NFC4Sure: Mobile Ticketing System

Diogo André Fonseca Antunes

Thesis to obtain the Master of Science Degree in

Telecommunications and Informatics Engineering

Supervisor: Prof. Carlos Nuno da Cruz Ribeiro
Eng. Nelson Nobre Escravana

Examination Committee
Chairperson: Prof. Paulo Jorge Pires Ferreira
Supervisor: Prof. Carlos Nuno da Cruz Ribeiro
Member of the Committee: Prof. Nuno Miguel Carvalho dos Santos

November 2015
Finishing this thesis was one of the most arduous work I had ever done. Moments when I felt this was a 
waste of my time, but luckily I had family, and friends that made me see otherwise. It is for my dear 
friend Bernardo Figueiredo, and my lovely girlfriend, soon to be wife, Mariana Abreu, that I dedicate all of 
this work and moments of personal growth. Without them I would not be the person I am today. 
Thank you.
Acknowledgments

My first years at Instituto Superior Técnico were moments of despair and confusion, I felt my time was being wasted into something I did not believe in. Luckily after some time I realized instead that I had been given one of the biggest opportunities of my life. The opportunity to study in one of the best universities in the world.

This thesis is the sum of my studies and my work, that could not have been done without the help and guidance of Professor Carlos Ribeiro, the Engineer Nelson Escravana, and my friend Eng. João Pedro Lima who went beyond and further in helping me become a better engineer, a better person, and making this thesis a readable document.

Thank you all for your support.
Resumo

Os sistemas de bilhética utilizados nas redes de transportes públicos evoluíram do uso de contactless smartcards para dispositivos móveis com Near Field Communication e Host Card Emulation. O sistema NFC4sure apresenta-se como uma solução de bilhética que se integra perfeitamente com as tecnologias de bilhética existentes, sem o uso de um elemento de segurança de hardware no telefone móvel e não requer uma ligação on-line permanente. O objetivo deste trabalho é apresentar a arquitetura NFC4sure e implementar um sistema de autenticação implícita capaz de identificar e autenticar o usuário do dispositivo móvel no sistema NFC4sure. Propomos atingir este objectivo através da utilização de um mecanismo que faz uso da interação do utilizador com o écran do dispositivo móvel de forma a criar uma impressão digital distingüível.

Palavras-chave: transportes públicos, sistema de bilhética, autenticação implícita, biometria comportamental
Abstract

Public transportation ticketing systems went from the usage of contactless smartcards to mobile devices using Near Field Communication and Host Card Emulation. NFC4sure is a ticketing solution that integrates seamlessly with existent ticketing technologies, without using a hardware secure element in the mobile phone and not requiring a permanent online interaction. The scope of this thesis is presenting NFC4sure architecture and implementing an implicit authentication system capable of identifying and authenticate a mobile device user into the NFC4sure system. The way we achieve this is by using a mechanism that makes use of the user interaction with a mobile device screen to create a distinguishable fingerprint.

Keywords: public transportation, ticketing system, implicit authentication, behavioral biometrics
Contents

Acknowledgments .......................................................... v
Resumo ................................................................. vii
Abstract ................................................................. ix
List of Tables ............................................................. xv
List of Figures ............................................................ xvii
Glossary ................................................................. xix

1 Introduction ............................................................. 1
  1.1 Motivation ............................................................ 3
  1.2 Objectives ............................................................ 4
  1.3 Organization .......................................................... 5

2 Background and related work ..................................... 7
  2.1 Background ............................................................ 7
  2.2 Related Work .......................................................... 8
    2.2.1 Touchalytics: On the Applicability of Touchscreen Input as a Behavioural Biometric for Continuous Authentication .............................................. 9
    2.2.2 Touch me once and I know it’s you! Implicit Authentication based on Touch Screen Patterns .......................................................... 11
    2.2.3 Secure Unlocking of Mobile Touch Screen Devices by Simple Gestures —You can see it but you can not do it ........................................... 12
    2.2.4 SilentSense .......................................................... 15

3 Fundamental Concepts ............................................. 17
  3.1 Identification and Authentication .............................. 17
  3.2 Explicit Authentication ............................................. 17
  3.3 Implicit Authentication ............................................. 17
  3.4 Support Vector Machines ........................................... 18
  3.5 STRIDE .............................................................. 20
  3.6 Risk Treatment ...................................................... 21
    3.6.1 Throttling ......................................................... 21
    3.6.2 Hardening ......................................................... 21
## 8 Conclusions

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.1 Limitations</td>
<td>65</td>
</tr>
<tr>
<td>8.2 Future Work</td>
<td>66</td>
</tr>
<tr>
<td>8.3 Conclusion</td>
<td>66</td>
</tr>
</tbody>
</table>

## References

67
List of Tables

3.1 Properties violated in STRIDE. ....................................................... 21

7.1 First application test results (Best Case). ........................................ 59
7.2 First application test results (Worst Case). ...................................... 60
7.3 First application test results (Worst Case - 5 Classification Swipes). .... 60
7.4 Second application test results (5 users - 3 Classification Swipes). ........ 61
7.5 Second application test results (5 users - 4 Classification Swipes). ........ 61
7.6 Second application test results (5 users - 8 Classification Swipes). ........ 61
7.7 Second application test results (10 users - 3 Classification Swipes). ...... 62
7.8 Second application test results (10 users - 4 Classification Swipes). ...... 62
7.9 Second application test results (10 users - 8 Classification Swipes). ...... 62
7.10 Training delay. ............................................................................. 62
List of Figures

2.1 List of extracted features and their mutual information. ........................................ 10
2.2 The Equal Error Rate (EER) [20] ................................................................. 10
2.3 Accuracy Formula. ......................................................................................... 11
2.4 Unlock Screen Patterns. .................................................................................. 12
2.5 Accuracy Results. ......................................................................................... 12
2.6 Top 10 most effective gestures. The number of unconnected arrow’s represent the number of fingers needed to perform the gesture. ............................................. 14
2.7 Model overview ......................................................................................... 15
3.1 Classification (linear separable case) [35]. ..................................................... 19
3.2 Non-linear separable case. .............................................................................. 19
3.3 Hyperplane projection. ................................................................................. 20
4.1 Various elements involved in the ticketing architecture. .................................. 25
4.2 Sequence of protocols run by the general ticketing architecture. ................... 27
4.3 Register user in the ticketing system. ............................................................ 27
4.4 Protocol used to recharge the user VirtualCard. ............................................ 28
4.5 Reconciliation protocol with the Validator’s and the Back Office. ..................... 29
4.6 VirtualCard structure. ................................................................................... 31
4.7 Detailed overview of the scenario architecture with the VirtualCard Integration Layer. ... 32
4.8 Register user in the cloud service. ................................................................. 34
4.9 Recharge Protocol. ....................................................................................... 35
4.10 Loading tokens to the user mobile device. ..................................................... 37
4.11 Token validation protocol. ............................................................................ 38
4.12 Second validation protocol using the same token. ......................................... 40
5.1 Elements involved in the system and their connections. .................................. 42
6.1 First application initial screen. ................................................................. 53
6.2 First application second screen. ............................................................... 53
6.3 First application gallery. ............................................................................. 53
6.4 First application text. ................................................................................... 53
6.5 Final application initial screen. ................................. 54
6.6 Final application shopping screen. ............................... 54
6.7 Final application payment mode. ............................... 55
6.8 Final application card selection. ............................... 55
Acronyms

Dos  Denial of Service.

ERR  Equal Error Rate.

FAR  False Acceptance Rate.
FN   False Negatives.
FP   False Positives.
FRR  False Rejection Rate.

HCE  Host Card Emulation.
HSE  Hardware Secure Element.

ISP  Internet Service Provider.
KNN  K-nearest-neighbors.

MNO  Mobile Network Operators.
NFC  Near Field Communication.

PAN  Primary Account Number.
PIN  Personal Identification Number.
PTO  Public Transportation Operator.

RTT  Round-Trip-Time.

SAM  Master Secure Access Module.
SE   Secure Element.
SLA  Service-Level Agreement.
SVM  Support Vector Machines.
TAL Technology Adaptation Layer.
TN True Negatives.
TP True Positives.

UICC Universal Integrated Circuit Card.

VCIL VirtualCard Integration Layer.
VC SL VirtualCard Security Layer.
VRL Validator Reader Layer.
Chapter 1

Introduction

Most public transportation services (metro, train, etc.) feature electronic ticketing systems to handle both the payment for the services by the customers and the management of the transport infrastructure. Smartcards are on the user’s side of the electronic ticketing systems, enabling him to use the infrastructure. However, while smartcards were once a convenience for many people, they are now becoming a burden, both in terms of the financial cost of acquiring a smartcard, but also due to the simple practical fact that smartcards do pileup in people’s wallets.

Consumer acceptance is one of the key factors driving technology evolution. In the case of smartcards, one of the key points that lead to its acceptance was the consumer’s perception that they were secure. However, smartcards are actually vulnerable to several types of attacks, such as differential power analysis [32], timing attacks [31], design or implementation flaws, and many others [22].

To increase the usability and security of the ticketing systems, companies turned their attention to a device that most people have and use on their daily lives, the smartphone. The number of people that own a smartphone is increasing at an astonishing rate. In 2014 there were about 1.9 billion smartphone users, and according to Ericsson report \(^1\) in the next five years this number will expand to 5.9 billions. Smartphone users are sharing everyday more and more private information without even knowing through applications that tell their location, have access to their contacts, or in more extreme cases have access to privileged data, for instance their bank credentials.

As this happens, many companies around the world invest in making their business or product that is available through a smartphone, more secure. This security may come in different flavors, such as authenticating the correct user, storing less sensitive information (token), and having a Secure Element (SE) to safely store the data. This SE is usually a chipset that provides an extra layer of protection, by creating a secure and isolated environment from the main memory of the smartphone.

In ticketing systems the communication usually occurs via Near Field Communication (NFC) \(^2\). NFC is a short-distance radio communication technology, which enables bidirectional communication between two devices. This communication channel is established by tapping or bringing the devices to close proximity (approx. 10 cm).

Typical smartphone NFC-enabled transactions (e.g. enter a public transportation service) usually require the existence of an Hardware Secure Element (HSE), containing the necessary credentials. One example of such hardware SEs is the Universal Integrated Circuit Card (UICC) (also know as SIM card), provided by Mobile Network Operators (MNO). When operating with an hardware SE to perform a transaction, the user holds the device over an NFC terminal which communicates with the device’s NFC controller, which is responsible for routing all the data received from the NFC terminal directly to the device’s SE.

Despite the security guarantees offered, the solutions relying on an hardware SE provided by MNO requires a certain degree of cooperation between companies and institutions. For instance, for a Public Transportation Operator (PTO) to charge a customer for a ticket, it has to interact with the MNO for the transaction data to reach the SE. Considering UICCS are supported and operated by MNOS this service might be subjected to additional fees. Putting all this together we have a power game for who controls the HSE, being the major financial institutions, the telecommunications companies, or the manufactures of the mobile devices. This leads to a necessary cooperation between organizations and companies that sometimes its very hard do accommodate.

To address some of the issues posed by NFC, and the use of HSE, Host Card Emulation (HCE) was introduced. HCE emulates a virtual smartcard into an application, enabling it to communicate with the NFC reader and bypassing the SE. HCE was first introduced in Android 4.4 (KitKat), by the creators of SimplyTapp in 2012 [7]. HCE allows the payment application to communicate directly with the NFC controller, and store the payment credentials on the application, instead of on the SE. When a user wants to perform a transaction using NFC, the NFC Reader (i.e. the payment machine) communicates with the NFC controller in the device, and instead of routing the data to the SE, it now sends the data directly to the HCE application. The application must implement the security mechanisms necessary to support the transaction. In many cases the SE is moved to a cloud server, and is now named remote SE. When a user wants to make a transaction using NFC with HCE, the application pulls the data from the remote SE, and passes it from the smartphone to the NFC terminal.

HCE main advantage is removing the constraints of having to negotiate with several financial institutions or network operators, in order to create a payment application. This has several positive outcomes as, for instance, reducing the production cost and time, and simplifying the application’s provisioning. Another HCE advantage is the ability to emulate the same operations performed by a contactless smart card. This is an important point, because it enables backward compatibility with existing technology.

However, most developed HCE solutions come at a cost. In most cases HCE security is paired with a remote SE, which stores the payment credentials, introducing non-negligible usability problems. One of the biggest problems occurs when a user wants to perform a transaction and the HCE application needs to contact the remote SE. This degrades user experience, if we consider the delays caused by network latency, typically in the order of hundreds on milliseconds, which are not compatible with NFC few milliseconds transaction time.

The two biggest problems when using HCE is handling the communication of privileged data where there is always a necessity of verifying the owner of the data, or by other words, identify and authenticate
1.1 Motivation

This thesis is driven by the motivation that ticketing systems using NFC with HCE technology are not secure. Using HCE a mobile device can emulate the functionality of a smartcard but exhibits a number of new security threats.

1. Insecure Storage: While the file system in a smartcard restricts the access to the data, the storage of a mobile device can be accessed by anyone that controls the operating system.

2. Relay Attack: Relay attacks occur when the attacker is able to position himself between the user secure device and the public ticketing infrastructure. Relay attacks on smartcards require the attacker to control or impersonate part of the public ticketing infrastructure (e.g. a validator) [23]. Other relay attacks on HCE mobile devices requires only the takeover of the operating system of the mobile device [19].

3. Insecure Communication: On NFC communication occurs through the air via radio frequency, allowing eavesdropping or man-in-the-middle attacks.

4. Denial of Service (Dos) Attack: Smartcards and mobile devices can be damaged or stolen. But mobile devices can also have the service disrupted by a rogue application, altering the NFC route table, and many other forms.

5. Privacy leakage: In HCE if the device suffers an eavesdropping attack, the attacker may gain private information without the user knowledge.

There is a necessity to create a usable and secure system, which can be defined as system that is effective, efficient and provides satisfaction to the users. To be able to deliver such capabilities and functionalities, ticketing systems employ tokenization strategies.

