TÉCNICO LISBOA

Authentication via User Profiling

André Filipe Freire Faustino

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisors: Prof. Pedro Miguel dos Santos Alves Madeira Adão

Examination Committee

Chairperson: Prof. José Luís Brinquete Borbinha
Supervisor: Prof. Pedro Miguel dos Santos Alves Madeira Adão
Members of the Committee: Prof. Nuno Miguel Carvalho dos Santos

November 2017

Acknowledgments

I would like to thank my professor Pedro Adão for giving me the opportunity of working in such an interesting idea and giving me the guidance to the development of the prototype and during the elaboration of this dissertation.

I would like to thank my parents Paulo and Margarida and my brother Pedro for supporting during my studies and in the thesis.

I would also like to thank my friends Pedro Reganha, Diogo Monteiro, João Santos and Raquel Casteleiro for helping me in finding bugs in the developed Android application and helping me on the document revision, and for all the support given to me while studying in Instituto Superior Técnico.

I would also like to thank my friends Daniel Leitão, Luís Freixinho, Marcos Faria, Catarina Costa, Rui Santos, Daniel Trindade, Rodrigo Freitas, Bernardo Andrade, Ricardo Campos and Natalino Cordeiro for the great moments that we've shared through this last five years.

Finally, I would like to thank everyone that filled the preliminary inquiry and tested my application in either test phases, since it would be impossible to verify the correctness of my solution without tests with users.

Abstract

Online authentication using only a username and a textual password has been proven flawed over years. Some solutions have been adopting different types of passwords (for example, Graphical Passwords) or adding more authentication factors such as biometrics, location and object possession.

Despite these solutions being secure, they sometimes require expensive hardware that is hard to distribute, or demand several actions from the user. In the particular case of authentication in mobile devices, where users want a fast authentication, some of these solutions are unusable.

The solution described in this work is an additional authentication factor that uses the smartphone characteristics and its user configuration in order to create user profiles that will be used to authenticate the user, requiring no actions from the user.

This way, since the smartphone is able to give all the information the system needs, the proposed solution provides a fast authentication that does not requires the user to perform any actions. The proposed solution is also as secure as existing solutions.

Since mobile devices are subject to changes such as the installed applications, memorized networks and accounts, operating system upgrades among others, profile verification must be made allowing small changes.

Therefore, the Authentication Server should evaluate whether the freshly received profile is similar enough to the stored profile using defined thresholds. If static attributes (such as the IMEI, screen resolution among others) are changed, or if a large number of attributes have been changed, then the verification fails immediately, activating the fallback mechanism defined by the third-party application.

Keywords

Online authentication, Two-Factor Authentication, mobile devices, user profile

Resumo

Autenticação online usando um username e uma password textual tem sido provada como uma solução insegura. Algumas soluções têm adoptado diferentes tipos de password ou adicionado novos factores de autenticação como biometrias, localização geográfica ou posse de objectos.

Apesar destas soluções serem seguras, requerem por vezes hardware que é caro e difícil de distribuir ou exigem demasiadas acções por parte do utilizador. No caso de autenticação online em dispositivos móveis, onde os utilizadores desejam uma autenticação rápida, estas soluções tornam-se ineficientes.

A solução descrita neste trabalho consiste num factor de autenticação que usa as características de um smartphone e a configuração introduzida pelo utilizador para criar um perfil único que vai ser usado para autenticar o utilizador, não exigindo acções extra por parte do utilizador, uma vez que o smartphone fornece ao sistema toda a informação necessária.

A solução proposta é tão segura como as soluções existentes.

Uma vez que os dispositivos móveis estão sujeitos a alterações como as aplicações instaladas, contas e redes memorizadas, actualizações do sistema operativo entre outras, a verificação dos perfis deve ser feita de forma a tolerar pequenas alterações.

Assim, o servidor de autenticação avalia as similaridades entre perfis, usando thresholds definidos. Se atributos estáticos (como o IMEI, resolução do ecrã entre outros) forem alterados ou se um grande número de atributos for alterado, a verificação do perfil falha imediatamente, activando o mecanismo de fallback definido pela entidade externa que serve de cliente.

Palavras Chave

Autenticação Online, Two-Factor Authentication, dispositivos móveis, perfis de utilizador

Contents

List of Figures

List of Tables

Acronyms

AES	Advanced Encryption Standard
API	Application Programming Interface
CPU	Central Processing Unit
FIDO	Fast Identity Online
HTTP	Hypertext Transfer Protocol
HTTPS	Hyper Text Transfer Protocol Secure
IP	Internet Protocol
IPC	Inter-Process Communication
IMEI	International Mobile Equipment Identity
IMSI	International Mobile Subscriber Identity
HMAC	Hash-based Message Authentication Code
MAC	Media Access Control
OS	Operating System
OTP	One-Time Password
RSA	Rivest Shamir Adleman
SDK	Software Development Kit
SIM	Subscriber Identity Module
SMS	Short Message Service
SSID	Service Set Identifier

TLS	Transport Layer Security
TEE	Trusted Execution Environment

Chapter 1

Introduction

Contents

1.1 Motivation

Online authentication has been done over the years using a username and a textual password. The main advantages of this system include being relatively simple, since the user does not need to carry extra devices that help him on proving his identity, and everything the user needs is to remember his username and password.

This system is also cheap to implement and distribute since there is no need to buy expensive hardware. All that is required of a server is to be able to store usernames and the matching password in plain text or its hash. The authentication phase is also quite simple: the user sends its username and password and the server only checks if the received password corresponds to the stored one for the corresponding username.

However, over the years and with the emergence of more powerful hardware which allows computers to perform faster and use more memory, this model has become vulnerable and easy to exploit. Short passwords have been considered insecure nowadays due to being easy to crack using brute force attacks or discover through phishing attacks, among other possible attacks.

As Coombs [14] pointed out, Kaspersky estimated that a phishing gang stole one billion dollars from several banks in 2014. Regarding brute force attacks, in an offline password-cracking exercise at the 2010 DEFCON Conference, the winning team (Hashcat) cracked 38000 passwords in 48 hours [20].

Social Engineering attacks are also a threat to short passwords. When the password is something personal related to the user like favorite sports club, the name of pets or a relative, birthday, among other personal information, there is a strong possibility that an attacker could discover the password by simply knowing pieces of information related to the user or finding that information in social networks.

Another problem occurs when users use the same password or variations of the same password in different services. In those cases, if an attacker discovers the used password then he will be able to gain access to all services where the user uses that same password or variations. This might be a critical problem if, for example, an attacker gains access to the password associated with the email account used to recover and redefine passwords for multiple services. In this case the rightful user may never be able to access all his accounts in the respective services.

Long randomly generated passwords are harder to guess and crack, but they are also harder to remember, especially if the user uses different passwords for different services [14]. Even though these passwords may be stored in a password manager, the master password has the same problems regarding length and discoverability. If the password is so hard to memorize that the user needs to carry a note to remember the password then there is a high probability that an attacker may obtain access to that note as well.

Some solutions to the problem related to textual passwords include the presence of a device that will give the user a one-time password to input and prove his identity. While these systems may solve the problem in desktop computers, they are hard to use in mobile devices since they require the user to carry an extra device that may get lost or stolen.

Secure online authentication in mobile devices is still an open problem with several approaches but without a definite solution. Smartphones in particular are a key component in people's lives nowadays, as they are carried everywhere and used to run a wide range of applications that access the Internet. The information contained in some smartphone applications like email or banking applications is typically personal, private and valuable. Therefore, users tend to access these applications only on their own smartphone and not on a friend's or family member's smartphone.

Some of the approaches to solve the secure online authentication problem in smartphones consist of requiring additional authentication factors, such as biometrics like fingerprint, voice or facial recognition, or the presence of some object like a smart card, a badge, or a key generator, to fully authenticate the user.

However, these solutions have some problems. Biometrics traditionally require a huge effort to falsify but once they are falsified the user may never use them again, since it is very hard or even impossible to replace them. For example, once a fingerprint is falsified the user cannot change his finger to use a different fingerprint. Another problem regarding biometrics is the quality of the sensors used to measure the biometric. In order to completely identify each user with minimum error rates the sensors used must be precise and exact, and these types of sensors are expensive [29].

Solutions that require objects being carried by the user are not ideal either, since the object can be lost, stolen or forgotten, preventing the user from being able to login again without changing the configuration of the system.

The most commonly used system consists in sending a code through Short Message Service (SMS) or generating one-time passwords sent to different applications that the user must input to verify his identity. The general idea behind these systems is to introduce a new authentication layer by generating and distributing a code that is hard to guess or falsify. This code can only be received on the device associated with the account.

However, the solution described above requires the user to perform a set of actions he would not perform otherwise. These actions include checking the received code in the specific application where the code is received, copying or memorizing it and inputting it when asked. Moreover, in some cases, to increase the security of the solution, all these actions should be performed within a short span of time (in most cases the user has a 30 second interval to input the generated code).

1.2 Proposed Solution

The proposed solution aims at solving the problem of online authentication in mobile devices using an authentication factor that is secure and requires minimum user interaction. The solution consists of a Two-Factor Authentication system where the second authentication factor is an adaptive unique profile that is composed of unique features of the smartphone and its user configuration like memorized networks, installed applications, memorized accounts, among others. In this system, user interaction would be unnecessary since the smartphone is able to provide by itself everything the system needs to authenticate the user.

The main idea behind this solution is that even though two different persons may have the same model of the smartphone, the applications installed in their smartphones will differ. Even if the applications installed are similar, the configuration of the smartphone made by the user, like which applications have stored accounts or the keyboard used will most certainly differ, creating two different profiles.

