authentication mechanism. In the future the system can be improved by adding new biometric authentication mechanisms, such as face and voice recognition.

BioAuth is only used as a second factor authentication mechanism on BankOnBox. Another improvement is to use BioAuth as first factor authentication mechanism, to perform logins. Like was mentioned by the tested users in usability tests, the BioAuth’s mobile application GUI can be improved too in the future. To improve the security between the User Device and the Authentication Service, the communication protocol can be improved, by adding an extra layer of security, like the one used with BankOnBox (WS-Security using x509 certificates).
Bibliography