The objective of tokenization techniques is to reduce the amount of sensitive information both kept on the mobile device as well as exchanged between the parties involved.

But even applying tokenization it is still needed to properly authenticate the user. When it comes to authenticate a user, there are several factors in which we rely on to assure this process occurs in a secure way:

- Something you know: The authentication method that most of the people use on a daily basis, for instance, a password to access their bank account, or smartphone.

- Something you have: In conjunction with something you know, something you have helps on the authentication process, for instance a smart card, or a SE.

- Something you are: Based on the variety of intrinsic characteristics in a person, the way a person is. Your fingerprint, your eyes, the way you move or talk.
If we store our SE in the cloud, the something you have factor disappears, and we are left with the other two, something a person knows, and something they are. The most common authentication method, in mobile devices, comes from the use of a username and password. Then there is lock-pattern and Personal Identification Number (PIN). While this methods dominate the authentication scheme today, they open a vast list of problems. From a security perspective, it is proven that lock-patterns are easily bypassed, in many cases, by a Smudge Attack [3]. Since most of the time smartphone users just perform quick actions on their smartphones, for instance reading a text or sending an email, the use of a password is in some ways inconvenient for the user, who has to remember long and complicated passwords, just to perform a quick action on his smartphone. To circumvent this discomfort, users employ weak and easily guessable passwords, or even disable the password completely. In order to alleviate the problem posed in this methods, comes implicit authentication.

First we need to understand that implicit authentication makes use of biometric features (something you are), and those are distinctive and measurable characteristics used to identify a person. This is quite an active research area, where there are studies that range from identifying a user through smartphone hardware differences [15, 36], to recognize a user through his biometric features [15, 38, 14, 20].

Considering people stay a far amount of time with the same smartphone, if we can distinguish smartphones from one another, even same model smartphones, it is safe to assume that if you are in legit possession of a certain smartphone, you can be identified and authenticated. Now this comes to contradict certain things said on this document, for instance, many times users disable passwords, or use weak ones, and even if they use lock-patterns, they are easily bypassed. Not to mention the amount of smartphones that are stolen\(^3\), roughly 4.5 million on the United States (US).

So how can we authenticate a user in a secure way? The answer is by combining authentication systems. We could imagine this by thinking, if instead of just being authenticated by a username and a password, the system passively collects user data, computes biometric features and recognizes you.

Although the integration of NFC technology with HCE in ticketing systems is an important motivation, this thesis focus on the analysis and implementation of an authentication process for the users in the ticketing system.

1.2 Objectives

Our solution applies HCE to the public transportation ticketing system, with the aim of providing a secure ticketing system, while not requiring the existence of a permanent online interaction with a centrally stored SE. This is achieved by creating a VirtualCard solution, which combines tokenization techniques with cryptographic mechanisms to ensure not only that the access to ticket information requires the ticketing system collaboration, but also that fraudulent usage attempt will be promptly detected. Moreover, our solution is expected to have a light footprint on currently deployed ticketing infrastructures, integrating seamlessly with existent ticketing technologies.

The scope of this project is substantially vast, in this thesis we purpose implementing one authentication method used in section 4.2.3 that allows the identification and authentication of the user. The mechanism chosen is an implicit authentication method, based on the user normal interaction with a mobile device. After a thorough research in the area we settled on the work done by Mario Frank et Al, Touchalytics [20]. We explain this decision meticulously in section 6.

With the work done the main question this thesis tries to answer:

- How can a user be authenticated in a mobile device using a non intrusive technique that explores the normal usage of the device?

### 1.3 Organization

This document is organized as follows. First, chapter 2 lays some background work done in the area of ticketing systems and complement it with related work directed to the scope of this thesis, implicit authentication in mobile devices. Following that we present some fundamental concepts (section 3) in order to understand the scope of the work more clearly. Next the architecture of the system is presented (section 4) which is divided into the reference ticketing architecture (section 4.1) and the proposed solution (section 4.2) to support the migration to HCE. On chapter 5 we present an analysis on the limitations of the proposed solution, along with possible mitigations. Section 6 is reserved to the implementation details of Touchalytics in this architecture. Finally on chapter 7 an analysis of the results is performed, and on chapter 8 we present some conclusions and possible future work.
Chapter 2

Background and related work

In this chapter we start by explaining some ticketing solutions that make use of HCE. Later we introduce Apple Pay and their tokenization strategy which is an important element of our solution. Finally in section 2.2 we converge to the main scope of our project, and lay out important work in the field of implicit authentication using mobile devices.

2.1 Background

Several payment systems have already been proposed with the intent of exploiting the ease of use of smartphone’s NFC technologies. Some of these solutions rely on HCE, while others, supported by very large market players, rely on other proprietary technologies.

SimplyTapp [43] was the first HCE solution, introduced in the CyanogenMod 9 [12] in conjunction with Android 4.4 (KitKat) to release the constraints in the NFC payments community. After that many different products using HCE have arisen. Cuscal ¹ has a mobile payments solution that makes use of HCE. The company developed a program called “CUA redi2PAY”, available at Google Play Store. “redi2PAY” allows clients to tap their phones on a POS payment terminal and use it to process a payment. Bell ID develops software solutions that allows issuers to perform end-to-end mobile cloud payments using HCE [26].

Google also provides an HCE solution, which stores the encrypted credentials, necessary for a transaction, in a remote secure server, managed by Google, that also keeps the credit card information linked with those credentials. Which means that every transaction goes through Google, which is a clear disadvantage in terms of privacy, compared to other solutions. With Google’s HCE solution the user authenticates himself for every transaction by inserting a 4 digit pin, then the user’s device communicates, via NFC, to the merchant POS and sends the transaction data to Google. Once on Google, the data is streamed to the payment service network, to create a payment token, which is then redirected to the mobile device and into the POS. If the token is valid the transaction is authorized.

In the domain of public transportation ticketing systems several HCE systems have been proposed. However, most of these systems are proprietary, and lack a comprehensive, community-accepted secu-

rity analysis. For instance, Bytemark\(^2\), developed an innovative mobile ticketing solution for transports, attractions, and events using a visual aid in fare control [27]. Contrarily to other companies, they do not offer a standard product, but yet a customizable solution, that range from trip planning, to securing tickets in a cloud system which may be shared with other users’ devices.

A Spanish company Aditium launched an application called TickTrack \(^3\), which is another mobile ticketing solution, but employs geolocation and other metrics to provide better security, in order to solve the problem of the device authentication.

Most of the above described solutions adopt a tokenization strategy to overcome the problem of network latency, although most of these are closed proprietary solutions that lack documentation for a proper security analysis.

Apple Pay \(^4\) provided by Apple Inc. aims at providing a mobile payment platform. The service leverages on HSE to store the encrypted credit card information, on the Touch-ID to read the user fingerprint, and on an NFC antenna to communicate with the payment terminal. Apple in conjunction with MasterCard and Visa provide compatibility between their service and major merchants.

Apple Pay applies a payment tokenization strategy [16]. When the user adds a credit card via iTunes or the Facetime camera, a cryptogram and a token (represented by a dynamic 16-digit number) \([29]\) are generated by the payment service network (e.g. MasterCard). Both the token and the cryptogram are sent to the consumer mobile device via NFC, and stored in a HSE. During a transaction between the consumer and the merchant, the consumer authenticates using the fingerprint scanner, Touch-ID. This action sends the cryptogram and the token to the Merchant. The cryptogram is decrypted by the merchant which verifies its integrity. If valid the merchant sends the token to the payment service network, which decrypts it, converts the token to the original Primary Account Number (PAN), and verifies its integrity. When the token is due valid, the transaction amount is debited from the consumer’s account and credited to the merchant's account.

Apple outsources payment services to a payment service network. For this service to work, MasterCard [8] and Visa [9], had to update their infrastructure.

### 2.2 Related Work

In order to better understand the scope of this project and the work done in this area, we provide an insight of the related work. We selected a set of papers that create an implicit authentication system. First we analyse Touchalytics [20], in second the work done by Alexander De Luca et al [14], then a system called GEAT [1], and in last SilentSense [4].

\(^2\)https://www.bytemark.co/
\(^3\)http://ticktrack.aditium.com/
\(^4\)https://www.apple.com/apple-pay/
2.2.1 Touchalytics: On the Applicability of Touchscreen Input as a Behavioural Biometric for Continuous Authentication

Touchalytics [20] is a system built to continuously authenticate users based on the way they use their smartphone screen. They accomplish this by extracting touch data, computing a set of 30 biometric features, and with the help of a classifier authenticate the user. Two types of classifiers were used, K-nearest-neighbors (KNN) [11] and a Support Vector Machines (SVM) with and rbf-kernel. This system is built on the premise that the data retrieved from the movements of the user on the touchscreen, is sufficiently different to serve as behavioural biometric.

Android and iOS prohibits touch data to be read from one application to another i.e. touch data recorded from one application is only accessible on that application and no other. For the extraction of touch data they built an Android application with the possibility of browsing a set of images, and a set of text documents. An experiment was conducted with 41 users and 4 different type of android smartphones, where the user’s were told to navigate in the application by reading text, and viewing images.

The touch data is collected using the Android API, when the user performs triggers actions, such as sliding horizontally (viewing images), or vertically (reading text). These movements are designated as a stroke, and consist of a vector recorded with the following information:

- Position x;
- Position y;
- Time Stamp;
- Pressure on the screen;
- Area occupied by the finger on the screen;
- Orientation of the finger;
- Orientation of the phone.

After collecting this data a set of biometric features were computed. This features are displayed in Figure 2.1, in conjunction with their mutual information, which is an informative analysis about their relevance.

The training phase consists of normalising the feature data, calculating the parameters of the classifier, and building a classifier for each of the different users. During this phase, the system is secured using a different method, for instance, username/password. The authentication phase occurs when the classifier is trained. During this phase the system logs all the user touch-data and tries to authenticate the user. The authors evaluated their system using a group of metrics:

- False Acceptance Rate (FAR): The fraction of strokes from imposters that are recognised as legit by the system.
Figure 2.1: List of extracted features and their mutual information.

- **False Rejection Rate (FRR):** The fraction of strokes of legit users that are rejected by the system.
- **Equal Error Rate (ERR):** Is the rate at which the FAR and FRR are equal.

Their approach was able to achieve a Equal-Error-Rate (EER) of $\leq 3\%$ by using a sample size of 13 strokes, as exemplified in Figure 2.2.

![Equal Error Rate (EER)](image)

Figure 2.2: The Equal Error Rate (EER) [20]

Overall the system makes an effort to achieve authentication without explicit user interaction, although the user has to interact with the android application, one can imagine the use of such application where the movements on the screen occur voluntarily.

In conclusion their results suggest that it is possible to distinguish users based on biometric features, calculated through stroke data. Touchalytics can be used by itself as an authentication system, but has some limitations. It needs a training phase, which means that a new user can not be authenticated right
from the start. During the training phase the system would have to depend on another authentication mechanism, for instance username/password.

2.2.2 Touch me once and I know it’s you! Implicit Authentication based on Touch Screen Patterns

Touch me once and i know it’s you [14] was inspired by lock-screens in the Android mobile operating system. Using an implicit authentication approach and the user interaction on the smartphone touch-screen, the authors extract a set of behavioral biometric features capable of recognising different users. Their motivation comes from a security point of view, where lock-screens are very easy to bypass, for instance using Smudge-Attacks [3], or shoulder-surfing.

Their first procedure consisted in developing an Android application with four different unlock screens, two of them horizontal and vertical requiring one finger, and the other two using specific patterns that permit more accurate extraction of data. One of the unlock screens required two fingers in order to unlock. The data extract was:

- Pressure, how hard the finger pressed the screen
- Size, the area of the finger on the screen
- X and Y Coordinates
- Time at which the extraction occurred

The authors tested their system by unlocking the phone 160 times with each of the four unlock screens, during the period of two days. 48 participants were recruited, and the smartphone used was an Android Nexus One. The quantity of data gathered summed to 30,720 unlocks (640 for each participant).

The evaluation of the data was made using Dynamic time warping [37], an algorithm that looks at different sets of data and calculates their similarity. As a reference set, to identify a user, they use the first 20 unlocks made. When using DTW to compare different users reference set’s, it is possible to calculate the True Positives (TP), legit users accepted, True Negatives (TN), non-legit users correctly rejected, False Positives (FP), non-legit users accepted, and False Negatives (FN), legit users falsely rejected. Using the formula represented in Figure 2.3 and the four different unlock screens, they calculate the accuracy of the system.

\[
\text{Accuracy} = \frac{\sum_{i}TN + \sum_{i}TP}{\sum_{i}TN + \sum_{i}TP + \sum_{i}FN + \sum_{i}FP}
\]

Figure 2.3: Accuracy Formula.

In this first experiment the accuracy values were quite low, reaching a minimum of 38% in some cases.
After evaluating the first system, they proposed a second one, that tries to emulate a real world usage, and achieve better results. The second system consisted of an Android application, that was sent to the participants via e-mail and also the respective unlock screen pattern they had to introduce. An example of such patterns is represented in Figure 2.4.

![Unlock Screen Patterns](image)

The application had two modes, a training mode, for the users to train the unlock pattern, and a study mode, where just one authentication per day was allowed, or three failed attempts. The study mode was conducted during a period of 21 days, and used 31 participants. After the 21 days had passed, the users went to a meeting and handover their smartphone for extraction of the data. During the meeting another application was installed on their smartphone, where they had to input the unlock patterns of the other users 3 times. This data was used to simulate possible attacks. The data evaluation was conducted like the first study, using DTW, and the data extracted was the same with the addition of the speed parameter. The reference set of each user, was built using the first five correct authentications. In order to escape the hardware differences of several Android phones of the participants, they only compared users with the same smartphone. The results are visible in Figure 2.5.

![Figure 2.5: Accuracy Results.](image)

<table>
<thead>
<tr>
<th>True Positives</th>
<th>False Negatives</th>
<th>True Negatives</th>
<th>False Positives</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>398</td>
<td>92</td>
<td>852</td>
<td>231</td>
<td>77%</td>
</tr>
</tbody>
</table>

False Rejection Rate: 19%  
False Acceptance Rate: 21%

In brief they propose a system capable of offering an invisible extra layer of protection on top of unlock screens, extracting touch data, and using DTW to measure the similarity between users. In a first time study they reach a minimum accuracy of 38%, but after trying a different approach and simulating a real world usage, the values of the accuracy reached a minimum of 77%.