The solution consists of an authentication server and an information collector application that will collect the relevant information for the creation of the user profile. This system can be used by third-party applications, such as banking applications, in order to check the authenticity of the user trying to login. Whenever a user logs in the client application, this application calls the information collector application running in the device which will retrieve the needed information and create the profile.

The solution consists then in a new Two-Factor Authentication scheme rather than consisting in a new Online Authentication process, due to the fact that it is impossible to know exactly who is using the smartphone. It is important to note as well that the solution uses an existing fallback mechanism since implementing a new fallback mechanism is out of the scope of this work.

For each retrieved attribute a hash is calculated, with the created profile being then a set of the calculated hashes. An Hash-based Message Authentication Code (HMAC) of the profile is generated using a symmetric Advanced Encryption Standard (AES) key that is shared with the server and stored safely in the smartphone's Trusted Execution Environment (TEE) (whenever possible). Both the profile and the HMAC of the profile are sent through a secure channel to the Authentication Server alongside with a digital signature of the created profile, to prevent the message from being tampered.

The authentication server will then verify if the digital signature is valid and the integrity of the received information by checking if the received profile HMAC matches the received profile. If the received information is valid, the server checks if it matches the stored information contained in the user profile. If it matches, the user is now authenticated and able to use the client application. If it doesn't match, the third-party application should activate its defined fallback authentication mechanism (in the prototype the default fallback mechanism is the verification using an SMS code).

Due to smartphones being customizable, the proposed solution should be able to deal with changes in the profile. If the profile is too strict, whenever a user downloads a new application, for example, a new profile would be required, triggering the fallback mechanism and the new authentication factor would never work.

To overcome the problems with the changing information on the smartphone, the authentication server will accept the profile if the majority of the information remains unchanged. Part of the profile is composed of static attributes (information that is impossible to change) like the International Mobile Equipment Identity (IMEI) number or the International Mobile Subscriber Identity (IMSI) number, and the other part is composed of dynamic attributes which may suffer changes in a regular use of the smartphone.

If a static attribute or a significant part of the dynamic attributes is changed, the user must acquire a new profile, activating the fallback mechanism of the client application. This is required because with such a great variation, it is impossible to know whether the user changed a large number of applications or an attacker is trying to impersonate a legitimate user.

Whenever the server accepts the profile of a user, the server will update the stored profile with the new information, making sure that the stored profile always contains the updated information of the smartphone. For example, considering the list of installed applications, if a user installs a new application from one login to another and the profiles match, the server will store the new list of installed applications.

The main advantage of retrieving attributes from the smartphone in order to create the unique smartphone profile consists in adding a new layer of required information that provides security to the solution. A simpler solution that would consist in generating a key that is securely stored in the device would have the drawback of the possibility of the key being exported and used in a different device. The proposed solution does not have this advantage, since even if an attacker would become able to export the key, he would still need to create a similar profile, which may not be an easy task.

1.3 Objectives

The objectives of this work are to introduce a new Two-Factor Authentication scheme that is simultaneously as fast and as secure as existing solutions, and that requires few actions from the user when compared to existing solutions.

Considering that resources, namely CPU, memory, network bandwidth and battery, are limited in smartphones, the solution should avoid using more resources than necessary. More concretely, the solution should not significantly increase the CPU and memory usage, the communications required to authenticate a user should not take a significant part of the network traffic and the battery life should not significantly decrease by using the information collector application.

1.4 Contributions

The major contributions for the online authentication problem of the solution described in the Section above are the following:

- First solution known so far to use a profile composed of device characteristics and user configuration in order to authenticate the user.
- Two-Factor Authentication solution that requires no explicit interaction from the user in the best cases and requires the same actions as any other solution in the worst case.
- Prototype implemented in Android Operating System that was tested with real users.

1.5 Organization

This document is organized in the following way:

- Chapter 1 is the Introduction, which introduces the motivation, the goals, the solution, the objectives of this work and the contributions done by this work.
- Chapter 2 is the Related Work, which gives an overview of the problems related to Textual Passwords and the limitations of existing solutions. This chapter also introduces some state of the art solutions that are the base of some ideas implemented in the proposed system.
- Chapter 3 is the Solution Overview, which introduces some background about Android, Android Keystore (which is used to guarantee the security of the solution), Trusted Execution Environments and some cryptographic primitives and security protocols such as the HMAC and HTTPS. This chapter then describes the proposed system architecture and ends up with the attacker models used to evaluate the system.
- Chapter 4 is the Evaluation, which evaluates the usability of the system when tested with real-world users and the security of the system.
- Chapter 5 is the Future Work, which describes some future enhancements that can be done to the proposed system.
- Chapter 6 is the Conclusion, which outlines the results and presents the conclusions of the implementation of the system and its evaluation.

Chapter 2

Related Work

Contents

This Section describes not only the state of the art of online authentication but also existing systems and techniques that are relevant to the solution.

Regarding the state of the art of online authentication, this Section will address the problems and most common attacks against textual passwords and then alternatives to textual passwords will be presented. These alternatives consist on Biometrics, Graphical Passwords and Location Based Systems. The main focus is to point out the advantages and disadvantages of each type of solution and give an insight on diverse systems related to that type of solution.

Regarding systems and techniques that are relevant for the solution presented in this paper, the main topics that will be described are Two-Factor Authentication (and in detail some disadvantages of SMS Code Authentication Factor), Behavioral Biometrics and User and System Profiling. The main focus is to present some key ideas of these techniques and point out the contributions of these ideas to the presented solution.

Finally, at the end of the chapter some attacker models and common attacks against Android devices, that are relevant to the problem, will be presented. These models were presented by different authors, with realistic assumptions and capabilities given to attackers. The models described in this chapter will be used or adapted to define the attacker models required to evaluate the security of the system.

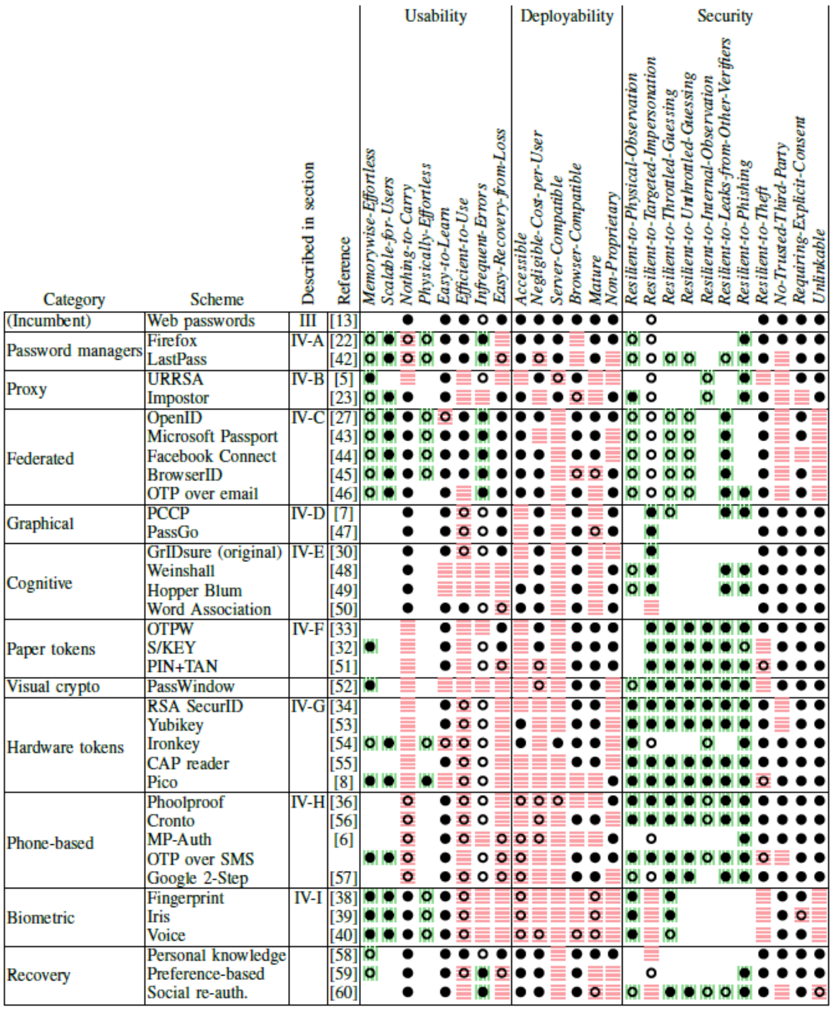
2.1 Textual Passwords Strength

Papers [20] [11] [29] [9] related to online authentication are unanimous that textual passwords are becoming a security problem due to having lots of weaknesses and flaws that can be explored by a malicious attacker or system.

A repeated idea is that passwords, especially short passwords, are weak, have lots of patterns, are vulnerable to lots of attacks and are easy to crack [20]. Long passwords are hard to remember, especially if the user has many different passwords.

The most common attacks to passwords are: Finding written passwords (in post-its, for example); Guessing the password through social engineering or by knowing something about the user (pets, children, favorite car, sports club); Shoulder Surfing where the attacker is behind the user and sees what keys are being pressed to input the password; Keystroke logging; Virtual keyboards/mouse that record the information; Screen scraping used alongside keystroke logging; Brute force attacks using rainbow attacks, hash tables, salt; Password Explosion attacks [11] [29].

Bonneau et al [9] developed a comparative study of 36 different authentication schemes divided by 12 categories such as Graphical Passwords, Textual Passwords, Hardware Tokens, Biometrics, Phone-Based Authentication Schemes, to name a few. Each authentication scheme was analyzed considering several benefits from three big categories such as Usability, Deployability and Security. The analysis



				Usability								Deployability						Security										
Category	Scheme	Described in section	Reference	*Memorywise-Effortless*	*Scalable-for-Users*	*Nothing-to-Carry*	*Physically-Effortless*	*Easy-to-Learn*	*Efficient-to-Use*	*Infrequent-Errors*	*Easy-Recovery-from-Loss*	*Accessible*	*Negligible-Cost-per-User*	*Server-Compatible*	*Browser-Compatible*	*Mature*	*Non-Proprietary*	*Resilient-to-Physical-Observation*	*Resilient-to-Targeted-Impersonation*	*Resilient-to-Throttled-Guessing*	*Resilient-to-Unthrottled-Guessing*	*Resilient-to-Internal-Observation*	*Resilient-to-Leaks-from-Other-Verifiers*	*Resilient-to-Phishing*	*Resilient-to-Theft*	*No-Trusted-Third-Party*	*Requiring-Explicit-Consent*	*Unlinkable*
(Incumbent)	Web passwords	III	[13]			●		●	●	○	●	●	●	●	●	●	●		○						●	●	●	●
Password managers	Firefox	IV-A	[22]	🟩○	🟩●	🟥○	🟩○	●	●	🟩●	🟥	●	●	●	🟥	●	●	🟩○	○					🟩●	●	●	●	●
	LastPass		[42]	🟩○	🟩●	🟥○	🟩○	●	●	🟩●	🟥○	●	🟥○	●	🟥	●	🟥	🟩○	○	🟩○	🟩○		🟩○	🟩●	●	🟥	●	●
Proxy	URRSA	IV-B	[5]	🟩●		🟥		●	🟥	○	🟥	🟥	●	🟥○	●	🟥	🟥		○			🟩○		🟩●	🟥	🟥	●	●
	Impostor		[23]	🟩○	🟩●	●		●	🟥	🟥	●	●	🟥	●	🟥○	🟥	●	🟩●	○			🟩○		🟩●	●	🟥	🟥	●
Federated	OpenID	IV-C	[27]	🟩○	🟩●	●	🟩○	🟥○	●	🟩●	●	●	●	🟥	●	●	●	🟩○	○	🟩○	🟩○		🟩●		●	🟥	●	🟥
	Microsoft Passport		[43]	🟩○	🟩●	●	🟩○	●	●	🟩●	●	●	🟥	🟥	●	●	🟥	🟩○	○	🟩○	🟩○		🟩●		●	🟥	●	🟥
	Facebook Connect		[44]	🟩○	🟩●	●	🟩○	●	●	🟩●	●	●	●	🟥	●	●	🟥	🟩○	○	🟩○	🟩○		🟩●		●	🟥	🟥	🟥
	BrowserID		[45]	🟩○	🟩●	●	🟩○	●	●	🟩●	●	●	●	🟥	🟥○	🟥○	●	🟩○	○	🟩○	🟩○		🟩●		●	🟥	●	🟥
	OTP over email		[46]	🟩○	🟩●	●		●	🟥	🟩●	●	●	●	🟥	●	🟥	●	🟩○	○	🟩○	🟩○		🟩●	🟩●	●	🟥	●	🟥
Graphical	PCCP	IV-D	[7]			●		●	🟥○	○	●	🟥	●	🟥	●	🟥	●		🟩●	🟩○			🟩●	🟩●	●	●	●	●
	PassGo		[47]			●		●	🟥○	○	●	🟥	●	🟥	●	🟥○	●		🟩●						●	●	●	●
Cognitive	GrIDsure (original)	IV-E	[30]			●		●	🟥○	○	●	🟥	●	🟥	●	🟥	🟥		🟩●						●	●	●	●
	Weinshall		[48]			●		🟥	🟥	🟥	🟥	🟥	●	🟥	●	🟥	●	🟩○	🟩●				🟩●	🟩●	●	●	●	●
	Hopper Blum		[49]			●		🟥	🟥	🟥	🟥	●	●	🟥	●	🟥	●	🟩○	🟩●				🟩●	🟩●	●	●	●	●
	Word Association		[50]			●		●	●	○	🟥○	●	●	🟥	●	🟥	●		🟥						●	●	●	●
Paper tokens	OTPW	IV-F	[33]			🟥		●	🟥	🟥	●	🟥	●	🟥	●	●	●		🟩●	🟩●	🟩●	🟩●	🟩●	🟩●	●	●	●	●
	S/KEY		[32]	🟩●		🟥		●	🟥	○	●	🟥	●	🟥	●	●	●		🟩●	🟩●	🟩●	🟩●	🟩●	🟩○	🟥	●	●	●
	PIN+TAN		[51]			🟥		●	🟥	○	🟥○	🟥	🟥○	🟥	●	●	●		🟩●	🟩●	🟩●	🟩●	🟩●	🟩●	🟥○	●	●	●
Visual crypto	PassWindow		[52]	🟩●		🟥		🟥	🟥	🟥	🟥	🟥	🟥○	🟥	●	●	🟥	🟩○	🟩●	🟩●	🟩●	🟩●	🟩●	🟩●	🟥	●	●	●
Hardware tokens	RSA SecurID	IV-G	[34]			🟥		●	🟥○	○	🟥	🟥	🟥	🟥	●	●	🟥	🟩●	🟩●	🟩●	🟩●	🟩●	🟩●	🟩●	●	🟥	●	●
	Yubikey		[53]			🟥		●	🟥○	○	🟥	●	🟥	🟥	●	●	🟥	🟩●	🟩●	🟩●	🟩●	🟩●	🟩●	🟩●	●	🟥	●	●
	Ironkey		[54]	🟩○	🟩●	🟥	🟩○	🟥○	🟥○	○	🟥	●	🟥	●	●	●	🟥	🟩●	○			🟩○		🟩●	●	●	●	●
	CAP reader		[55]			🟥		●	🟥○	○	🟥	🟥	🟥	🟥	●	●	🟥	🟩●	🟩●	🟩●	🟩●	🟩●	🟩●	🟩●	●	●	●	●
	Pico		[8]	🟩●	🟩●	🟥	🟩●	🟥	🟥○	○	🟥	🟥	🟥	🟥	🟥	🟥	●	🟩●	🟩●	🟩●	🟩●	🟩●	🟩●	🟩●	🟥○	●	●	●
Phone-based	Phoolproof	IV-H	[36]			🟥○		●	🟥○	○	🟥	🟥○	🟥○	🟥○	🟥	🟥	●	🟩●	🟩●	🟩●	🟩●	🟩○	🟩●	🟩●	●	●	●	●
	Cronto		[56]			🟥○		●	🟥○	○	🟥	🟥	🟥○	🟥	●	●	🟥	🟩●	🟩●	🟩●	🟩●	🟩○	🟩●	🟩●	●	●	●	●
	MP-Auth		[6]			🟥○		●	🟥○	🟥	🟥○	🟥○	🟥○	🟥	🟥	🟥	●		○					🟩●	●	●	●	●
	OTP over SMS			🟩●	🟩●	🟥○		●	🟥	○	🟥○	🟥○	🟥	🟥	●	●	●	🟩●	🟩●	🟩●	🟩●	🟩○	🟩●	🟩●	🟥○	🟥	●	●
	Google 2-Step		[57]			🟥○		●	🟥○	○	🟥○	🟥○	🟥	🟥	●	●	🟥	🟩○	○	🟩●	🟩●		🟩●	🟩●	●	●	●	●
Biometric	Fingerprint	IV-I	[38]	🟩●	🟩●	●	🟩○	●	🟥○	🟥	🟥	🟥○	🟥	🟥	🟥	🟥○	🟥	🟩●	🟥	🟩●					🟥	●	●	🟥
	Iris		[39]	🟩●	🟩●	●	🟩○	●	🟥○	🟥	🟥	🟥○	🟥	🟥	🟥	🟥○	🟥	🟩●	🟥	🟩●					🟥	●	🟥○	🟥
	Voice		[40]	🟩●	🟩●	●	🟩○	●	🟥○	🟥	🟥	🟥○	🟥○	🟥	🟥○	🟥○	🟥	🟩●	🟥	🟩○					🟥	●	●	🟥
Recovery	Personal knowledge		[58]	🟩○		●		●	●	○	●	●	●	🟥	●	●	●		🟥						●	●	●	●
	Preference-based		[59]	🟩○		●		●	🟥○	🟩●	🟥○	●	●	🟥	●	🟥	🟥		○					🟩●	●	●	●	●
	Social re-auth.		[60]			●		●	🟥	🟩●	🟥	●	●	🟥	●	🟥○	🟥	🟩○	🟥	🟩●	🟩●	🟩○	🟩○	🟩●	●	🟥	●	🟥○

●= offers the benefit; ○= almost offers the benefit; *no circle* = does not offer the benefit.
🟩= better than passwords; 🟥= worse than passwords; *no background pattern* = no change.
We group related schemes into categories. For space reasons, in the present paper we describe at most one representative scheme per category; the companion technical report [1] discusses all schemes listed.

Figure 2.1: Bonneau et al results concerning Authentication Schemes comparison [9]

of each authentication factor consisted on verifying whether the evaluated authentication scheme provided or not a benefit and whether the selected authentication factor was better or worse than textual passwords in that particular benefit. The results of this study can be seen in Figure 2.1.

2.2 Biometrics

Biometrics, as described before, are metrics related to human characteristics. The most common types of biometrics are: Fingerprint, Hand Geometry, Voice, Retina, Iris, Signature and Face [11].

However, in online authentication, more concretely mobile online authentication, only fingerprints, voice and face are usable.

Bhagavatula et al [7] conducted a lab study that aimed to compare facial recognition (using Android's face unlock), fingerprints (using Iphone's fingerprint unlock) and conventional numerical pins. The major conclusions of this study consist on the user's perception that authentication using biometrics is more secure than using a pin (with fingerprints being considered more secure than facial recognition), and based on the user's perception that biometrics are easy and convenient to use. The main registered problems with biometrics consist on facial recognition being hard to use when the room is dark, and fingerprints being somehow hard to use when the user is walking or when the fingers are sweaty or dirty.