### 2.2.3 Secure Unlocking of Mobile Touch Screen Devices by Simple Gestures

—You can see it but you can not do it

In this paper a system called GEAT [1] is proposed. GEAT is a gesture based user authentication scheme for secure smartphone unlocking, implemented on a Samsung Focus smartphone, using Win-
dows Mobile operating system. Instead of just using a typical unlock screen, where the user has to input a defined pattern to gain access to the smartphone, they add an extra layer of security that also checks how the user inputs this patterns. They claim that is not possible to perform a smudge attack or shoulder surfing on the system, because the extracted data from the user movements on the smartphone screen is unique to each user, or by other words, biometric.

In order to test GEAT, they conducted a study with 50 participants, where every time a user wanted to unlock his smartphone, first he had to perform a gesture and then input is PIN or unlock pattern (if enabled), if both actions were successful the data collected during the gesture input was recorded as a training sample. The training phase endured until 15 to 25 samples were collected. The training procedure summed to a total of 15009 gesture samples, across all participants. The set of data extracted from interaction with GEAT resulted in a computation of the following features:

- Velocity magnitude: speed of the finger at different time instants.
- Device acceleration: the acceleration of movement in the touch screen.
- Stroke time: the time duration a user needed to complete a stroke.
- Inter-stroke time: the time duration between the starting time of two consecutive strokes for multi-finger gestures.
- Stroke displacement magnitude: euclidean distance between the centre of the bounding boxes of two strokes for multi-finger gestures.
- Stroke displacement direction: the direction of the line connecting the centres of the bounding boxes of two strokes for multi-finger gestures.
- Velocity direction: the direction of finger motion at different time instants.

In addition to the features extracted, there was also accelerometer data collected for each gesture. Each stroke was then divided into sub-strokes using an algorithm capable of selecting the appropriate time duration for each sub-stroke, that maintains the user intrinsic behavior. This happens because the features above are calculated taking into account the whole stroke duration or the sub-stroke duration. The stroke features are, stroke time, inter-stroke time, stroke displacement magnitude, and stroke displacement direction. The sub-strokes features are, velocity magnitude, velocity direction, and device acceleration.

From time to time users perform the same gesture in a different way, this could lead to a high level of discrepancy between samples of the same user. In order to solve this problem they calculate a threshold for each distinct posture, and add this information to each gesture. Finally to remove the noise introduced by smartphones low screen resolution, they apply a low pass filter.

A set of 39 gestures were created to help extract relevant biometric information capable of authenticating a user, and after a thorough evaluation 10 gestures were selected. The top 10 most effective gestures are displayed in Figure 2.6.

---

5 bounding box of a stroke is the smallest rectangle that completely contains that stroke.
During the training phase they built a Support Vector Distribution Estimation (SVDE), or Support Vector Machine (SVM) classifier for each user using only samples from the legit user. Also during training phase the user may choose how many gestures he wants to use from the top 10, bearing in mind that higher the number of gestures, the higher the accuracy of the system. The user can also report how many different postures he used during the input of those gestures, for instance laying down or sitting. They also rank each gesture performed, based on their EER. To calculate the EER for each gesture they need to have the values of the True Positives Rates (TPR) and False Positives Rates (FPR). Since they only use legit user samples on the training phase, and for the FPR they need un-legit user samples, they way they obtain this is by deforming the legit user samples just enough so that it can pass has a possible attacker sample. For the TPR they use 10 fold cross validation, which consists in breaking the training data into 10 sets of size \( n/10 \) where \( n \) is the number of samples in the training set. Then train on the 9 datasets and test on 1. Repeat this step 10 times and calculate the mean accuracy.

On the authentication phase, the user is prompted with the top 10 gestures to perform, chooses how many he desires, and replicates the gesture. GEAT assembles the feature vector, feeds it to the SVDE and gets a classification decision for each different gesture. If the decision is positive, which means that the variation from the gesture performed in the training phase is minimum, then the user is accepted as legitimate, otherwise it is classified as un-legit, for that gesture. After performing each gesture selected in the login screen, the final decision is done using a majority vote system of the classification decisions.

The authors reported several experimental results on different scenarios. An accuracy evaluation when the user has performed the gestures using only one posture \( (n = 1) \), with accelerometer data or without, which reported an average of 4.8% EER. If the number os postures increased to \( n = 3 \), the average of EER decreased to 1.7%. Finally they evaluate the sized of the training sample and their impact on the EER. Using only one gesture and 25 training samples, they report 3.2% EER. If the number of gestures increases to three and the training samples remain 25 per gesture (65 total), they reach 0.5% EER.

In brief GEAT offers an extra layer of security when unlocking smartphones by calculating biometric features capable of distinguish different users. They report an average equal error rate (EER) of 0.5% with 3 gestures and using 25 training samples.
2.2.4 SilentSense

SilentSense [4], presents an implicit authentication scheme capable of authenticating users in a transparent, secure way. By extracting a set of touch-based features from the user interaction with the smartphone, and combining it with sensors inside the same device, they are capable of creating a touch-based biometric fingerprint of the user.

SilentSense motivation comes from a privacy, security and usability perspective, since most authentication mechanisms are easily bypassed, a possible solution can be achieved by using a more enhanced password pattern, since the one used in [14]. Such methods can be inconvenient for the average user and reduce the overall experience and usability of using a personal smartphone. Also more and more users share their device with a friend, or family, as a result private data may be disclosed. As so, SilentSense proposes a system that is able to identify when a guest user is using the device, and activate a privacy protection mechanism.

The system built is exemplified in Figure 2.7.

![Figure 2.7: Model overview](image)

In Figure 2.7 we can observe that the system consists in a training and identification phase. On the training phase, a set of biometric features are extracted:

- Pressure on the screen.
- Area occupied by the finger on the screen.
- Duration of the movement on the screen.
- Position of the finger in the screen.

These features are extracted during 4 different gestures, fling, scroll, tap and multi-touch. Also during the extraction these features are combined with the sensor data from accelerometer and gyroscope. Such sensor data results from a corresponding reaction to the interaction with the smartphone. The combination of all those features is used in conjunction with Support Vector Machines (SVM), to build a classification model. When the user is walking the data extracted from the sensors is skewed with the normal interaction with the smartphone. In order to deal with this problem, the authors calculate an extra set of features:

- Vertical displacement of each step.
• Current step frequency.
• Mean horizontal acceleration for each step.
• Standard deviation of vertical acceleration for each step.

To test the performance of the system, they built a service running on the background of an Android phone, that extracts the needed data from sensitive applications. An example of the applications considered sensitive are, message, mail, album, contacts, and social networks. They used the data generated from 100 volunteers, where 10 are classified as legit owners of the phone, and the other 90 guest users. In a first scenario they test the usage of SilentSense in a static environment, where the user is using the device without walking. They report the number of observations needed to accurately identify a guest and a legit user. In order to identify the legit user, after 8 observations the accuracy of the system is nearly 100%, but in the case of recognizing a guest the accuracy is around 80% within 10 observations. In a second scenario the evaluation consists of a dynamic set-up, where the user interacts with the smartphone while walking. In this case they use 50 volunteers using the device freely, and extract the extra set of features built specifically for when the user is walking. They report a false acceptance rate (FAR), of nearly 0% after 3 steps. When it comes to the accuracy of detecting a guest user or the legit user, the values are, around 100% after 12 steps when detecting a guest, and 100% after 7 steps for the legit user.

In conclusion SilentSense offers a secure, invisible touch-base biometric system, capable of authenticating a legit user, or identifying a guest. This information can be used to activate a privacy mechanism when a guest is present, or treat every guest as a possible attacker, and deny access. They report accuracy values for detection of legit and guests of nearly 100% after a few observations.
Chapter 3

Fundamental Concepts

In this chapter we lay down some concepts that we think to be crucial for a better grasp of the project. First we start by mentioning some of the authentication techniques used today, and their drawbacks. In second place we introduce the notion of SVM, a part of machine learning used in this project. STRIDE (section 3.5) is presented as a threat modeling approach used to evaluate the system. Finally in section 3.6 various concepts applied to our system in order to solve and mitigate security issues.

3.1 Identification and Authentication

As described in [21], Identification is the way a user provides is identity to the system. Authentication is the process where the identity of the user is validated. This can be exemplified by a simple username/password, i.e. identification could be the username Bob and the authentication Bob’s password.

3.2 Explicit Authentication

There are many ways to authenticate a user. One of the most used consists of the user inputing a known secret, typically a username/password, and if it is correct the user is authenticated. This form of authentication is commonly referred to as explicit or static authentication. The main problem that arise from this form of authentication, is the fact that once an attacker knows your username and password, he can gain access (if no more layers of authentication are applied) to private data. While this is a very used technique, the strength of this method lies on the difficulty of guessing the password.

3.3 Implicit Authentication

The notion of Implicit Authentication [28] applied on mobile users states that it is possible to authenticate a user based on actions that he already performs on a daily basis. It does so by observing user behaviour.
Implicit Authentication can be used to add an extra layer of security to an existing mechanism, for instance it can offer an augment password protection, by extracting behaviour patterns of the user. It can also be used as a primary authentication system, taking the need to enter a password altogether, and reaching higher usability. Or in a more sensitive area, implicit authentication can be used to prevent fraudulent credit transactions in a mobile device, verifying with a certain degree of certainty if the user is who he claims to be during the transaction.

3.4 Support Vector Machines

Support Vector Machines or kernel machines [10], are a part of supervised learning models in Machine Learning. SVM are used to perform linear classification and regression analysis. For the scope of this project we are going to focus on the linear classification part. In linear classification there are two phases, a learning phase, and a classification phase.

In the learning phase the system develops a classification model capable of classifying random input data. Or by other words, the system receives a training set, normally a vector of features, and their respective label, then the system recognises patterns in the data, and builds a classification model. Imagine we want to classify different people, we can receive a vector containing a group of distinctive features from that person, for instance, their eye colour, their size, weight, health problems, body mass, and we also receive the information that this vector of features belongs to the person named Bob. With this information the SVM algorithm can generate a classification model, with the ability to classify new people.

In the classification phase the system looks at new data without label. Following the person example, without the person name attached, just the vector of features. Using the classification model, generated in the learning phase, the system can dictate if this new vector belongs to a known person or not.

SVM [30] came to solve several problems of supervised learning, one of them called Bias-variance tradeoff. Bias is when after training the kernel machine with any of the different training sets, we always predict the same incorrect value for the output of \( x \). If that happens we can say that the system is bias for the input \( x \).

Variance happens when the kernel machine is trained with the different training sets, and predicts different values for the output of \( x \). In that case, we say the system has high variance for the input \( x \).

The idea is to have the so called tradeoff between bias and variance, present in the system. We want a system that can capture the similarities in the data, but can also adapt to unseen data. Basically a machine that can learn a training set without error. SVM solves this problem by introducing a bias/variance tradeoff variable that can be changed to accommodate the variances in the training set.

SVM is built to solve linearly separable binary classification problems. This means solving problems where only two classes exist. This is done by choosing the best separating hyperplane \(^1\) that maximises the margin between the two classes. An hyperplane is a subspace with one dimension less of the actual plane. Which means if we are at a plane with 2-dimensions, our hyperplane is a plane of 1-dimension.

\(^1\)http://mathworld.wolfram.com/Hyperplane.html
or in this case 1-dimensional lines.

The points of data that fall on the boundaries of the margin are called Support Vectors. If we look at Figure 3.1, we can observe the two classes, the blue and black dots, the hyperplane, and the support vectors that lay on the margin.

There are times when a problem cannot be linearly separable in the actual dimensional space. Observing Figure 3.2 it is impossible to draw a hyperplane that can divide the two types of classes. What is done in this situations is projecting the problem into a higher dimensional plane, Figure 3.3 demonstrates this method. This projection is called Hilbert-Schmidt Kernel, or sometimes referred to as the “kernel trick”[41].

So far what we have seen is an algorithm capable of solving two-class problems. But if we think about our example, where we have a vector containing features of one specific person, we gonna end up with a lot of different vectors, and consequently many different persons also. Which make’s this a multi-class problem. In order to solve this, SVM reduces the multi-class to several binary problems. Many methods emerged to provide this extension to SVM, one-against-all, one-against-one and directed acyclic graph (DAG) methods [25].

One-against-all strategy commences by training a single classifier for each class, where the same number of positive and negative samples are provided. Picking up our Bob example, in this situation we
would train the classifier with $x$ vectors containing Bob features (positive samples), and $y$ vectors of all the other users (negative samples), where $x = y$. In this strategy the classifier instead of just predicting the label, also need to output a confidence level. This happens because on the classification phase the unclassified vector of features is going to be applied to each classifier built, and the one with the highest confidence score is selected. This may pose some problems, for instance the class distribution on the training set may not be evenly distributed.

In the one-against-one strategy, if we have $x$ classes, $\binom{x}{2}$ binary classifiers are trained. Which means, each classifier separates a pair of classes, that it knows how to distinguish. In the moment of the decision a majority vote is taking into account, and the classifier that got the highest votes, get’s the final decision. In comparison this method only involves a small subset of the dataset per classifier, whereas the one-against-all technique requires the entire dataset.

![Hyperplane projection](image)

Figure 3.3: Hyperplane projection.

In summary, SVM solves linearly separable binary classification problems, but can also be extended to include multi-class and non-linear problems, through the use of a kernel function and methods like, one-against-all, one-against-one and DAG.

### 3.5 STRIDE

STRIDE is a model-based threat modeling technique developed by Microsoft [24]. STRIDE is an acronym that stands for [2]:

- **Spoofing**: Impersonating a legitimate user;
- **Tampering**: Modifying application resources, such as, memory data, without permission;
- **Repudiation**: Claiming that a certain action within the system was not responsibility of a certain user;
- **Information Disclosure**: Accessing information without authorization;
- **Denial of Service**: Absorbing necessary resources to maintain the system, therefore denying the service to others;
• Elevation of Privilege: Obtaining authorization to resources that are normally protected.