Christian Holz and Frank Bentley in [24] proposed a system that introduces fingerprints as a second authentication factor. In this work online authentication is performed on a computer and the authors point out the increased complexity added by the need of inputting a temporary code, claiming and later verifying with users that the need of receiving and inputting a temporary code is what keeps most users from using Two-Factor Authentication systems. The users do not mind having the smartphone around to use as second factor but waiting for the code to arrive and inputting them on the browsers on computer requires too many actions that the users do not want to perform.

To face that problem and simplify the login, while keeping the same security level of a Two-Factor Authentication system, Holz and Bentley introduce a system where the user inputs his username and password in his computer browser and receives a request on his trusted smartphone linked with the account to input a fingerprint using a scanner sensor embedded in the home button of the phone. This system was integrated with Yahoo email, replacing the login with this authentication procedure. Later, the new system was evaluated with a set of users that were interviewed about their usage of email, how sensitive is the information contained in their email, how simple the new system was, if they preferred the new system or the old one, if they felt more secure using the new system, among other questions.

Summarizing the results of the interviews done by Holz and Bentley, users claimed that their email inboxes contain very sensitive data worth protecting. Users found the new system simple, convenient and easy to use. The system alleviated the fear of impersonation. However, some users also noted that the system needed a fallback alternative since the smartphone is not always reachable.

Despite this work being focused on authenticating a user using a computer browser it contains some key conclusions that were considered in the development of the presented solution. Users consider information contained in critical applications like email sensitive and worth protecting, which is a relevant conclusion since smartphones usually have installed similar critical applications. Another important remark is that users do not want to perform lots of actions on their smartphones, preferring simpler authentication methods over more secure and complicated authentication methods like Two-Factor Authentication.

2.3 Graphical Passwords

Graphical passwords are an alternative to the traditional and problematic textual passwords since images are easier to memorize than text. Graphical passwords are also resistant to attacks that compromise the security of textual passwords like brute force and dictionary attacks. Graphical passwords also have a clear advantage related to biometrics and token because unlike those systems it doesn't require extra hardware it only depends on what the user remembers [27] [29].

Sahu et al [29] categorize graphical passwords in two categories:

- Recognition based - users are displayed a group of images and for authentication the user has to click on the correct images. For more security, the ordering of the selection of images can also be used. This scheme is vulnerable to mouse tracking and replaying.

- Recall based - users are asked to reproduce something that had been created during the registration phase. For example, a user selects a pattern during registration procedure and has to remember and enter that pattern whenever access is desired.

Sahu et al also propose a system that resists to Shoulder Surfing attacks. The system is a mixture of textual passwords, recognition based graphical passwords and recall based graphical passwords. At the time of registration, the user will select a picture, and chose some points on it. For each point, he will select a pattern. During authentication phase the user will select the points in any order, and according to the point selected he will select the pattern by writing 0 or 1 in a textbox.

Dunphy et al [18] describe two common attacks on graphical passwords:

- Lunchtime attack - an attacker tries to compromise the device protection over time, through physical access, while the legitimate user is absent. There are two classes of attackers: naive attackers who can only make random guesses, and knowledgeable attackers who by means of shoulder surfing, or another eavesdropping technique, gained some knowledge.

- Intersection attack - the frequency of an image appearing at login can be used to determine its role as either a key or a decoy.

Dunphy et al also state the importance of having variation in the key images. To prevent patterns in frequency, the variation in the decoy images should be exactly the same. This is done by randomly selecting keys and decoys from larger, fixed portfolios.

A potential problem in graphical passwords is the source of the images. Images can have two sources: stock images that come from online sources or images that come from the user's personal collection. If user images are used as authentication images and stock photos as decoy, an attacker may easily identify the decoy images through social engineering attacks or due to the fact the quality of stock photos is usually higher.

Some image characteristics are also problematic, creating the need of filtering images in these graphical password systems. The most common problems are:

- Unmemorable Images - images that contain few instances of visual content for the user to remember. Examples: photographs taken in low light, images containing excessive movement.
- Visually Similar Images - images that share a theme or contain the same people. The user is likely to focus their memory strategy on this generalization, creating conflicts with other images.

Gurav et al [21] present an overview of different graphical password schemes, naming some advantages and disadvantages of each scheme:

- Image based scheme - the user has to select points in an image or some images in a set. Advantages: given that the password consists on a set of images, it is easy to remember. Disadvantages: this scheme is a very long process that consumes user's time.
- Grid based scheme - the graphical password is a grid background. Advantages: no need on storing a graphical database at server side. Disadvantages: sequence could be changed or grids might be different.
- Triangle scheme - users have to select points forming a particular triangle. Advantages: the display is very crowded making it hard for an attacker to guess the password. Disadvantages: with the convex surface, the assigning process takes more attempts to be successful.
- Hybrid textual authentication - users have to rate each color using a number, creating a color sequence. Advantages: the user only has to remember the rating of each color. Disadvantages: somewhat difficult to remember colors with sequence.
- Signature based scheme - user signature's is used as the password. Advantages: Signature are very hard to copy, a small mistake in signature can deny the access. Disadvantages: remembering the grid of a signature is hard.

While these systems are good alternatives to textual passwords, some key disadvantages make these types of systems not so secure on smartphones as other alternatives like biometrics. Picking images and points or drawing patterns is a time-consuming task that requires undesired user interaction, which is a problem considering that users want mobile online authentication to be simple and fast.

2.4 Location

One possible solution to the online authentication problem is to use location information to check if the identity of the user is legitimate. These solutions are rather difficult to get widely deployed since they usually require a specially designed infrastructure and special devices that can be used to determine their locations.

Zhang et al [37] proposed a system that uses smartphones to retrieve location information that will be used in location-based authentication and authorization schemes.

The system proposed by Zhang et al consists on a server called Location-based ID (LBID) which stores users' location information and authorization policies, an Authentication Server, an Authorization Server and an application called Location-based Client (LBC) that will run on the smartphone. The LBC application is capable of collecting location information from trusted location providers and providing user interfaces to register, store and manage location data.

When a user registers to the system, he must specify a pin that will be used to encrypt and decrypt the private key that will be used to encrypt the location messages and provide identity information such as a username and a password. After the server stores this information, the server will send a random number to the user, which the user has to input after receiving it. Then, the user can specify some authorization policies that will be stored in the LBID server.

During the authentication phase, the user will input his username and password, and after being asked to input his PIN, his location will be sent to the LBID server which will verify if that location matches the stored location.

Zhang et al then note that there are several attacks that could be executed in order to spoof the location provided by a smartphone. In order to prevent location spoofing, the solution developed combines different techniques, taking advantage of their strengths in order to verify the location with enough confidence. The location verification mechanism takes into account the following parameters:

1. Two sets of location coordinates from two different location sources;
2. Internet Protocol (IP) address of the client (smartphone);
3. MAC (Media Access Control) address of a nearby access point with the strongest signal.

In order to use these parameters to ensure the legitimacy of the user, Zhang et al system retrieves location information from two different sources, checking if the received location information overlaps or not. If the locations overlap, then the system calculates the position of the retrieved IP address, since the location area obtained from IP address usually covers a bigger area. If both location coordinates are not contained within the larger range of area from the IP address, then it is good indication that one or both of the location coordinates are inaccurate and may have been spoofed.

If everything checks, the MAC address of the access point with the strongest detected WiFi signal is captured and compared to the one saved during the registration process. If the two values do not match the verification process fails and stops. On the other hand, if the values match, the verification process succeeds and becomes completed.

While the proposed system by Zhang et al doesn't require the user to carry more than his smartphone with him, authentication will only be possible in certain locations making it impractical if the user goes on a trip and needs to access the system, for example.

2.5 Two-Factor Authentication

Typically, there are three categories of authentication factors: something that the user knows (passwords), something that the user has (physical tokens that are carried everywhere) or something that the user is (biometrics). Nowadays, a secure system should use at least two of those three factors, hence the general term of Two-Factor Authentication that is used to characterize thousands of systems.

Aloul et al [6] address the Two-Factor Authentication problem by pointing out that carrying tokens or smart cards has two main disadvantages: these objects are expensive to build and distribute among all the users of the system; a user may lose, break or get his token stolen. Then the user will have to pay for a new token and wait for it to arrive, if no problems occur within the distribution of the token.

To overcome these problems and since smartphones are carried with users every day, Aloul et al proposed a system that use smartphones as security tokens. The system consists on generating a One-Time Password (OTP) using factors like the smartphone's IMEI and IMSI number, a username, a pin and a timestamp. Aloul et al's system contains two modes of operation:

- Connection-Less Authentication System: The client and the server generate the OTP without being connected to each other. Since the server knows the factors used to generate the OTP, the same OTP will be generated by both client and server.

- SMS-Based Authentication System: The mobile phone request the OTP directly from the server, sending some unique information about the user to the server. If the information matches the stored information by the server, the server returns a randomly generated OTP to the mobile phone which the user will have to input within a given amount of time.

Despite the two operation modes seeming interesting, this system looks problematic since the OTP is generated using a timestamp and the same static factors. An attacker might become able to discover these factors using brute force attacks, becoming able to falsify the passwords. The size of the password being defined by the user might also be problematic. If the user defines a short size for the password the password becomes easy to guess.

Acharya et al [4] proposed a similar system that also uses OTPs as a second authentication factor and has the same operation modes (connection-less and SMS-based). The main differences reside in the fact that the factors in the work of Acharya et al consist on the IMSI number, a pin, a timestamp, a username provided by the server owner and the date of birth of the user. These factors, excluding the IMSI number, are then concatenated in a string that is given as input to a SHA1 algorithm. The result is then converted to a character string and displayed to the user, who must input this OTP in the system.