In a certain sense STRIDE is the opposite of a property that a system should have.

<table>
<thead>
<tr>
<th>Threat</th>
<th>Property Violated</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spoofing</td>
<td>Integrity</td>
</tr>
<tr>
<td>Tampering</td>
<td>Non-Repudiation</td>
</tr>
<tr>
<td>Repudiation</td>
<td>Confidentiality</td>
</tr>
<tr>
<td>Information Disclosure</td>
<td>Confidentiality</td>
</tr>
<tr>
<td>Denial of Service</td>
<td>Availability</td>
</tr>
<tr>
<td>Elevation of Privilege</td>
<td>Authorization</td>
</tr>
</tbody>
</table>

Table 3.1: Properties violated in STRIDE.

STRIDE does not aim to present a solution to the identified threats, it helps in the process of threat modeling by offering a standard way to identify the usual attacks the system tends to experience, or by other words, enumerate the various things that might go wrong with the system.

3.6 Risk Treatment

This sections addresses several options to mitigate and solve security problems arisen during the implementation of our system. To do so, these are the key tools used:

• Risk Mitigation - implementing measures to reduce the risk impact or the likelihood to an acceptable or transferable level;
• Risk Avoidance - avoid carrying out an activity that would carry risk;
• Risk Sharing/Transfer - totally or partially transfer (through insurance or outsourcing) the risk impact to a third-party;
• Risk Acceptance - accept the impact of a specific risk, considering that any other risk treatment option is too costly or inadequate.

3.6.1 Throttling

To prevent some types of attacks, as for example DoS to a server, bandwidth throttling can be used. It is a reactive measure employed in communication networks in an attempt to regulate network traffic and minimize network congestion by intentionally slowing down the service access.

This throttling can be defined in the Service-Level Agreement (SLA) established with the Internet Service Provider (ISP).

3.6.2 Hardening

Hardening is the process of securing a system by systematically reducing its attack surface and by providing appropriate vulnerability mitigation and patching. A system has a larger attack surface the more
functionalities it provides to the surrounding environment. Reducing available attack vectors typically includes the removal of unnecessary software, unnecessary user accounts and disabling or removal of unused services.

There are various methods of hardening. This may involve, among other measures, patching the OS kernel, closing open network ports, setting up intrusion detection systems, firewalls and intrusion prevention systems.

3.6.3 Intrusion Tolerance

Instead of focusing on preventing all intrusions, intrusion tolerance deploys triggering mechanisms that prevent intrusions from leading to a system security failure, by providing adequate mechanisms to fall-back to and to restore the systems to a secure state after the occurrence of an intrusion. In other words, the objective of intrusion tolerance is to maintain system services secure even in the face of an intrusion.

3.6.4 Secure Channel

A secure channel is a communication channel which aims at delivering the following properties:

Confidentiality - By making use of encryption technology, it is possible to ensure the information exchanged in a secure channel is only accessible to authorized parties.

Integrity - The integrity property aims at ensuring that the packets are not tampered with during the transmission, or if so that it is easily recognized by the party receiving the information. This property is typically supported by using cryptographic hash functions, by having a digest of the information being exchanged sent along, and being confirmed by the party receiving it;

Authenticity - The authentication of the packets exchanged between the parties intervening on a protocol ensures the identity of the party that generated each packet, and thus prevent the packets from being generated by a malicious party. This property typically consists of applying some operation to the information being exchanged using a private cryptographic key that is only possessed by the generator. Later this operation might be verified by any other party.

3.6.5 Multi-factor Terminal Authentication

Multi-factor authentication is an approach to authentication which requires the presentation of two or more authentication factors, to provide a stronger assurance of the user’s identity. After presented, each factor must be validated by the other party for the authentication to occur.

With this technique, some threats can be mitigated, as for example, a session hijacking is more difficult to be executed since the attacker as to be able to exactly replicate all the required authentication factors.
3.6.6 Location

By determining the location of the user, either in an absolute or relative way, it is possible to provide some additional protection against some types of attacks, such as a man-in-the-middle proxy attack.

To get the user location, more than one technique can be used. The location can be determined based on the IP address, on the mobile network information, GPS, and several other ways.

3.6.7 History

Correlating information about the users transactions, the system can define a threshold for the expected actual behavior of each user, and thus delimit what that user is expected to do. As an example, if a user did validates a metro ticket in a location and some minutes later tries to validate again in a different location, which is geographically impossible for the user to be at, the system detects fraudulent usage and does not give access to the attacker.

3.6.8 Timing

Another possible technique to prevent proxy attacks is by measuring the communication latency, as a time boundary. Based on the result, if the time taken to perform a given task surpasses a predefined value, the system can assume that the mobile phone is relaying the information to another device.

As for example, if a communication takes, in average, 30 ms to execute from end-to-end and if the time to perform that operation takes longer than the average plus a given threshold, the system can deny the access to a possible attacker.

3.6.9 Sparse ID Domain

A sparse ID domain is a possible technique to mitigate login abuses that eventually could lead to block a victim’s account.

The main idea is to have a large number of possible IDs for the users and only have some of them being used, in a non-sequential order. If an attacker wants to abuse the login service, using a brute-force technique, it would be harder since the attacker would need to guess which IDs are being used, from the complete domain of possible IDs.

3.6.10 Authentication Throttling

To prevent login abuses, one possible solution is to apply throttling to the login service. One possible application of this technique is to introduce a delay between consecutive authentication attempts, and thus increase the amount of time a brute-force attack would take.
3.6.11 Data Protection at rest

Data at rest, is any data that is not being transmitted or used. Data protection at rest refers to the protection of persistent data stored. To prevent this data from being accessed, modified or stolen, data protection best practices should be applied, as for example password protection, data encryption, etc.
Chapter 4

Architecture & Solution

In this chapter there is presented a ticketing system used as a reference (section 4.1), following by a proposed solution (section 4.2) that aims at implementing HCE using NFC technology in the reference ticketing architecture.

4.1 Reference ticketing architecture

The proposed solutions aims to fit on a standard ticketing system without significant changes. Implementing a completely new ticketing system incurs in enormous costs, which would prevent the uptake of the solution. This section explains the generic components and processes of a standard ticketing system. Without loss of generality, the Calypso™ ticketing system [5] will be used to describe the reference architecture, whenever necessary.

4.1.1 Ticketing system components

Figure 4.1 depicts the relations between the elements of the reference ticketing architecture. The Master Secure Access Module (SAM), is responsible for the generation and storage of the various systems keys [6]. The SAM only supports symmetric keys that, by definition, must be shared by every communication party, including the users, which are not completely trustworthy to not leak the keys. To prevent it, the SAM implements a set of security management measures (section 4.1.1).

![Diagram of ticketing architecture]

Figure 4.1: Various elements involved in the ticketing architecture.
Master Sam

The Master Secure Access Module is responsible for maintaining all system keys generated during the system initialization (Bootstrap). In the case of the Calypso™, these keys are set on the course of a process named “Key Creation Ceremony” [33]. The Master SAM is in charge of the distribution of those keys to all the other system components. The SAM server only supports the creation of symmetric keys. These keys must be shared with the users devices that are not completely trustworthy or have no internal clock (smartcards), therefore, are not able to validate the lifetime of a key. To prevent these and other security problems, the SAM personalizes keys to specific applications, thus reducing the impact in case of leak, and limit the number of their usages. For instance, in the Calypso Technology the keys can be linked to an event counter which is decremented for each usage until it reaches a specified ceiling and the SAM rejects it [6]. Further information about SAM keys is denoted in section 4.2.1.

Back Office

The Back Office provides back-end support for the ticketing system. For example, database management, card management, transactions, merchants management, and all the interfaces of these systems.

Validator Machine

The validator machine is the component that interacts with the user to validate his smartcard, granting him access to the transportation service, or to recharge the smartcard.

Fiscalization Machine

The fiscalization machine is often a mobile machine transported by an agent, and is responsible for inspecting smartcards, ensuring their validity and detect possible frauds.

User smartcard

User tokens are usually contactless electronic smartcards. This physical cards enables the interaction with the Validator/Fiscalization machines, to ensure the valid usage of the PT.

Each of these protocols have a different purpose and run at different tiers (Figure 4.2)

4.1.2 Ticketing Protocols

The ticketing architecture, just described, executes a wide variety of protocols. In Figure 4.2 is presented a succinct general architecture of a ticketing system and their operations.

Bootstrap

The Bootstrap protocol is responsible for the initialization of the system. During this protocol, a master authority instructs the master SAM to generate and store the necessary cryptographic keys, for each
component of the system. After the generation of the keys, they are distributed among all the necessary
Validator/Fiscalization machines.

User enrollment

The user enrollment protocol, starts with the user buying a ticket from the Validator Machine, or inter-
acting with a vendor. After a period of time, the user receives his smartcard with the necessary data to
ensure its integrity and authenticity. Enrollment of a new user in the system is depicted in Figure 4.3.

Recharge smartcard

The smartcard recharge protocol loads a specified value into the smartcard. The protocol starts when
the user inserts his smartcard in the validator machine, which verifies the card integrity, authenticity, and
validity, after which the user selects the value he wants to recharge into the smartcard, and pays using
money or credit card. When the Validator receives the correct value, updates the smartcard, stores the
operation in the database, and concludes the protocol.
Use smartcard

During the Validation/Fiscalization protocol the user taps the smartcard into the Validator/Fiscalization machine, which uses a specific key emitted by the SAM Server, to verify the card integrity and validity. If the smartcard is valid, the user is granted access to the service.

Reconciliation

The reconciliation protocol is executed on a pre-defined period of time, typically at the end of the day, or during a maintenance period. The reconciliation protocol starts when all the Validator machines send the event logs of all operations executed since the last Reconciliation to the back office. Once the data arrives, it is analyzed to search for anomalies so the back office can update the information concerning the smartcards, and register the operations in a database. The protocol finishes by sending updated information (e.g. user blacklists) to Validators.

4.2 Proposed Solution

In this section we present the components involved in the design of the architecture, their technical limitations, and possible workarounds. The cornerstone of our solution is a VirtualCard (Fig. 4.6) per client maintained by a central server in the cloud. The main objective of the VirtualCard is to enable the secure usage of a mobile device in a ticketing system, which is achieved by a combination of a tokenization strategy (section 4.2.1) and the maintenance of revision files per token (section 4.2.1).

This section starts with a description of the main components of the solution. Then it proceeds with the descriptions of the overall architecture (Figure 4.7), and of the new augmented protocols used to support that architecture (section 4.2.3).
4.2.1 Solution Components

Cryptographic Keys

Besides the necessary cryptographic keys to maintain a ticketing system, our solution uses of two extra symmetric cryptographic keys generated by the Master SAM:

1. Master Cloud Validation Key ($K_{vc}$) - the symmetric key $K_{vc}$ is composed by two different keys: $K_{vc1}$ to encrypt/decrypt the information and $K_{vc2}$ to sign/verify the content. Although these keys are not used in current deployed solutions, most current deployed SAMs may be configured to use these keys without much changes, therefore their usage has only a low impact on the current deployed systems. These keys are shared by both the cloud service (CS) and the PT Validators. For simplicity, on our solution we make no distinction between $K_{vc1}$/$K_{vc2}$, and refer to both as $K_{vc}$.

2. Validator Machine Key ($K_{vv}$) - $K_{vv}$ is owned by the Validator Machine and is used in the first and subsequent usage of the VirtualCard (section 4.2.4).

Tokenization

The process by which value is transferred in different types of digital tokens which gives access to PT systems. Tokenization is used to create a low value token containing only the amount necessary to use the service on a defined period of time, therefore reducing the impact of a potential attack. If a token is predefined to be used only on a certain route and within a certain period, if the token resides inside the mobile device, and the device gets stolen it is likely that the validity of the tokens in the device has already expired, and the ones that have not expired have a low value. The token is encrypted with a master cloud validation key ($K_{vc}$), only accessible by the CS. The tokens contain the following information (Fig 4.6):
1. **Emulation Block**: The structure depends on the technology being used by the underlying ticketing system. The objective is for the structure of this block to be as similar as possible to the existing card technology. By doing this we require fewer changes to the Validator, and therefore easier integration with already deployed systems;

2. **Security Block**: Composed of various fields containing the following information:

   - **Environment and Holder file**:  
     - Transport application information (version number, the network identity, a validity date);  
     - Card holder information (social profile, reduction rate, etc).

   - **Contract file**:  
     - Client identification;  
     - Contract validity period;  
     - Contract type;  
     - Geographical validity (optional);  
     - Usage restrictions;  
     - Sales information.

   - **Event Log containing the timestamps of each validation and further relevant information to be used in the exit validation protocol;**

   - **The average Round-Trip-Time (RTT) measured between the user’s device and the Validator machine. The RTT is used as an extra security measure, to verify if the communication time with the user mobile device is within the average period. This measure helps the identification of a possible attacker using a mobile device remotely. The overhead latency in the network, is often enough to identify a non legit access.**

   - **Token validity, which specifies for how long the token is valid. This information is used in the VirtualCard Integration Layer, more specifically in the Security Layer (section 4.2.2). Issuer tokens are generated by the cloud service upon request by the user mobile device and are kept together with the correspondent revision file (section 4.2.1).**

### Revision File

Revision files contain historic information on each issued token. After being issued, tokens are never changed, although they may be partially consumed. Tokens are signed by the CS, with \(K_{vc}\), and cannot be changed by the Validators. On the other hand "revision file entries" are created and signed by each validator with the Validator Key \(K_{vv}\). A "revision file entry" not only consumes part of the token but also extends its validity. This is crucial to prevent denial of service attacks, if by some reason a user is able to prevent a revision file entry from being added to the token with the intent to hide its partial consumption, the token will be invalid as a whole. Revision files are created on every operation originated by a Validation/Fiscalization, and associate the following information:
Figure 4.6: VirtualCard structure.

- Revision File Id;
- Assigned Token Id;
- The Validator machine Id;
- Card validity time period;
- The changes done to the blocks of the FS;

With the information of what changes have been done to the original blocks of the VirtualCard FS, the VirtualCard Integration Layer (section 4.2.2) can apply those changes to the static token, and reproduce the updated VirtualCard.

### 4.2.2 Architecture

**Cloud service**

The cloud service, is our version of the Back Office (section 4.1.1), it manages the user account, load tokens into the user mobile device, and perform the reconciliation protocol stated in section 4.1.2.

**End user mobile device**

The user mobile device is a direct substitute of the smartcard, that contains a VirtualCard.

**Technology Selector**

The Technology Selector is an extra feature added to the validator machine, in the reference ticketing architecture (section 4.1), that enables the identification of the technology being used, either VirtualCard or a physical card.

**VirtualCard Integration Layer**

Validator machines are usually not able to handle VirtualCards, they only know how to handle specific technology cards. In the proposed solution a VirtualCard Integration Layer (VCIL) will be deployed which is divided into two components:
1. The Technology Adaptation Layer (TAL): Responsible for providing backwards compatibility to existing technology, it achieves this by emulating a specific card technology protocol (e.g. Calypso, CTS-512, etc.) and using a VirtualCard File system, provided by the layer below, to securely store the necessary state information. With this layer no other change is necessary to upper layers of the validator machine.

2. The VirtualCard Security Layer (VCSL): Assures the security of the VirtualCard to the TAL. The VCSL maps the VirtualCard into a File System (FS), which permits secure read/write operations on the VirtualCard, by the TAL or the upper application layer, making the revision file mechanism totally transparent to the upper layers. Each time the user performs an operation that interacts with the VirtualCard, the VCSL creates an updated version of the revision file. Prior to mapping the VirtualCard into a FS, the VCSL checks the integrity and validity of both sections of the VirtualCard, the token and revision file. The VCSL asks the Validator SAM to decrypt the token with the master cloud validation key ($K_{vc}$), decrypt the revision file with the Validator key ($K_{vv}$), and check if their validity field is within the established limit. Later at the application layer (section 4.2.2), the contract validity is verified.
Validator Reader Layer

The Validator Reader Layer (VRL) provides, to the application, access to specific card technology, through the VCIL and the Technology Selector.

Application layer

Already present in the reference ticketing architecture, and is responsible for implementing the ticketing application. This layer can be used without major changes, regardless of the technology used at the moment, smartcard or VirtualCard.

4.2.3 Protocols

Messages Syntax

The following syntax is used to describe the messages exchanged in each protocol:

$$
\langle x \rangle_y = \text{Enc}(\text{Sign}(x)_{y2})_{y1} \\
x = \text{Message Data} + \text{ReceiverId} \\
y = \text{Keys} = y1 + y2 \\
y1 = \text{Encryption Key} \\
y2 = \text{Signature Key} \\
+ = \text{Concatenated with} \\
\text{Sign}(x)_{y2} = x + \text{Enc}(\text{Hash}(x))_{y2} \\
\text{Sign}(x)_{y2} = x + \text{Encrypt(Hash(x)) with key y2} \\
\langle x \rangle_y = \text{Encrypt(RESULT)} \text{ with key y1} \\

\langle X \rangle_{Y \oplus Z} = \langle X \rangle_{\text{Hash}(Y + Z)}
$$

All the secure content represented as $$\langle x \rangle_y$$ uses a Sign then Encryption solution with name binding [13].

Registration

The registration of a new user in the cloud service is as shown in Figure 4.8. The registration protocol starts by establishing a secure communication channel with the cloud service, using Transport Layer Security (TLS) and requesting for a new registration. After receiving the registration request the cloud service asks the user to define a password and picks a random unused identifier from a sparse ID domain, to be used as a user ID. Both the chosen ID and a derivation (PBKDF2 [39]) of the chosen password are stored in an account for the user, together with a new VirtualCard. Finally, the cloud
service sends a notification to the user informing him of his user ID. In normal operation the user ID will not be typed by the user, instead it will be stored by the mobile device and handed to the cloud service whenever necessary, therefore it can be a big number from a sparse space, thus raising the bar to perform brute-force attacks on the system authentication.

Comparing this protocol to the enrollment of the user in the ticketing solution (section 4.1.2), we introduce the cloud service, and the creation of the VirtualCard. The biggest drawback in this case is the requirement of an internet connection.

Mutual authentication sub-protocol

After the registration protocol, a mutual authentication is executed each time a consumer uses the cloud service. This protocol starts by establishing a TLS channel between the user’s mobile phone and the cloud service. The user inputs his password on the mobile phone and submits it to the cloud, along with the device ID and the user ID assigned by the cloud service in the registration protocol. On the cloud service side, the system applies a PBKDF2 to the user password, transforms it to the derived key, and compares it with the one stored in the database. If the key matches, the authentication is successfully completed. The main objective of this protocol is to validate the user identity through a mobile device. It must be noted that, given the modular nature of the proposed architecture, the mutual authentication protocol may be strengthened with additional authentication factors.

For instance, non invasive authentication mechanism like Touchalytics [20] could be added without much visible changes. To enhance the security of this protocol, the user and his device could be authenticated with a rotation of mechanisms, adding diversity and longevity to the system. For instance, the user may be authenticated, on a first login, with the use of his password, on a second authentication, with Touchalytics, and on a third authentication using a biometric mechanism, like a fingerprint scanner, if the device is compatible.


**Recharge**

The recharge protocol loads value into the VirtualCard stored in the cloud service. The recharge protocol begins by executing the mutual authentication sub-protocol, and if successful, requesting the cloud service to initiate a recharge. The cloud service asks the user for the amount to recharge, and the user replies with a message containing the value and the method of payment, after which the payment must be completed through an external payment company, and if successful the specified amount is added to a user store-value account.

![Recharge Protocol Diagram](image)

**Figure 4.9:** Recharge Protocol.

### 4.2.4 Usage of VirtualCard

Using a VirtualCard requires transferring some of the value in the store-value account into a specific token and load it into the VirtualCard and the user device. Only after loading the token into the device it is possible to redeem the token on a validator machine and use the PT.

Tokens may be redeemed in a single trip or in multiple trips. The process differs slightly if the token is being redeemed for the first trip or for subsequent trips.
Load tokens sub-protocol

Loading tokens into the user mobile device (Fig.4.10) has to be executed, at most, $T_{load}$ minutes before the validation protocol starts. While in the device the token can be copied and used by anyone else, by restricting the usage of the token to a short period after it has been generated the vulnerability window is reduced. $T_{load}$ can be changed but not without taking into account the trade-off between usability and the level of risk associated.

Whenever a user plans to take a trip in the near feature, he selects on the application what is the transportation service he intends to use. If required the application requests additional information about the topology of the network (e.g. metro zone). After choosing the transportation service the device requests a token for it and the user is required to authenticate himself by the executing the Mutual Authentication sub-protocol (section 4.2.3).

Upon receiving the request, the cloud service loads the user VirtualCard and verify if it has enough balance to create the token requested and if the user is not blacklisted. If the verification checks, the token is generated with the information depicted in section 4.2.1, and two special messages are generated and sent to the user’s device, the TokenDecryptMessage message and the EncryptedTokenMessage.

The TokenDecryptMessage message contains a random value $R$ and the user password $Spw$ encrypted with the Master cloud validation key $K_{vc}$:

$$TokenDecryptMessage = (R + Spw)_{K_{vc}}$$

The TokenDecryptMessage can be decrypted by all the validators and by the cloud service, therefore it cannot be changed in transit or in the users device. A better solution may be implemented if the cloud service is provided with more information about where the user is going to use the token. In that case the cloud service may encrypt the TokenDecryptMessage with a key shared by those specific validators.

The EncryptedTokenMessage contains the created token, encrypted with both the Master cloud validation key ($K_{vc}$), and a key generated from the random $R$ combined with the user password $Spw$:

$$EncryptedTokenMessage = ((Token)_{K_{vc}})_{R\oplus Spw}$$

By double encrypting the token the protocol ensures that the token must be redeemed on a legit validator (one with the correct key $K_{vc}$), at the will of the user that knows the password $Spw$. The random value $R$ prevents trivial replay attacks.

Single or first redeem of token

 Redeeming a token requires the user to insert his password in the application, transforming it to $Spw$, and start the NFC communication with the Validator machine, by tapping the Validator with the mobile device. When the connection is established the user device sends the TokenDecryptMessage to the Validator, that decrypts it using the master cloud validation key, $K_{vc}$, gains access to the random value $R$ and the user password $Spw$.  

To assure that the user is legit, and to grant him the possibility to decrypt the EncryptedTokenMessage, containing the encrypted token, the Validator creates the UserCheckMessage, encrypted with the user password, $Spw$:

$$\text{UserCheckMessage} = (R + \text{Chal})_{Spw}$$

Moments before sending UserCheckMessage, the Validator starts measuring the RTT of communicating with the user’s device, to later verify if it is within the average/expected time. When the user’s device receives the message, decrypts it using $Spw$, and gains access to $R$ and $\text{Chal}$. With that information the device decrypts the EncryptedTokenMessage, ciphered with the master cloud validation key, $(\text{Token})_{Kvc}$. Finally the mobile device creates the TokenMessage, and sends it to the Validator:

$$\text{TokenMessage} = (\text{Token})_{Kvc} + (\text{Chal})_{Spw}$$

After receiving this message the Validator decrypts the token, using $Kvc$, checks the token integrity, then decrypts the $\text{Chal}$, using $Spw$, and compares it with the $\text{Chal}$ created previously. To ensure that the time is within acceptance range, the Validator stops the RTT timer, reads the executions times from the token (updated during reconciliation), and compares them. This security measure takes into consideration that the overhead injected by a internet connection, is many times superior to a NFC connection, making it possible to identify a possible attacker using a botnet of mobile devices. If the execution times are within an acceptable range, and the user $\text{Chal}$ is correct, the Validator analyses the token validity period, and if provided, checks if the geographical location is on the same area of the Validator. After these verifications the Validator executes the technology specific protocol, and emulates
a specific card technology as described in section 4.2.2.

Upon emulation of the specific card technology, the Validator creates an updated version of the Revision File (section 4.2.1), and a random value $R_2$ used to encrypt the Revision File. To send this information to the user mobile device, the Validator creates and sends the RevisionFileMessage:

$$\text{RevisionFileMessage} = \langle\langle\text{Revision}\rangle_{K_vv}\rangle_{R_2}^{\oplus}S_{pw} + \langle R_2\rangle_{K_vv}$$

Upon sending the updated version of the Revision File, the Validator grants the user access to the transportation network. This protocol is depicted in Figure 4.11.

Subsequent redeems of a token

On subsequent redeems of the same token, the Validator Machine has to receive from the user mobile device, the token, and the updated revision file. With this information the Validator can apply the revision file information to the token, and compile the current state of the VirtualCard FS. The protocol involved in this procedure is depicted in Figure 4.12.

The protocol begins by the user inserting his password into the application, which transforms it to $S_{pw}$, and starts the NFC communication. After establishing the connection, the mobile device sends
the *TokenDecryptMessage* and \( R_2 \) to the Validator. Upon receiving this messages the Validator gains access to the random value \( R \), the user password \( Spw \), and the most recent random value \( R_2 \). With this information the Validator creates a new challenge (\( Chal \)) to assure the next message sent is recent and mitigate replay attacks. After this the Validator sends an *UserCheckMessage* to the user mobile device, along with \( R \), and \( R_2 \) all encrypted with the user password \( Spw \). Ciphering the message with \( Spw \) guarantees that only the user can read its contents, therefore assuring confidentiality. The random value \( R \) is sent to the mobile device, because on the first validation (section 4.2.4) the device had access to \( R \) but does not save it on the device, for security precautions.

It is important to remember that at this moment the device contains the messages:

\[
\begin{align*}
TokenDecryptMessage &= ⟨(R + Spw)⟩_{Kvc} \\
EncryptedTokenMessage &= ⟨⟨Token⟩⟩_{Kvc} R \oplus Spw \\
RevisionFileMessage &= ⟨(R_2)⟩_{Kvv} + ⟨⟨Revision⟩⟩_{Kvv} R_2 \oplus Spw \\
UserCheckMessage &= ⟨(R + R_2 + Chal)⟩_{Spw}
\end{align*}
\]

With the reception of the *UserCheckMessage* the device decrypts it, and gains access to \( R \), \( R_2 \), and \( Chal \). The random value \( R_2 \) and \( Spw \) are used to decrypt the *RevisionFileMessage*, thus gaining the Revision File. With the random value \( R \) and \( Spw \), the device decrypts the message *EncryptedTokenMessage* with the Token. The \( Chal \) is encrypted with the \( Spw \), to ensure the user confidentiality. Finally the device creates the *VirtualCardMessage* and sends it to the Validator:

\[
VirtualCardMessage = ⟨Token⟩_{Kvc} + ⟨Revision⟩_{Kvv} + ⟨Chal⟩_{Spw}
\]

After receiving this information, the Validator verifies the token, along with the revision file authenticity, and checks if the user is legit by comparing the \( Chal \) received with the one generated previously. As in the first validation protocol (section 4.2.4), the Validator stops the RTT timer and checks if it is acceptable. With the revision file and the token, the Validator is capable of reconstruct the latest version of user’s VirtualCard in the VCSL, perform the respective operations and generate an updated version of the revision file. Finally the Validator sends to the mobile device the *NewRevisionFileMessage*:

\[
NewRevisionFileMessage = ⟨⟨newRevision⟩⟩_{Kvv} R_3 \oplus Spw + ⟨R_3⟩_{Kvv}
\]

The *NewRevisionFileMessage* serves the same purpose of the *RevisionFileMessage* sent in the first validation protocol (section 4.2.4). That is providing the encrypted, and updated revision file to the user mobile device. Upon sending the message, the Validator grants the user access to the transportation network.
Figure 4.12: Second validation protocol using the same token.
Chapter 5

Solution Limitations & Security Analysis

This chapter contains an analysis on the limitations of the proposed solution, along with possible mitigations. On section 5.2 we build a threat model using STRIDE (section 3.5).