The work of Acharya et al has the same drawbacks of the system proposed by Aloul et al. The static factors used to create the OTP can be discovered by an attacker using brute force attacks. The factors used by Acharya et al are also vulnerable to social engineering attacks, since they consist of personal information about the user, namely the IMSI number and the date of birth of the user.

Harini et al [22] developed a system which they called 2CAuth that uses QR codes in an OTP authentication protocol instead of SMS messages, claiming that distributing OTPs using SMS is not practical due to the SMS transmission delay.

Not every SMS is guaranteed to be delivered within an acceptable timeframe. This happens since SMS traffic is not sent point to point but instead it is queued and then sent on to the required network cell where it is again queued and finally sent to the end-user's phone. The late delivery of an OTP contained in an SMS text message can be problematic in the case of a critical login, since the user will not have access to critical enterprise resources if the SMS text message doesn't arrive on time.

To overcome the SMS transmission delay problem, 2CAuth system consists on generating an Rivest Shamir Adleman (RSA) pair and issuing a corresponding smartcard. During the authentication phase, a random number is selected and an asymmetric challenge is generated, only the smartcard with the correct RSA pair will return the right value. Then, another random number is selected and encrypted into a QR code. The user should capture the QR code and decrypt it, with the result being typed into a web application. If the input matches the random number selected, the user is now authenticated.

2CAuth system has a major disadvantage since it requires too many user actions which is undesirable. Capturing the QR code and waiting for the system to decrypt it could be a lengthy process as well. This system also has another problem, if by any reason an attacker is able to read the RSA keys from the smartcard or if the attacker is able to discover the RSA keys a new smartcard must be issued and distributed, which will cost money and time. The system also looks vulnerable to Man-in-the-Middle attacks.

Dmitrienko et al [17] conducted a study on the security of Two-Factor Authentication implementations of worldwide Internet service providers like Facebook, Google, Twitter and Dropbox. While this study is directed to using a computer, and receiving in the smartphone an OTP that should be inputted in the computer's browser it still gives an insight on bad implementations of OTP with real-world examples and describes real-world malware that affects Two-Factor Authentication mobile schemes.

Dmitrienko et al found the following problems concerning OTPs in Two-Factor Authentication schemes:

- Low-Entropy OTPs - Facebook OTPs were not possible to analyze due to accounts being blocked, Twitter and Dropbox OTPs passed standard randomness tests but Google OTPs were noted never to start with a zero, which reduces the entropy of the generated OTPs in 10%.
- Lack of OTP Invalidation - Google repeatedly creates the same OTP if the Two-Factor Authentication process is not completed. OTPs are only invalidated by Google after an hour or if the Two-Factor Authentication process is successfully completed. This fact can be exploited by attackers to capture one OTP and prevent the user from inserting it, allowing the attacker to use the captured OTP in a different session.
- Two-Factor Authentication Deactivation - After the user logs in, Google and Facebook allow the user to deactivate the Two-Factor Authentication system without additional information. Twitter and Dropbox require the username and the password to deactivate the system. Since all these schemes do not generate an OTP to securely allow the user to deactivate the Two-Factor Authentication system, an attacker can exploit this fact to deactivate the system.
- Two-Factor Authentication Recovery Mechanisms - Since the OTP is generated and sent to the smartphone which may be stolen, lost or forgotten a recovery system is required to allow the user to authenticate himself. Twitter was noted to not provide any recovery mechanism, Dropbox uses a 16-symbol string that is stored in the account settings and Google and Facebook generate a list of 10 OTPs that are stored in the account settings. Facebook also requires the login credentials, where Dropbox and Twitter do not require any additional information. An attacker that found himself able to hijack the session will be able to retrieve the recovery strings and use it to define a different device, controlled by the attacker.
- OTP Generator Initialization Weakness - whenever the OTPs are generated client-side, transmitting it requires pre-shared secrets between client and server. One example of a system that generates OTPs client side is Google Authenticator, which is used by Google, Facebook and Outlook, among others. In Google Authenticator initialization phase the service provider generates a QR code that is displayed on the user's computer and must be captured with the user's smartphone. This QR code contains valuable information like the type of the scheme, service and account identifier, the length of the OTPs generated, and the shared secret in clear text. If an attacker can capture the QR code he will be able to read all the valuable information contained there since it is stored in clear text and then use it to generate valid OTPs.

Regarding malware that targets mobile Two-Factor Authentication schemes, Dmitrienko et al conducted an analysis that resulted in identifying 207 samples of distinct types of malware comprising 10 malware families capable of reading SMSs. Furthermore 4 malware families were identified of being capable of sending SMSs to hard-coded phone numbers.

While the results of the study conducted by Dmitrienko et al seem frightening, the major problems consist on generating low-entropy OTPs. Generating OTPs client side turns out to be problematic as well as a general lack of security in phases like the invalidation of the OTP and the recovery mechanism. However, as Dmitrienko et al noted, most of these problems can be solved by using secure channels to transmit information, such as HTTPS or HTTP, through a secure VPN using secure hardware capabilities of the smartphone such as ARM TrustZone, looking for suspicious installed applications or suspicious packets being sent through the network, among other countermeasures.

Schrittwieser et al [31] studied the security of Android messaging applications analyzing some potential problems concerning Two-Factor Authentication schemes that use SMS messages and how these problems are dealt with in several messaging applications. This study focuses on five attack vectors: Authentication Mechanism and Account Hijacking, Sender ID Spoofing/Message Manipulation, Unrequested SMS/phone calls, Enumeration and Modifying Status Messages.

Considering only the first attack vector it is the only one relevant to the problem, Schrittwieser et al concluded that most of the evaluated applications have flaws in their Authentication mechanisms which allows attackers to easily hijack accounts. The most common errors consisted in generating the code in the client side with the client sending it to the server via HTTPS so that the server would send an SMS with the code and the server sending the code via both HTTPS and SMS. These errors allow Man-in-the-Middle attacks, where the attacker starts the communication with the server using the victim's phone number and then eavesdrops on the communication between server and victim in order to get the code and hijack the victim's account.

These conclusions show that despite SMS verification schemes being secure, some misimplementated algorithms that aim to use these schemes make applications using these schemes vulnerable to several types of attacks. Implementing a Two-Factor Authentication that sends verification codes via SMS requires a good planning and testing phase otherwise the system will be even less secure than alternative solutions.

Rob Coombs [14] described how ARM TrustZone can be used with Fast Identity Online (FIDO) to replace textual passwords. Coombs started to note that smartphones are vulnerable to malware, which would be allowed to read and write information outside of its sandbox. An attacker with access to the smartphone would be allowed to perform further attacks, such as copying data (even if encrypted) and then perform offline attacks to the encrypted data. ARM TrustZone aims at mitigating these attacks by storing information and executing code inside its Trusted Execution Environment.

Coombs then explains that the ARM TrustZone can be used by FIDO to provide several security requirements such as ensuring the integrity of the device, securely store keys and maintain the integrity and confidentiality of sensitive processes. Coombs ends his paper stating that further enhancements to the ARM TrustZone would allow the system to protect the touchscreen inputs (securing that way the PINs that are inserted) and protect the display output.

The ideas introduced by Coombs are ideal to ensure the security of the developed system. The implementation of Android Keystore tries to use the ARM TrustZone whenever the smartphone hardware supports it to securely store cryptography keys in the Secure World and securely execute cryptographic primitives. In the particular case of our proposed system, the ARM TrustZone will be used through the implementation of Android Keystore to store the symmetric keys used to create the profile and to securely create digital signatures that will ensure the integrity and the non-repudiation of the messages sent.

2.6 Behavioral Biometrics

Behavioral biometrics, as described by Deutschmann et al [16], is a measurable behavior used to recognize or verify the identity of a person. Behaviometrics focuses on behavioral patterns rather than physical attributes.

That way, behavioral biometrics utilize certain characteristics of the user's behavior in order to create a finger print of the user that could be used to uniquely identify the user. BehavioSec's system, as described by Deutschmann et al [16], focuses essentially in monitoring user inputs, taking into account not only how the user inputs the information but also some characteristics like the native language in which the user types or the layout of the keyboard. The objective of this system is to detect in real time anomalies in the user input behavior and preventing the access to the system in that case within seconds.

While the system presented has some limitations regarding the biometrics used to identify the users, the idea of monitoring the system in real time to prevent intruders from using it is interesting and could be adopted in the proposed solution in order to detect thefts or someone other than the regular user using the smartphone. Also, some behavioral characteristics are interesting and should be integrated in the profile to make it less dependent on the hardware of the smartphone but also dependent on its user configuration.

DARPA Active Authentication Program described by Guidorizzi [20] aims to validate BehavioSec work on behavior biometrics and extend it to include several types of behavior biometrics in order to use this type of biometrics in online authentication. The different types of behavior biometrics are composed of:

- Neurocognitive patterns which aim to develop digital cognitive fingerprints from various biometric sources;

- User Search patterns that identify the user using his patterns for searching for information on the computer, focusing on the high volumes of decoy document touches placed in the file system;
- Stylometry that uses traditional stylometric methods to validate whether it is the rightful user typing or an attacker trying to mimic the typing methods of the rightful user;
- Covert Games introduce patterned system aberrations that the user intuitively learns;
- Screen Interface uses spatio-temporal screen fingerprints to identify the user;
- Behavioral Web Analytics identifies the user looking at this Web browsing activities, including semantic (what kind of web pages are visited) and syntactic session features.