5.1 Limitations of the Solution

The proposed solution has limitations that are important to be aware. Even though the redeem protocol is executed offline, the solution requires the end user to have a network connection and load tokens (section 4.2.4) prior to using the transportation service, which can lead to usability issues. A possible solution is to ubiquitously schedule loading tokens everyday during a period of time suitable for the end user. Another usability issue is due to the fact that in a validation protocol (section 4.2.4), a user has to insert his password on the device, in order to be transformed using a PBKDF2 algorithm. The protocol of inserting the password during a congestion time in the transportation service may lead to a certain degree of inconvenience, for that reason the user is advised to insert the password moments prior to using the service.

5.2 Threat Modeling

This section serves the purpose of analyzing the presented system, in a security point of view, with the help of STRIDE. As mentioned in section 3.5, STRIDE stands for, spoofing, tampering, repudiation, information disclosure, denial of service, and elevation of privilege.

Also on this section we answer the security threats described in section 1.1.

On Figure 5.1 there is displayed the main components involved in the system. The cloud service (section 4.2.2) manages the user accounts, loads tokens into the users mobile device, and performs the reconciliation protocol stated in section 4.1.2. The cloud service is also a dangerous element on our system in case of compromise, because it holds very sensitive and private information. The user mobile...
device (section 4.2.2) holds the transportation application which enables the interaction with the validator machines, the cloud service, and other components of the system. If the device is stolen, hijacked, or other form of attack, it may disclose the attacker to private information and unwanted access to the system. The validator machines (section 4.1.1) are responsible for interacting with the VirtualCards, implement security measures, and grant access to the transportation service. Finally there is the Master SAM (section 4.1.1) which is responsible for maintaining all system keys generated during the system initialization.

On the following subsections the problem is divided into the various elements of the system, in order to apply STRIDE to each one of them for a more thorough security analysis. The subsections identify groups of threats that may fall into the diverse categories of STRIDE, and a possible solution or mitigation mechanism. From the several system components we choose to focus our security analysis on the validator machines (section 5.2.1), the cloud service (section 5.2.2), the user mobile device (section 5.2.3), and the communication channels (section 5.2.4). The later refers to the various communications made during the protocols of our system, for instance, the communication between the validator machine and the cloud service or the cloud with the user mobile device.

5.2.1 Validator Machine

Spoofing

1. **Attack:** If the attacker breaks the cryptographic challenges of any Validator Machine, he can control it and pass as a valid machine to the user; **Mitigation/Solution:** The mutual authentication protocol (section 4.2.3) between the user and the Validator machines assures a secure communication.

Tampering

1. **Attack:** During the first and subsequent redeem of the tokens (sections 4.2.4, 4.2.4) there is an update message sent at the end of the protocol (RevisionFileMessage), which contains the new Revision file. If the attacker creates an application by reverse engineering the protocol he can
bypass this message and not update the VirtualCard;

**Mitigation/Solution:** During the reconciliation protocol (section 4.1.2) all the Validator machines send their data to the Cloud Service. With that data in possession it is easy to identify frauds like the one occurred in a rollback attack, because there is the information that a token has been used two times without any modification.

**Repudiation**

After an extensive analysis of several possible attacks we did not found any fitting for this section of STRIDE.

**Information Disclosure**

1. **Attack:** When the VirtualCard is emulated in the VCSL (section 4.2.2) if the attacker controls the validation machine he can access private information;

**Mitigation/Solution:** The VCSL maps the VirtualCard into a File System (FS) to enable secure write/read operations by the upper layers. But even those operations require the Validator SAM to decrypt the token with the master cloud validation key ($K_{vc}$), and decrypt the revision file with the Validator key ($K_{vv}$).

**Denial of Service**

1. If an attacker breaks the cipher’s challenge or cracks the cryptographic algorithm, he can discover all the Validator SAM’s cryptographic keys. With this information the attacker can simulate any Validator, and with it disrupt the service:

- **Attack:** By staying near the user device, or by using a proxy, the attacker can initiate a Validation protocol (section 4.2.4) and spend all the value loaded into the VirtualCard;

- **Attack:** The attacker may initiate a Validation protocol (section 4.2.4) and not emit a Revision File (section 4.2.1), thus not extending the validity period of the VirtualCard, and making it impossible for the user to validate it on a valid Validator machine.

**Mitigation/Solution:** A solution to the above attacks is rotating the SAM keys delivered to the Validator machines, this way we assure the attacker only possess a valid key for a short period of time.

**Elevation of Privilege**

After an extensive analysis of several possible attacks we did not found any fitting for this section of STRIDE.
5.2.2 Cloud Service

Spoofing

1. **Attack:** Infecting the Cloud Service with malware adds the danger of letting the attacker control the Cloud Service; **Mitigation/Solution:** Hardening mitigates the possibility of the Cloud Service getting infected, and Data protection assures even in the case of infection the data is maintained secure and safe by cyphering mechanism and encryption.

Tampering

1. **Attack:** If the Cloud Service is hijacked by the attacker, he can change various elements in the user account, for instance changing the payment information;
   **Mitigation/Solution:** Outsourcing the payment process, will transfer the risk of handling payments and payment card information, by using a service provided by a specialized market operator, with expertise in handling secure payment operations.

Repudiation

1. **Attack:** If the attacker controls the Cloud Service, he can manipulate the logs of all the transactions done, by inserting false attacks into the logs to confuse the mechanism.
   **Mitigation/Solution:** By applying Data Protection at rest (section 3.6.11) techniques, we can prevent the logs from being accessed. Keeping backups of the logs, enables a rollback of the data. Checking the integrity of the logs from time to time mitigates a possible attack.

Information Disclosure

1. **Attack:** Infecting the Cloud Service with malware that redirects users accounts to a server controlled by the attacker;
   **Mitigation/Solution:** Hardening mitigates the possibility of the Cloud Service getting infected, and Data protection assures even in the case of infection the data is maintained secure and safe by cyphering mechanism and encryption.

Denial of Service

1. **Attack:** If the attacker gains access to the cloud service and obtains the user accounts, he can abuse the login service during the mutual authentication protocol (section 4.2.3), making the system block any account obtained;
   **Mitigation/Solution:** A possible solution is throttling the number of logins allowed by the service.

2. **Attack:** An attacker could infect the cloud service with a malware that has the objective of corrupting every information intended for the users, making it impossible to create valid tickets. This attack allows to target all the users, or just a specific one;
Mitigation/Solution: Data protection techniques assures the security of the information. Intrusion detection systems running on the cloud service helps the detection and prevention of an attack.

3. The Cloud Service and the communication with the Cloud Service by the user mobile device plays an essential role in this system, if the service or the communication gets disrupted the user is unable to recharge the VirtualCard (section 4.2.3) or load tokens into the mobile device (section 4.2.4). This could be achieved, for instance, by corrupting the network packets sent to the cloud service;

Mitigation/Solution: These attacks represent a low risk to the user and to the system. If the user device is unable to connect to the network because, for instance, a rogue application installed disabled the network interface, or the network packets are being forged, the device can always be restored, and the rogue application removed.

Elevation of Privilege

1. **Attack:** A possible attack is sending input that the Validator machine can not handle. This can be done by changing or injecting blocks of memory into the VirtualCard, in a way that when the VirtualCard Integration Layer (section 4.2.2) reads the VirtualCard crashes;

Mitigation/Solution: This is a very hard attack because the attacker has to have a deep understanding of the composition of the VirtualCard and how to manipulate the block of memory reserved to it. And even if the attacker has that knowledge, he has to break the encryption involving the VirtualCard. A possible mitigation from the validator machine is verify the blocks of the VirtualCard prior to acting upon them.

5.2.3 User Mobile Device

Spoofing

1. **Attack:** Shoulder surfing attack while the user is performing the authentication with the help of the mutual authentication protocol (section 4.2.3) and Touchalytics mechanism;

Mitigation/Solution: An high diversity of authentication mechanisms used during the mutual authentication protocol (section 4.2.3) helps on the mitigation of this attack. The user can be asked to use different login mechanisms, for instance, a username password, a secret question, a fingerprint reader, and many others.

2. **Attack:** Stealing the mobile device to impersonate a user; **Mitigation/Solution:** In order to access the user account and impersonate him the attacker needs to break the user account security mechanism used at the time during the mutual authentication protocol (section 4.2.3). An ultimate case where the attacker gain access to the device there is also the tokenization strategy (section 4.2.1), which mitigates the amount of value stolen. And even the token is cyphered, so the attacker would need to break the cypher to read it.
### Tampering

1. **Attack:** An attacker can try to bypass the validation process and evade the ticket fare. A possible path on this attack is cloning the VirtualCard before being processed in a validation. After that the attacker uses the original card on the validation process and then restores the VirtualCard to its original state with the help of the cloned card (rollback attack), erasing the current usage of the VirtualCard;

   **Mitigation/Solution:** During the reconciliation protocol (section 4.1.2) all the Validator machines send their data to the Cloud Service. With that data in possession it is easy to identify frauds like the one occurred in a rollback attack, because there is the information that a token has been used two times without any modification.

2. **Attack:** The attacker can steal the mobile device and have access to its contents, for instance, the VirtualCard;

   **Mitigation/Solution:** Since all the data used by the proposed system that lies on the user device is encrypted, in order to access it, the attacker needs to break the encryption. An ultimate case where the attacker gain access to the device and breaks the encryption, there is also the tokenization strategy (section 4.2.1), which mitigates the amount of value stolen. And the attacker would need to comprehend the various blocks that compose the token.

3. **Attack:** The VirtualCard can be cloned by copying the memory segments on the mobile device that contains it. Thus granting the attacker a copy of the VirtualCard. Since the Reconciliation protocol (section 4.1.2) only happens once a day, the data on the Validators is only synced on that time. Allowing the attacker to use a different Validator from the legit user, or using all the Validators if the user did not use any on that time;

   **Mitigation/Solution:** During the reconciliation protocol (section 4.1.2) all the Validator machines send their data to the Cloud Service. With that data in possession the cloud service can analyze it and detect possible frauds, for instance if the attacker used a different Validator from the user, but at the same time. If that happens the Cloud Service can black list the user account, further investigate and refund it after the investigation is done.

### Repudiation

1. **Attack:** A user mobile device may be involved in fraudulent operations, and the user deny any involvement in them, for instance by saying the device got stolen;

   **Mitigation/Solution:** A possible solution for this situation is performing a history check on the user account to identify patterns that may lead to identify a possible attack, for instance, if the user always communicates from the same IP address range, or the same location, and the attacker did not. The user account is also immediately revoked and depending on the investigation results, restored or not.

2. **Attack:** If the attacker clones the blocks of memory containing the VirtualCard and uses the
account until it gets banned, there is no way to clarify if it was the legit user;

Mitigation/Solution: Same as the last attack;

3. Attack: An attacker can use a legit user payment card information, and the legit user claim to be a fraud victim;

Mitigation/Solution: Outsourcing the problem to a third party payment service assures the security of every payment card information. An extra security measure is checking on the user logs for fraudulent usage, for instance validation process in different places at impossible time frames.

Information Disclosure

1. Attack: An attacker may disclose information about the user, for instance, by infecting the user mobile device with a rogue application that returns the VirtualCard inside the device;

Mitigation/Solution: The VirtualCard is composed of the token plus the revision file, and both of the components are securely cyphered;

Denial of Service

1. Attack: The user can have the VirtualCard inside the mobile device, blacklisted by an attacker that gained control of the device, an uses it in a malicious way ensuring it gets blacklisted.

Mitigation/Solution: A possible solution is throttling the number of times a user can perform a login, preventing the account from being blacklisted. Also the mutual authentication protocol (section 4.2.3) assures even if the attacker gains access to the VirtualCard he can not use it. Tokenization lowers the value stolen by the the attacker.

2. The user can have his mobile device disrupted, thus disabling the access to any service:

  - Attack: The attacker can gain access to the mobile device, for instance by installing a rogue application, and remove the transportation application, and with it all the valid VirtualCards;
  - Attack: If the attacker tampers the NFC routing table by installing another application with the same application ID (AID), or changes the routing table making it invalid;
  - Attack: The attacker can drain all the battery from the user device;
  - Attack: The attacker can turn off the interface that gives access to the internet.

Mitigation/Solution: All of the above attacks disrupt the service to the user, but does not compromise the system. The user account is still intact in the Cloud Service, and can be reinstated.

Elevation of Privilege

1. Attack: A user may change the blocks of the VirtualCard in order to gain access to the token and change it to his benefit, for instance add extra value; Mitigation/Solution: Token security is assured by its encryption with the Master Cloud Validation Key ($K_{vc}$).
5.2.4 Communication Channels

Spoofing

1. **Attack:** The attacker can perform a Man-in-the-middle attack on the communication between the mobile device and the validator machine, this way the attacker can forge his identify and impersonate the user; **Mitigation/Solution:** The various messages traded between the validator machine and the user mobile device are ciphered with distinct symmetric keys (section 4.2.1), and spread across a TLS connection, assuring a secure communication. Also the mutual authentication protocol (section 4.2.3) adds extra security in the beginning of the communication.

2. **Attack:** The attacker can compromise the communication between the Validator machines and the Cloud Service by performing a Man-in-the-middle attack, adding the possibility that the attacker can impersonate any of the parties involved; **Mitigation/Solution:** The various messages traded between the validator machine and the cloud service are ciphered with distinct symmetric keys (section 4.2.1), and spread across a TLS connection, assuring a secure communication.

Tampering

1. **Attack:** The attacker can break the mutual authentication protocol (section 4.2.3) between the mobile device and the Validator machine, and bypass it completely. **Mitigation/Solution:** Tokenization (section 4.2.1) lowers the value of the VirtualCard compromised. During the reconciliation protocol (section 4.1.2) all the Validator machines send their data to the Cloud Service. With that data in possession the cloud service can analyze it and detect possible frauds, for instance if the user account had several trips inside the network not accounted for. If that happens the Cloud Service can blacklist the user account.