While the objective of this program is to use behavioral biometrics in online authentication, the biometrics suggested are strongly focused on desktop environments and even though the core idea is interesting, they are not so effective to use in mobile environments.

Trojahn and Ortmeier [33] proposed the usage of keystroke dynamics as second authentication factor in mobile online authentication. The touchscreens existing in smartphones can provide specific data of fingertip pressure, fingertip size or position where a key has been hit.

Trojahn and Ortmeier claim that an authentication method that would rely on using keystroke dynamics also has the advantage on not needing additional hardware, since the touchscreen of the mobile device can retrieve all the needed information. In addition, such an authentication method would be transparent to the user, as the keystroke pattern could be recorded while entering the password.

Trojahn and Ortmeier identified and described new features offered by smartphones and tablets that are useful and interesting to identify unique keystroke dynamics such as:

- Pressure during typing: Touch pads have sensors to get information on how much a person has pressed the surface of the mobile device.
- Finger tip size: Some people short press the keys with only a tiny part of their fingertip normally without much pressure while others are use the whole fingertip. The size of the pressed area can be measured.
- Physics of the mobile device: Means how the user holds his device and which is the preferred hand. The majority of people in the world use one hand most of the times to write and normally carry the smartphone also in one hand. It should be possible to evaluate in which hand the user is carrying the smartphone (by evaluating the speed in the corner of the keyboard) and whether he is typing with the same hand or using the other. The angle in which the user is holding the device could be tracked by the gyroscope in the smartphone and is also an interesting feature.

The Swype input technique (method in where the user has the normal keyboard on the touch pad and is drawing a continuous line that starts at the first letter and ends at the final letter of the word) also introduces new features like:

- Double letter: The rule to write a letter two times is to draw a circle on it. That means a circle clockwise or the other way around.
- Digraph: Digraph shows the characteristics during writing two characters. In addition to the time needed between one and the next letter there could be added information about the acceleration (acceleration between two letters) and speed (maximum or average speed between two characters).
- Holding time: The time holding at a position to search for the next letter.
- Straightness: How straight or curvy are the lines between two letters and whether the user is chattering during drawing the line.

While these features are interesting, they can only be used in smartphones that have all the necessary sensors. The False Acceptance Rate and False Rejection Rate of these sensors must also be considered.

Wu et al [35] also studied keystroke and gesture dynamics profiling, but their system uses this type of behavior biometrics in order to continuously authenticate the user, guaranteeing that the person using the smartphone is the rightful user and not an attacker.

The major contribution of their solution consists on not only profiling the behavioral biometrics with keystrokes and gestures, but also acquiring the specific characteristics of one-touch motions during the user's interaction with the smartphone. Wu et al focused on the way a user interacts with the touchscreen, analyzing the specific location touched on the screen, the drift of the finger moved in all directions, the area that is touched and the pressure applied to the screen.

Like in the system proposed by Trojahn and Ortmeier, the smartphone must be equipped with good sensors in order to correctly identify each user. These sensors are expensive, if the smartphone doesn't have them yet.

2.7 User and System Profiling

Techniques of profiling consist on studying, finding and retrieving unique information about the system and the users in order to create a profile that uniquely identifies the user or machine. These techniques could be applied to browsers, smartphones and computers using diverse information.

Ali et al [5] proposed a system to perform user profiling through browser fingerprinting, working on desktop computers and on smartphones. Cookies can't be used to track the users since they may get deleted or the browser may be set to reject them. Most mobile phones don't even support cookies. Fingerprinting may be used to help websites to track users by looking at the characteristics or attributes of a browser like plug-ins, time zone, fonts and many other features, making it possible to track users even if they erase their cookies.

After combining the mentioned attributes in a hash, and creating user profiles, Ali et al made some interesting findings:

- Different screen sizes in smartphones are more unique than one might think. About 20% of the system users were identified by their unique screen resolutions.
- The maker, model version and operating system of the smartphone also created a unique input in the generation of the hash.
- For Android and Windows smartphones the mentioned attributes worked and contributed about 60% in the results. Apple devices were not so easy to penetrate, these devices only showed the IOS version.

These finding are useful to the proposed solution, even though the described attributes alone aren't enough to identify every smartphone, they could be used with more factors like applications installed, IMEI and IMSI number, just to name a few, to create a full profile of a smartphone.

Boda et al [8] also address the subject of user tracking using browser fingerprinting. However, the goal of their study is to identify common attributes of all browsers, achieving that way a cross-browser fingerprint. Boda et al state some browser and system-independent properties:

- Networking information - since HTTP requests are sent via TCP/IP, the server always sees the IP address (and hostname), and the TCP port number. The location of the client can also be inferred from the IP address in most cases.
- Application layer information - the user agent string is a standard HTTP header, and is sent with every request. It contains the type and version of the browser; the name and version of the operating system; the type and version of the layout engine; and the names and versions of certain extensions.
- Information gained by querying the browser - JavaScript programs have access to the list of fonts, plugins (along with their version numbers), screen resolution, and the time zone.

The goal of the system proposed by Boda et al is to create a unique identifier from specific browser data using JavaScript and server-side algorithms. Cross domain tracking is achieved by using only the first two octets of the IP address, which remain constant in many cases even if the IP address of the client changes dynamically. Boda et al explain that using TOR or any other proxy against their system is futile because these techniques modify the IP address only, and therefore the other parameters could still be used to track the user.

Despite that the described system is intended to work in desktops, some ideas can be applied to smartphone profiling and to the proposed solution in particular, namely retrieving the IP from the HTTP request and inferring the user location through the IP address, it is quite expensive to the battery life of a smartphone to constantly monitor user location via GPS.

Regarding profiling in smartphones, more concretely Android, Stöber et al [32] proposed a system capable of performing smartphone fingerprint based on application behavior and the traffic generated by applications.

Stöber et al claim that most of the apps generate traffic, which is not only initiated by the user, but also from the apps itself to maintain the most current state, receive updates, or synchronize cloud services, estimating that only 30% of the overall smartphone traffic can be attributed to user interactions. This way many of such background activities result in characteristic traffic patterns, especially in the time domain and in the volume of transmitted data. In addition, the generated traffic highly depends on the multitude of installed applications and their personal configuration. A passive eavesdropper adversary in this scenario might be able to capture encrypted wireless 3G/UMTS data from a victim's smartphone and use that traffic to extract smartphone fingerprints. The fingerprints would then allow him to identify the device afterwards and make a point of whether the victim's smartphone is present within a certain UMTS radio cell.

The system proposed by Stöber et al uses data mining algorithms and the burst of information sent by applications to identify one user. These algorithms allow the system to know each application generated a specific part of the traffic, identifying that way the application and the user behavior within the application.

The ideas introduced by Stöber et al are interesting and can be used to detect the applications running on the smartphone and some actions performed by the user in these applications, creating that way a behavioral profile of the user.

Dai et al [15] proposed a system to automatically fingerprint Android applications constructing a user profile with that information. The main idea behind this fingerprinting consists on the fact that applications could be identified by looking at the 3rd party servers they use, or looking at User-Agent or Host field since some applications have their name in those fields (Apple applications are enforced to have their name in the User-Agent field).

To identify applications, Dai et al developed the concept of a network profile for an Android application that uses unique characteristics of the network behavior of the application (which these authors refer as network fingerprints) to identify the application.

The network fingerprint of an Android application consists of two components: hosts that the application connects to and a state machine representing the patterns over the HTTP header of the requests made by the application to those servers. Applications are then identified by the Host that they try to contact in the HTTP connection and by looking at some keywords in the manifest file of the .apk of the applications.

Dai et al claim that advertisement providers also help on identifying the applications. Most of the free applications in Android Market contact advertisement providers to generate revenue for the developers. Many of these providers use unique identifiers for each application in the flows to identify the applications that caused the traffic to be generated. Dai et al found that almost half of the tested applications had information about the advertisement libraries in the manifest file.

Dai et al also note that applications can also be identified by looking at their query-key that is used to connect to third-party providers. This happens since some of these providers such as platform providers (e.g. Facebook) and advertisement providers (e.g. Admob) typically publish guidelines for developers which mention that the query-key should identify the app.

Even though the installed applications will be identified in the proposed solution using the Android API, the ideas described by Dai et al consist on an alternative way of identifying applications. This alternative way of identifying applications could be used alongside the proposed solution in order to better identify the applications installed in the smartphone.

Conti et al [12] proposed a system that aims to identify users based on their actions on Android applications via analyzing the encrypted traffic generated by these applications.

Conti et al point out that several attacks may violate the privacy of communications, even when the adversary has no actual control of the phone. If the network traffic is not encrypted, the task of an eavesdropper is simple: he can analyze the payload reading the content of each packet. However, many mobile apps use the Secure Sockets Layer (SSL) and its successor Transport Layer Security (TLS) as a building block for encrypted communications. While SSL and TLS protect the content of a packet, they do not prevent the detection of network packet patterns that may reveal some sensitive information about the user behavior.

Conti et al captured all the traffic generated by Facebook, Gmail and Twitter Android applications. Using machine learning algorithms, they found themselves able to identify some user's actions in those applications, namely detect whenever a user posted status on his or other user's wall, open a profile or a message. In Gmail, they were able to identify whether a user was sending email, tapping the reply button, opened the chat or sent an email reply. In Twitter, they were able to identify whether a user was tweeting or sending messages to someone.

The most interesting finding of this study was that even though some applications encrypt their traffic whenever a user is logging in, all the remaining application traffic is not encrypted at all. This study also consists on a way of identifying user actions within certain applications, and identifying those actions could be interesting to create a user behavioral profile.

Wei et al [34] performed a completely different, yet interesting work. The goal of their system was to statically analyze Android applications in all the application layers, in order to find information.

The most relevant application layers described by Wei et al are:

- Static Layer - app's functionality and permissions. The APK file is analyzed on two dimensions to identify app functionality and usage of device resources: the permissions that the app asks for and the app bytecode which is parsed to identify intents (indirect resource access via deputy apps).

- User Layer - input events that result from user interaction, in particular, the touches generated when the user touches the screen. Touch events include pressing the app buttons of the apps and swipes. The intensity of events and the ratio between swipes and presses are powerful metrics for GUI behavioral fingerprinting.

These layers are interesting and relevant to the proposed solution because they consist on a way of analyzing the installed applications and some user actions within those applications, namely the pressing of some buttons that are part of the application.

2.8 Android Privilege Escalation

The Android Operating System is based on Linux, having similar user types with different privileges. Each Android application has its own space inside the file system to write and read information and this space is protected from other applications, which means that an application can only read and write information in his own space and can't perform those activities in the space reserved for another application. However, an application with root capabilities can read and write everywhere in the file system, controlling the file system as it likes [36].

While some users root their own device in order to make themselves able to use applications that require root privileges or to control parts of the smartphone that otherwise would not be controlled, a rooted device is less secure since it may be targeted by malware who become able of doing anything it wants. Privilege Escalation attacks consist on applications (typically malware) trying to gain root access to the smartphone, making themselves able to read information it shouldn't be able to read, or writing in spaces it shouldn't be writing.

Heuser et al [23] presented DroidAuditor, a forensic tool to analyze application layer privilege escalation attacks using interactive behavior graphics. The tool created by Heuser et al is capable of detecting malware corresponding to the most common types of application layer privilege attacks: Confused

Deputy Attacks and Collusion Attacks.

Confused Deputy Attacks are attacks where an attacker abuses vulnerable software components via Inter-Process Communication (IPC) to perform sensitive operations. Collusion Attacks are attacks where multiple applications operate together to share their permissions towards a common goal.

2.9 Attacker Models

This section contains several attacks and models of attackers present in the literature relevant to the solution proposed in this work, comprising several attacks against Android applications and its operating system, attacks relevant to the communication between an Android application and a server and attacks against the Android Keystore (which saves the generated keys in the TEE whenever it is possible). These attacker models will be used to define the attacker models used to evaluate the security of the solution proposed in this work.

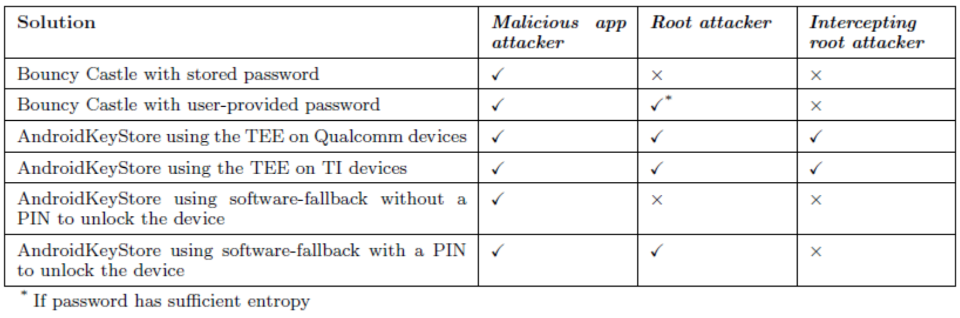
Cooijmans et al [13] developed a comparison study between the Android Keystore implementation for Android Application Programming Interface (API) Level 18 and a third-party cryptography library for Java called Bouncy Castle. Three attacker models were defined, ordered by increasing order of power:

- Malicious app attacker - attacker that tries to attack the Keystore using an application installed on the device. This application is assumed to be installed from Google Play Store and has all the permissions allowed by the store.
- Root attacker - attacker that has root credentials and is able to run apps under root permissions and inspect the file system.
- Intercepting root attacker - attacker with the same abilities as the Root attacker but also capable of capturing user-input in real time.

Cooijmans et al then analyze 6 different solutions of secure key storages (Bouncy Castle with stored password, Bouncy Castle with user-provided password, AndroidKeyStore using the TEE on Qualcomm devices, AndroidKeyStore using the TEE on Texas Instruments devices, AndroidKeyStore using software-fallback without a PIN to unlock the device and AndroidKeyStore using software-fallback with a PIN to unlock the device) in respect to three security requirements:

- Device-binding - the key can only be used on a certain device.
- App-binding - the key can only be used by a certain application on a certain device.
- User-consent required - the key can only be used when the user gives explicit consent to use it.

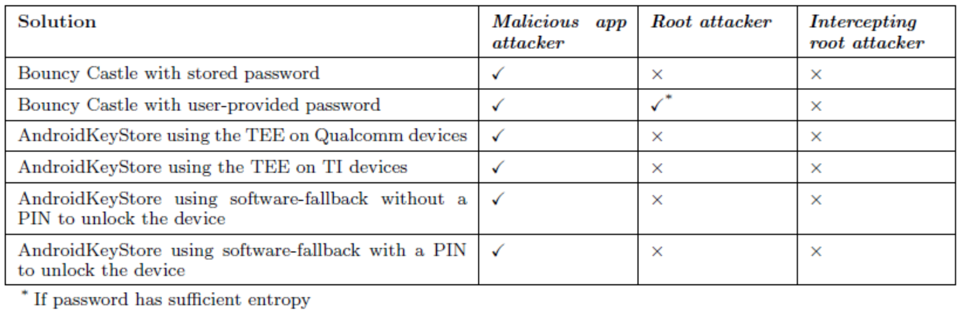
Cooijmans et al results are described in the Figures 2.2, 2.3 and 2.4. The malicious app attacker is unable to break any security guarantee, no matter the solution used to securely store the keys. The major security flaws consist on attackers with root capabilities being able to read and write everywhere in the file system, making themselves able to copy the Keystore file and the file that stores the password used when storing the Keystore file to another place, being able to reconstruct the same Keystore in a different location.



Solution	*Malicious app attacker*	*Root attacker*	*Intercepting root attacker*
Bouncy Castle with stored password	✓	×	×
Bouncy Castle with user-provided password	✓	✓*	×
AndroidKeyStore using the TEE on Qualcomm devices	✓	✓	✓
AndroidKeyStore using the TEE on TI devices	✓	✓	✓
AndroidKeyStore using software-fallback without a PIN to unlock the device	✓	×	×
AndroidKeyStore using software-fallback with a PIN to unlock the device	✓	✓	×

* If password has sufficient entropy

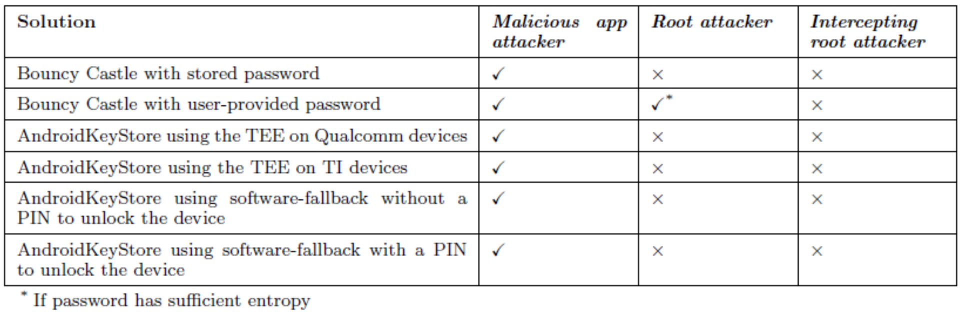
Figure 2.2: Cooijmans et al Device-binding results [13]



Solution	*Malicious app attacker*	*Root attacker*	*Intercepting root attacker*
Bouncy Castle with stored password	✓	×	×
Bouncy Castle with user-provided password	✓	✓*	×
AndroidKeyStore using the TEE on Qualcomm devices	✓	×	×
AndroidKeyStore using the TEE on TI devices	✓	×	×
AndroidKeyStore using software-fallback without a PIN to unlock the device	✓	×	×
AndroidKeyStore using software-fallback with a PIN to unlock the device	✓	×	×

* If password has sufficient entropy

Figure 2.3: Cooijmans et al App-binding results [13]



Solution	*Malicious app attacker*	*Root attacker*	*Intercepting root attacker*
Bouncy Castle with stored password	✓	×	×
Bouncy Castle with user-provided password	✓	✓*	×
AndroidKeyStore using the TEE on Qualcomm devices	✓	×	×
AndroidKeyStore using the TEE on TI devices	✓	×	×
AndroidKeyStore using software-fallback without a PIN to unlock the device	✓	×	×
AndroidKeyStore using software-fallback with a PIN to unlock the device	✓	×	×

* If password has sufficient entropy

Figure 2.4: Cooijmans et al User-consent results [13]

While these results are important and the attacker models contained are still relevant nowadays it is important to note that Android developers are putting an effort to fix security flaws with each version and some of the described problems may no longer exist in the most recent versions of Android. Android Keystore was totally renewed with the release of Android 6, having its functionality increased and working in a more secure manner.

R. Loftus and M. Bauman [25] studied File Disk Encryption in Android 7, analyzing if known attacks to File Disk Encryption in previous versions of Android were still possible in the version 7 or not. Particularly, Loftus and Bauman analyzed whether Brute Force Attacks (either online and offline) against Android authentication were possible and if Cold Boot Attacks, Evil Maid Attacks and Fingerprint Bypass Attacks are possible.

Cold Boot Attacks are attacks where an attacker freezes the device RAM in order to attempt recovering information that was still in memory, this attack is very effective if the keys are loaded in memory and the attacker is able to recover them.