2. **Attack:** The attacker can forge his way into the validation protocol, if he has a group of infect users under his control (botnet), with a active legit account. The attacker can perform a proxy attack by redirecting the communication to a infected user, during the period of Validation; **Mitigation/Solution:** A possible solution is measuring and keeping a log of the Round-trip-time between the communication of the user and the validator machines. Since a proxy attack usually has a much higher RTT, this assures that when the attacker uses a proxy the validator machine can block the access.

3. **Attack:** After gaining access to the mobile device, the attacker can install a rogue application with the objective of performing a man-in-the-middle attack, and intercept the communication between the payment service and the user mobile device, in the Recharge protocol (section 4.2.3). Once the session is intercepted the attacker can give another bank account information to the payment service; **Mitigation/Solution:** Outsourcing the payment process, will transfer the risk of handling pay-
ments and payment card information, by using a service provided by a specialized market operator, with expertise in handling secure payment operations.

4. An attacker could tamper the payment reference information, or by other words, the user would be paying a bank account controlled by the attacker, and not a legit payment service.

- **Attack:** Man-in-the-middle attack in the terminal intercepting the communication during the Recharge protocol and changing the payment reference to the attackers account;

- **Attack:** Man-in-the-middle attack in the Cloud Service, the attacker can change the reference information on every recharge transaction to his bank account reference.

  **Mitigation/Solution:** Outsourcing the payment process, authenticating the payment token, hardening, and Data Protection, all of these solutions offer a solid solution to the attacks above.

**Repudiation**

After an extensive analysis of several possible attacks we did not found any fitting for this section of STRIDE.

**Information Disclosure**

1. **Attack:** Performing a man-in-the-middle attack can allow the attacker to eavesdrop the credit card information when the user is performing the payment through his mobile device;

  **Mitigation/Solution:** Outsourcing the problem to a third party payment service assures the security of every payment card information.

2. **Attack:** The attacker can discover which machines are involved in a communication by analyzing the network traffic.

  **Mitigation/Solution:** Hardening techniques (section 3.6.2) diminishes the possibility of this attack. Also as a preventive measure, in case of compromise, the servers can keep a range of valid networks addresses for each user, and when the user connect from a different range trigger an extra security measure, for instance, a secret question.

**Denial of Service**

1. **Attack:** A user can be prevented from using the service if the NFC communication is disrupted, to achieve this the attacker can jam the RF;

  **Mitigation/Solution:** A possible solution is using different frequencies from time to time.

2. Part of the Recharge protocol (section 4.2.3) is outsourced to a payment service, if an attacker compromises the connection between the user and the payment service, the Recharge protocol can not complete. The attacker can accomplish this by:
• **Attack:** Disrupt the connection between the user mobile device and the payment service;

• **Attack:** Perform a DoS attack to the payment service;

**Mitigation/Solution:** Outsourcing the payment service to a experienced and secure provider transfers the risk of handling payments and payment card information.

**Elevation of Privilege**

After an extensive analysis of several possible attacks we did not found any fitting for this section of STRIDE.

### 5.2.5 Security Threats Analysis

Our solution approaches the security threats described in section 1.1 in the following way:

1. Insecure Storage: The device contains only part of the information (token) and the revision file, both securely cyphered.

2. Relay Attack: By measuring the various RTT between the device and the Validators, and having the token loaded $T_{Load}$ time before an authentication, relay attacks become very hard to accomplish.

3. Insecure Communication: All the messages passed through the device and the Validator are cyphered, making the information more secure. Also by performing mutual authentication sub-protocol (section 4.2.3) the system prevents a possible insecure communication

4. Denial of Service Attack: One possible limitation comes from the fact that the device may be damaged in away that makes the use of the service impossible. The NFC route table on the device can be altered, therefore disrupting the communication. The DoS attack can also target the CS, which makes impossible to load tokens to the user device, and therefore also disrupt the service. DoS is a limitation that we can not prevent.

5. Privacy: By cyphering the various messages, and performing an authentication protocol with the user, we assure the privacy of the data.
Chapter 6

Implementation

Researching in the area of implicit authentication in mobile devices, we found work that look at the problem through a very different perspective, for instance, continuously analyzing typing patterns to create a user fingerprint [38]. Soon we abolished those options because our application will not have enough text input data to generate a fingerprint. So we turn our focus to a mechanism that takes advantage of the user interaction with the device screen.

After researching for systems that meet our criteria, we found some promising work [14, 4, 42, 18]. We prefer a solution that doesn’t affect usability too much, or if possible at all. A solution that identifies the user in an invisible way, and its resistant to shoulder-surfing or smudge attacks. And doesn’t need extra hardware in order to work. Taking all those criteria to weight, we choose the system proposed by Mario Frank et Al, Touchalytics [20]. As reviewed in section 2.2.1, touchalytics proposes a system capable of identifying a user by extracting a set of raw data, position, time, pressure, area, and with it compute a set of biometric features. We choose touchalytics because on contrary to other work reviewed, it doesn’t require a specific user action. For instance in [1], the user has to choose from the top 10 combinations, which ones he wants to perform. This reduces the usability of the system and makes the user involved in the procedure, both things we are trying to reduce.

6.1 Touchalytics implementation

Touchalytics implementation is divided into two sides:

- **Server-side**: Written in Node.js and responsible for exposing a web service via a JSON interface, handling the user connections, calculating all the biometric features, and storing the necessary data in a persistent way.

- **Client-side**: Mobile application for iOS, responsible for monitoring the user interaction, extracting the necessary raw data, position, time, pressure, area, and send it to the server-side in a JSON format.
6.1.1 Used technologies

• Server-side:
  
  – Node.js \(^1\): Node.js is a programming language written in JavaScript that excels in building RESTful APIs. This happens because it is a non blocking single-thread asynchronous language. Making it very good in handling web services via JSON interface.
  
  – Express \(^2\): Framework for Node.js that enables the creation of web services. It has a big community around it, it is the most used framework, and fills our necessity to create a simple web service.
  
  – Node-svm \(^3\): A library for Node.js that implements Support Vector Machines.
  
  – MongoDB \(^4\): NoSQL database, that provides an easy way to structure data in a JSON format, which for us was ideal because it was the chosen format to communicate between the client and the server.
  
  – Mathjs \(^5\): Mathjs provided an entire math library that was crucial in the calculation of the various biometric features in the Touchalytics study.

• Client-side: The client-side was composed of a mobile application for iOS, written in Objective-C. In section 6.1.2, the various design decisions and states of the application are presented.

6.1.2 Design decisions

Client Side

To make a user fingerprint there is a necessity to emulate an application that set the minimum constraints. In other words, an application that is capable of offering the same result with different paths. On our first approach we designed an application that did not meet this requirements. On an initial screen (Fig. 6.1) the application requests the user email, for identification purposes only, and some other data for statistics (age, sex, device model). On a second screen (Fig. 6.2) the user was asked to select either a Gallery (Fig. 6.3) or a Text (Fig. 6.4), and swipe through them. The gallery provided us horizontal swipes, and the text vertical. After completing 20 swipes in each section, the data was sent to the server. The results of this approach are depicted in section 7.3.

Designing the client side application was one of the biggest challenges. Our initial approach did not offer many ways for the users to navigate in the application. After observing several users performing this experiment, we could conclude that the first swipes were made in a “natural” way, but the majority was performed in a quick matter just to finish the experiment. Taking all this into mind we tried a different approach and designed an application with the objective of offering more navigation and less methodical orders, like perform a swipe 20 times.

\(^1\)https://nodejs.org/
\(^2\)http://expressjs.com/
\(^3\)https://github.com/nicolaspanel/node-svm
\(^4\)https://www.mongodb.org/
\(^5\)http://mathjs.org/
Figure 6.1: First application initial screen.

Figure 6.2: First application second screen.

Figure 6.3: First application gallery.

Figure 6.4: First application text.
The design decision for our second application was to create a shopping application, where the user is able to browse different products, select various payment methods, and simulate a product acquisition. The application starts with an initial screen similar to the previous one (Fig. 6.5), where the user inserts his email for identification purposes, and some other statistical data. On a second screen (Fig. 6.6) the user can browse different products, providing, in a best scenario, to the iOS application vertical and horizontal swipes. After selecting a product a screen shows different payment methods (Fig. 6.7), this screen may provide vertical swipes. Finally the user selects from a range of cards (Fig. 6.8), and concludes the acquisition of the product selected. This last screen may also provide horizontal swipes.

**Server Side**

For Touchalytics we have two possibilities. Either we implement the raw data extraction in javaScript, and the mobile device sends this data to the server, where everything else takes place. Or we implement also the computation of the biometric features inside the iOS application, and then send it to the server.

Sending just the raw data may require a higher number of communications to the server and more data transferred. But making the computation of the biometric features inside the mobile device may lead to performance issues.

If we choose to send the raw data to the server, we need to create a server-side application capable of reading it, compute the biometric features, and build a classification model using SVM.

This mechanism requires a training phase prior to the authentication phase. During the training phase, the system has to fallback to another security method, for instance one of the other mechanisms, or a username/password. In that period, the system will collect raw data from the user interaction with
the device. After a number of strokes done by the user, the device sends the raw data to the server. On
the server side, an application responsible for implementing SVM, creates a classification model using
one-against-all technique. After the training phase comes the authentication phase, where the raw data
extracted from the user mobile device, is sent to the server in order to verify if the user is legit or not.

We identified some key concerns with this mechanism and the integration with the general project.
In the following list we explain those concerns and our initial idea on how to approach them:

- **Dataset**: generating a dataset usable in this type of situation. Touchalytics tested the influence of
  the sample size in the system, after 20 users the EER of the system converges to around 3%. Our
  initial approach will be building a dataset using a sample off no-less then 20 users.

- **Classification model**: Touchalytics used SVM as their classifier, they also tested using k-nearest-
  neighbors. k-NN stores each training observation and their label, this may pose some problems
  in our system, where we expect a high number of observations. SVM merely saves the decision
  hyperplane. We propose testing SVM, evaluate the results, and if needed compare it with another
  classifier. Another concern is the implementation of a classifier. What we propose is using the
  work done by the authors of LIBSVM \(^6\), and award winning software solution for support vector
  machines. There is also many extensions to their software that support a substantial range of
  programming languages, including JavaScript.

- **Mobile application**: in our situation we are using iOS system, on contrary to Touchalytics (and
  \(^6\)http://www.csie.ntu.edu.tw/~cjlin/libsvm/)
most of the work researched) that uses an Android system. We suspect this has to do with the restrictions applied by Apple on designing applications, and the public API available to users. For instance Touchalytics extracts the pressure of finger on the screen, and the area occluded. This was not possible in iOS prior to 8.0. And even in the current iOS (8.1) there is only the possibility of obtaining the area occupied by the finger on the screen. What we propose to obtain the pressure on the screen, is calculating an approximation, and give it a predefined value. For instance if the area occupied by the finger is bigger than a threshold we say the pressure on the screen is high and we give it the value of 1, if the area is less then the threshold we say the pressure is low and give it the value 0. We suspect this approach will be sufficient, because after examining the data made public by Touchalytics, most of the pressure retrieved from the devices had the value 0. This happened because the majority of the devices used in their experiment didn't have a pressure sensor available.

In our project we divided the implementation of Touchalytics into four parts:

1. Create a mobile application in iOS that is capable of interacting with the iOS API and retrieve the raw data used in Touchalytics.

2. Implement RESTfull architecture that enables the communication between the client and the server side.

3. Generate a Node.js application that gathers the raw data and compute the biometric features.

4. Use LIBSVM implementation, to create a SVM capable of generating a fingerprint of the user, with the help of his biometric features.
Chapter 7

Evaluation and Results

In this chapter there is presented the various evaluation metrics used in the project (section 7.1), and the tests performed (section 7.2) in order to obtain the results derived from the evaluation metrics. The results are displayed and analyzed on section 7.3.

7.1 Evaluation Metrics

The project evaluation was made in the terms of the following standard metrics, false positives (FP), false negatives (FN), true positives (TP), true negatives (TN), and accuracy. Also we took some extra scenarios in consideration, when comparing to related work in the field. Besides the standard metrics we also evaluate the system in terms of training delay.

In the following subsections we explain the metrics used in a more thorough manner.

7.1.1 Standard Metrics

Accuracy (7.1) is used to calculate the rate of positive results when taking into consideration all the results obtained.

\[
accuracy = \frac{TP + TN}{TP + FP + TN + FN} \times 100\%
\]  

(7.1)

This metrics will help us decide when the classifier has ended the training phase. When we achieve an overall high accuracy, we can admit that the training phase is over, and the system collected enough data to create an effective user fingerprint. What we propose to test the system with these metrics is building a dataset with no less then 20 different users, containing their biometric features. This way when we get a new user into the system, we have sufficient data to build a classifier with positive and negative samples, and we can simulate an attack (using negative samples) to calculate the true negatives rate.

57
7.1.2 Training Delay

An interesting metric is the training delay. What this measures is the delay in seconds, that it takes the system using the Touchalytics mechanism, to train the classifier. The way we accomplish this is by measuring the time between sending the data from the mobile device to the server, during the training phase, and the time it takes the server to respond with a positive answer, indicating that the training as concluded with success. The less the training delay takes, the less the system takes to transit from the training phase to actually being able to identify an authenticate a user.

7.2 Evaluation Tests

To obtain the values used in the evaluation metrics, we performed a series of tests, that evolved along with the knowledge obtained during the implementation of Touchalytics.

Using the first version of the iOS application, we conducted a laboratory experiment with a group of 14 people. The test consisted in handing the iOS device to the user, and without further guidance let him use the application. The iOS device’s used was an iPhone 4, iPhone 5S and an iPhone 6. The iOS application extracted 40 user swipes and sent them to the server side for further computation and storage in the MongoDB database.

After collecting all this information we performed the following tests:

- **Test 1:** The objective of this test is to simulate the legit user authentication by the system. During this test we also measure the amount of true positives and false negatives calculated by the classifier, and also the necessary time to train the classifier. The test consisted in selecting the first user along with 30 of his swipes, and save the other 10. Querying the database for the remaining 13 users and their swipes, calculate the biometric features detailed by the authors of Touchalytics, train the SVM classifier and record the training time. After this the classifier passes from training mode to classification mode, and we can select the 10 swipes saved from the first user, feed it to the SVM classifier, and record the amount of TP, along with the FN.

- **Test 2:** The second test simulates a possible attacker trying to access the system by performing a certain number of swipes. The test begins by selecting a random user to be the attacker, and another user to be the victim. In similarity to the previous test, all the biometric features are calculated, and inserted into the classifier, in this case we leave the attacker out of the training phase. During the classification the swipes of the attacker are converted into the biometric features and inserted into the classifier. This tests enables the retrieval of TN and FP.

A key point that we miss on the tests performed is the fact that after some time people tend to change their their touch behavior, which affects the data collected. Either on the first time the user was standing or siting while performing the tests, or was in a hurry and without patience to make them. Taking this point into consideration we decided to divide the tests into two groups:
<table>
<thead>
<tr>
<th>User</th>
<th>Device</th>
<th>TP</th>
<th>FN</th>
<th>TN</th>
<th>FP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>iPhone 6</td>
<td>20</td>
<td>0</td>
<td>20</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>iPhone 6</td>
<td>20</td>
<td>0</td>
<td>20</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>iPhone 6</td>
<td>20</td>
<td>0</td>
<td>20</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>iPhone 5S</td>
<td>20</td>
<td>0</td>
<td>20</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>iPhone 4</td>
<td>18</td>
<td>2</td>
<td>20</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>iPhone 4</td>
<td>20</td>
<td>0</td>
<td>20</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>iPhone 4</td>
<td>13</td>
<td>7</td>
<td>20</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>iPhone 4</td>
<td>18</td>
<td>2</td>
<td>20</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>iPhone 4</td>
<td>15</td>
<td>5</td>
<td>20</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>iPhone 4</td>
<td>20</td>
<td>0</td>
<td>20</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>iPhone 4</td>
<td>15</td>
<td>5</td>
<td>20</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>iPhone 4</td>
<td>14</td>
<td>6</td>
<td>14</td>
<td>6</td>
</tr>
<tr>
<td>13</td>
<td>iPhone 4</td>
<td>16</td>
<td>4</td>
<td>20</td>
<td>0</td>
</tr>
<tr>
<td>14</td>
<td>iPhone 4</td>
<td>20</td>
<td>0</td>
<td>20</td>
<td>0</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td>249 (88.9%)</td>
<td>31 (11.1%)</td>
<td>274 (97.9%)</td>
<td>6 (2.1%)</td>
</tr>
</tbody>
</table>

Accuracy 93.4%

Table 7.1: First application test results (Best Case).

1. Best Case: The data used in the classifier for the training phase and the classification phase was collected in the same date.

2. Worst Case: The data used in the classifier for the training phase and the classification phase was collected in different dates, typically two weeks later.

By taking this point into consideration the tests simulate a more realistic approach.

### 7.3 Results

This sections presents the results of the various experiments done. The best case results are depicted on Table 7.1, and the worst case on Table 7.2.

By analyzing the best case results depicted on Table 7.1 we observe that the number of TP is 77.8% higher then the number of FN, which is a relative good result, even in this case of TN (97.9%) the results are very pleasing. The results in the worst case scenario (Table 7.2) reveal, as expected, a less percentage of TN (72.1%) in comparison to the best case scenario. One interesting result if the number of TN which is always very high, around 99%. In terms of accuracy, the best case scenario reached 93.4%, and the worst case scenario 86.1%. The training delay to train the classifier with the 14 users was 8292 milliseconds. Both tests consisted in using for the classification phase the remaining number of swipes not used during the training phase, which was around 10 swipes. It might seem like a small number, but in fact if we think about real time authentication and Touchalytics being the solo mechanism used to perform it, if ask the user 10 swipes before we can authenticate him, it might degrade user experience and the global usability of the application. Therefore we performed the same tests but now only using 5 swipes for the classification phase, and only taking into consideration the
<table>
<thead>
<tr>
<th>User</th>
<th>Device</th>
<th>TP</th>
<th>FN</th>
<th>TN</th>
<th>FP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>iPhone 6</td>
<td>0</td>
<td>20</td>
<td>20</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>iPhone 6</td>
<td>20</td>
<td>0</td>
<td>20</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>iPhone 6</td>
<td>20</td>
<td>0</td>
<td>20</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>iPhone 5S</td>
<td>20</td>
<td>0</td>
<td>20</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>iPhone 4</td>
<td>20</td>
<td>0</td>
<td>20</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>iPhone 4</td>
<td>20</td>
<td>0</td>
<td>20</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>iPhone 4</td>
<td>20</td>
<td>0</td>
<td>20</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>iPhone 4</td>
<td>0</td>
<td>20</td>
<td>20</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>iPhone 4</td>
<td>20</td>
<td>0</td>
<td>20</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>iPhone 4</td>
<td>20</td>
<td>0</td>
<td>20</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>iPhone 4</td>
<td>20</td>
<td>0</td>
<td>20</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>iPhone 4</td>
<td>19</td>
<td>1</td>
<td>20</td>
<td>0</td>
</tr>
<tr>
<td>13</td>
<td>iPhone 4</td>
<td>0</td>
<td>20</td>
<td>20</td>
<td>0</td>
</tr>
<tr>
<td>14</td>
<td>iPhone 4</td>
<td>3</td>
<td>17</td>
<td>20</td>
<td>0</td>
</tr>
</tbody>
</table>

Total | 202 (72.1%) | 78 (27.9%) | 280 (100%) | 0 (0%) |
Accuracy | 86.1%

Table 7.2: First application test results (Worst Case).

| Authentication Experiences | 280 |
|--------------------------------------------------|
| False negatives | 116 (41.4%) |
| True positives | 164 (58.6%) |

Number of Attacks | 280 |
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>True negatives</td>
</tr>
<tr>
<td>False positives</td>
</tr>
</tbody>
</table>

Accuracy | 72.9%

Table 7.3: First application test results (Worst Case 5 Classification Swipes).

worst case scenario, to better sustain the idea that the training data might not be retrieved on the same date as the classification data.

On Table 7.3 we depicted the results obtained by performing test 1 and 2 but this time only using 5 classification swipes. In comparison to using 10 swipes there was a noticeable degradation of the results, TP went down from 72.1% to 58.6%, TN from 100% to 87.1%, and finally the overall accuracy went from 86.1% to 72.9%. This results are far from satisfactory, and this was one of the main reasons we decided to pursue the design of a different iOS application as described in section 6.1.2 with the objective to better emulate a real world application usage and as a result obtain better and more realistic results.

The results obtained in the second application cannot be directly compared to the first one. The applications are very different and generate very contrasting data. The first one obligates the user to perform 40 swipes in a "laboratory" manner, as the second application does not have a minimum swipe requirement. For that reason we decided to clean the entire user database and start from the beginning, collecting new data. On this second application there was only the possibility, because of scheduling
Training users  5  
Training swipes  13  
Classification swipes  3  

Authentication Experiences  220  
False negatives  5 (2.3%)  
True positives  215 (97.7%)  

Number of Attacks  220  
True negatives  195 (88.6%)  
False positives  25 (11.4%)  
Accuracy  93.2%  

Table 7.4: Second application test results (5 users - 3 Classification Swipes).

Training users  5  
Training swipes  13  
Classification swipes  4  

Authentication Experiences  220  
False negatives  1 (0.5%)  
True positives  219 (99.5%)  

Number of Attacks  220  
True negatives  196 (89.1%)  
False positives  24 (10.9%)  
Accuracy  94.3%  

Table 7.5: Second application test results (5 users - 4 Classification Swipes).

Training users  5  
Training swipes  13  
Classification swipes  8  

Authentication Experiences  220  
False negatives  5 (2.3%)  
True positives  215 (97.7%)  

Number of Attacks  220  
True negatives  189 (85.9%)  
False positives  31 (14.1%)  
Accuracy  91.8%  

Table 7.6: Second application test results (5 users - 8 Classification Swipes).

and project deadlines, to collect data from 11 users, and perform the tests in the best scenario, where the training and classification data are from the same time, or by other words, the classifier is trained after a certain number of swipes, and then automatically passes to the classification phase. The tests were performed with a static number of training swipes (13), a diverse number of classification swipes (3, 4, and 8), and the amount of users in the classifier diverged from 5 to 10.

By observing Tables 7.4, 7.5, and 7.6, there is no substantial better results by increasing the number classification swipes, and after a certain amount of swipes, the accuracy of the system tends to converge.
<table>
<thead>
<tr>
<th>Table 7.7: Second application test results (10 users - 3 Classification Swipes).</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training users</td>
</tr>
<tr>
<td>Training swipes</td>
</tr>
<tr>
<td>Classification swipes</td>
</tr>
<tr>
<td>Authentication Experiences</td>
</tr>
<tr>
<td>False negatives</td>
</tr>
<tr>
<td>True positives</td>
</tr>
<tr>
<td>Number of Attacks</td>
</tr>
<tr>
<td>True negatives</td>
</tr>
<tr>
<td>False positives</td>
</tr>
<tr>
<td>Accuracy</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 7.8: Second application test results (10 users - 4 Classification Swipes).</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training users</td>
</tr>
<tr>
<td>Training swipes</td>
</tr>
<tr>
<td>Classification swipes</td>
</tr>
<tr>
<td>Authentication Experiences</td>
</tr>
<tr>
<td>False negatives</td>
</tr>
<tr>
<td>True positives</td>
</tr>
<tr>
<td>Number of Attacks</td>
</tr>
<tr>
<td>True negatives</td>
</tr>
<tr>
<td>False positives</td>
</tr>
<tr>
<td>Accuracy</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 7.9: Second application test results (10 users - 8 Classification Swipes).</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training users</td>
</tr>
<tr>
<td>Training swipes</td>
</tr>
<tr>
<td>Classification swipes</td>
</tr>
<tr>
<td>Authentication Experiences</td>
</tr>
<tr>
<td>False negatives</td>
</tr>
<tr>
<td>True positives</td>
</tr>
<tr>
<td>Number of Attacks</td>
</tr>
<tr>
<td>True negatives</td>
</tr>
<tr>
<td>False positives</td>
</tr>
<tr>
<td>Accuracy</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 7.10: Training delay.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of users</td>
</tr>
<tr>
<td>Time to train (ms)</td>
</tr>
</tbody>
</table>

In terms of training delay the data is depicted on Table 7.10.

By comparing the results from using 3, 4 and 8 swipes for the classification there is not much overall accuracy gained, in reality most of the results show the opposite. This systems suffer from having
multiple variables conditioning the results, for instance, the slightest change in the training data might change the results completely. Even if we don't change the data itself but normalize it in a different range, the results change. A possible analyzes on this tests and their results is that applying machine learning to this type of problem works, but there is a necessity to process the input data and find the best machine learning algorithm to the problem being solved.
Chapter 8

Conclusions

8.1 Limitations

Our final iOS application tries to achieve a higher level of resemblance to something real, something most of the smartphone users are accustomed to resort. Even with those efforts the experience remains artificial, in a sense that, most of the users if not all, did not behave in a similar matter like the application was indeed real. The fact that the experience passes through handling the mobile device to the users and ask them to perform a purchase in the application, already obligates them to do something they might not want or do not really need, which in result introduces a high disturbance in the data collected. A possible solution to conduct a better experience is to release a real application into Apple App Store, or Android's Play Store, inform the users that data is gonna be collected for study purposes, and let them use the application without constraint. This might be a problem if we take into consideration Apple's privacy agreements, with that in mind there is always the possibility to release the application privately to the university campus and ask users to use the application during a period of time.

Another limitation is the fact that every mechanism that makes use of machine learning algorithms has a high dependency on the data collected for the training phase. During our initial implementations we had some biometric features miscalculated which lead to low results. And even after fixing the calculations we were still obtaining low results, which turned out to be bad parametrization of the values used by SVM and not normalizing the data.

Finally there is the possibility that other machine learning algorithms achieve better accuracy and overall results. The authors of [34] try to solve a classification problem in a different subject, tcp and udp traffic classification, nonetheless a important point the authors mention is the fact that by applying a discretization method called Minimum Description Length criterion (Ent-MDL) [40] to the data the results of the overall accuracy using the SVM algorithm increase by 58.7% in their experience, which could apply directly to ours.
8.2 Future Work

In the matter of future work there is a possibility to increase the overall results of the system by applying a different machine learning algorithm, for instance, K-NN [11], or even by studying the impact of using unsupervised algorithms, like Clustering implementation, DBSCAN [17], and other variants. When it comes to simulate a possible attacker into the system, the methodology used might not have been the most proper. When an user is selected from the database to be the attacker and another one to be the legit user, the attacker is left out of the training phase in the classifier, in order to simulate a real attack on the classification phase, but what this really accomplishes is how similar the attacker is to any user the classifier knows. A possible solution to better simulate a attack, is to have one person perform a legit authentication and another person impersonate an attacker and carry out a shoulder surfing attack, by looking at the user swipes and try to replay them.

8.3 Conclusion

In this thesis we presented a mobile ticketing solution using HCE technology combined with tokenization strategy (section 4.2.1) and revision files (section 4.2.1) to increase overall security in case of compromise. Most solutions are proprietary and lack the comprehensive documentation of the methodology used. A major advantage of the presented system when compared to others is the fact that our solution offers transparency about the used architecture and the various elements involved.

The proposed system is able to mitigate the most common relay attacks to mobile ticketing while keeping the time necessary to validate a transportation ticket within the time to execute a NFC local communication. Furthermore, it requires only but a few changes to existing public transportation systems, thus simplifying its uptake.

Additionally we designed and implemented a system capable of interacting with NFC4sure during the mutual-authentication protocol (section 4.2.3), with the objective of identifying and authenticate an user with a mobile device.

Touchalytics was designed to work in iOS 7.1 and later versions. The implementation involved a client-side written in Objective-C, and a server-side in Node.js. Touchalytics achieved an overall 97.0% accuracy in the various tests performed in section 7.3.
References


[27] Bytemark Inc. Method and system for distributing electronic tickets with visual display, granted patent us 8494967 b2, 2012.