Evil Maid Attacks consist on an attacker installing a keylogger in an unencrypted part of the device, if the device is left unattended. When the rightful user inserts the key that decrypts the disk, the keylogger catches the key and sends it to the attacker. Android is vulnerable to Evil Maid Attacks since the Binder can be used as a keylogger. The Binder is the method which deals with IPC on Android, being also used in communications between activities in the same application. If an attacker is able to intercept the communication between the defined input method and the Binder, he will be capable of recovering the inputted key.

Fingerprint Bypass Attacks are also a threat since Android started supporting fingerprint authentication from the version 6.0 onwards. An enrolled fingerprint is stored in the device's TEE, which assigns a random number to the fingerprint excluding 0, since 0 is used for non-recognized fingerprints. That way, when a user attempts to authenticate using his fingerprint, the Android system sends the freshly acquired fingerprint to the TEE which checks if the received fingerprint matches any stored one. If it matches, TEE returns its identifier, and in case it doesn't match it returns 0. Android will then reject any fingerprint that returns 0 but accept every other. An attacker with root capabilities may change the response value from 0 to a random value, making that way the system accepts an invalid fingerprint.

R. Loftus and M. Bauman note that Brute Force Attacks are not possible in Android 7, since the vulnerabilities that allowed offline attacks were fixed and online attacks are mitigated by timeouts imposed in the system when several pins attempts are invalid, it is noted that an online Brute Force attack against a 4-digit pin would take 27 years to be completed. Cold Boot Attacks are also not possible in Android 7, since Google enforced that the keys that encrypt the parts of the disc should be securely stored in the TEE. Due to this fact there aren't cryptographic keys written in memory. Evil Maid Attacks through the Binder are still possible and Fingerprint Bypass are still possible if the attacker has root capabilities.

Chapter 3

Solution Overview

Contents

3.1 Background

3.1.1 Android Operating System

Android [1] is an open-source mobile operating system developed by Google, and based on Linux. The architecture of the system, which can be seen in Figure 3.1, consists of:

- Linux Kernel - Android is based on a modification of the Linux Kernel, which allows functionalities like threading, low-level memory management and some key security features.
- Hardware Abstraction Layer - set of multiple library modules that connects the device hardware with the higher-level API.
- Android Runtime - created in Android version 5.0, allows each application to run in its own process. Android Runtime is designed to run multiple virtual machines on low-memory devices by executing optimized bytecode files, called the DEX files.
- Native C/C++ Libraries - many core Android system components and services, such as Android Runtime and Hardware Abstraction Layer, are built from native code that require native libraries written in C and C++.
- Java API Framework - APIs written in Java that give the developer the needed building blocks to create applications.
- System Apps - Android comes with a set of default core applications installed, like email, SMS messaging, calendars, Internet browsing, contacts, to name a few. These applications can be replaced by the user. Developers can also extend these applications to create their own applications.

3.1.2 Android Keystore

The Android Keystore [3] system was introduced in Android 4.3 (API level 18) and allows users to store cryptographic keys in a secure container, making it difficult to extract the keys from the device. The keys stored in the Keystore can be used for cryptographic operations with the key material remaining non-exportable. The Keystore system also offers the feature of restricting how keys can be used. For example, the Keystore may be defined to require user authentication to use the keys or restrict which keys can be used in certain cryptographic modes.

Android Keystore system prevents extraction of the key material from application processes and from the Android device and by preventing applications of using keys that are outside of the applications' processes.

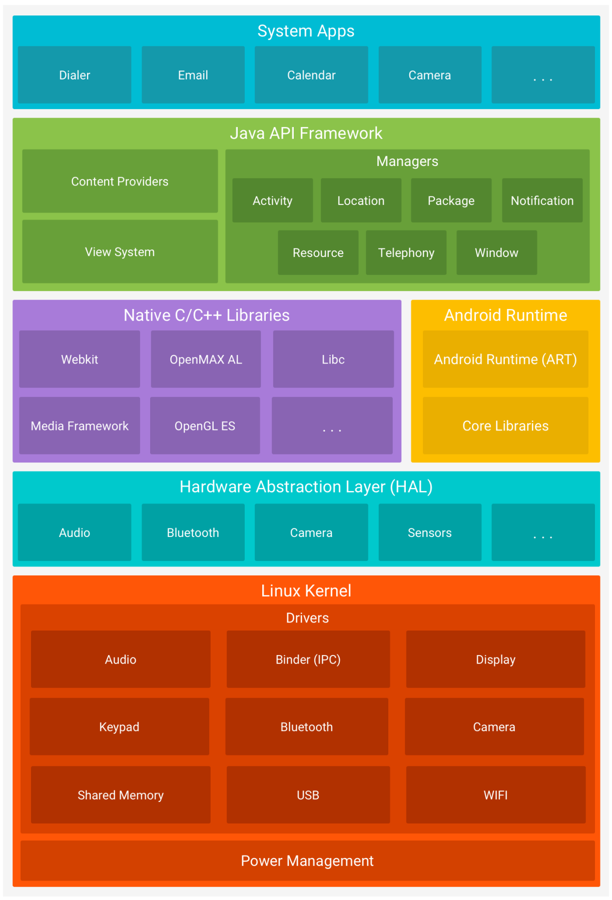


Figure 3.1: Android Software Stack [1]

Regarding extraction prevention, the Keystore system provides a system process that deals with all cryptographic operations, in this way protecting all the key material (like plaintext, ciphertext, messages to be signed or verified, among others) from being extracted by an attacker. This is achieved by storing the keys in secure hardware (like ARM TrustZone TEE) whenever possible. When this feature is enabled, key material is never exposed outside of the secure hardware.

If the device's secure hardware supports a particular combination of key algorithms, the keys stored in the Keystore may never be extracted from the device, even if the Android OS is compromised or an attacker can read the device's internal storage. It is important to note that in these cases, the attacker may be able to use the stored keys.

Android Keystore system mitigates unauthorized use of stored keys by letting applications specify authorized uses of their keys. These authorizations are enforced whenever the key is used. Once these keys are generated or imported, its authorizations may not be changed.

3.1.3 Trusted Execution Environment

The TEE is an isolated secure processing environment capable of processing information and storing information, assuring the integrity of the processing. The TEE makes it possible to partition applications in a way that sensitive operations and data never leave the TEE, making the applications more secure. The TEE also has the potential of replacing hardware tokens such as the tokens capable of generating an OTP, improving the usability and reducing the costs associated with these services [19].

ARM TrustZone [2] is a TEE implementation done by ARM which is aimed at system-on-a-chip architectures used in mobile devices. The core concept of the TrustZone is the separation between the normal world and the secure world, as seen in Figure 3.2 [26].

These separated worlds have different privileges and independent memory addresses. Code running in the normal world can only access the normal world address space while code running in the secure world can access both worlds address space [30].

Since the processor only executes a security mode at a time, there must be a mechanism that allows the processor to switch worlds. This is done using a special instruction called the Secure Monitor Call. When this instruction is executed the hardware switches into the secure monitor, performing a secure context switch into the secure world and enabling shared data from being copied across worlds [30].

Android Keystore uses the ARM TrustZone to securely store keys and perform operations with the stored keys. Since memory addresses space is separated between worlds, all the operations performed in the secure world cannot be accessed in the normal world (where the applications execute).

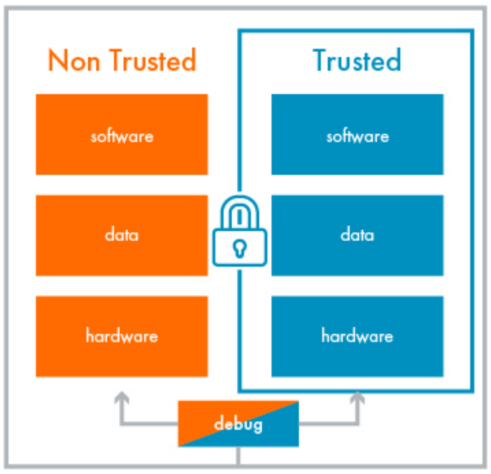


Figure 3.2: ARM TrustZone Two Worlds Paradigm [2]

3.1.4 Cryptographic Primitives and Security Protocols

3.1.4.A HMAC

A Message Authentication Code is a piece of information used to guarantee that the message has not been changed and that the message came from the rightful sender. This way, a Media Access Control (MAC) protects the data integrity and the authenticity of the corresponding message.

To achieve these objectives, a secret key is shared between the two parties that are communicating and then used to verify whether the message was changed or not. Since the key is only shared between the communicating parties, the receiver has the guarantee that the message was sent by the expected sender. This guarantee only exists if the shared key can be successfully used to verify the message.

An HMAC is a type of Message Authentication Code that involves a cryptographic hash function and a secret cryptographic key. Similarly to any MAC, an HMAC is used to simultaneously verify both data integrity and message authenticity. An HMAC is as secure as the underlying hash function, the size and security of the secret key and the size of the corresponding hash output.

An HMAC does not encrypt the message but instead the message must be sent alongside with the HMAC hash output. In order to verify the message, the receiver must calculate the HMAC of the message himself and check whether its calculated hash matches the received hash.

3.1.4.B HTTPS

HTTPS is a secure communications protocol that consists on communication over Hypertext Transfer Protocol (HTTP) within a connection encrypted by Transport Layer Security (TLS). HTTPS guarantees the privacy and integrity of the exchanged data, as well as authentication of the communicating parties.

HTTPS encrypts the communications between a client and server, protecting it against eavesdropping, message tampering or forged messages. This ensures that:

- No one but the correct communicating parties will be able to read the content of the exchanged messages.
- No one will be able to change the contents of the messages without being detected.
- No one will be able to introduce forged messages without being detected.

3.2 Problem

Smartphones are widely used nowadays because: 1) they do everything a regular phone does (like phone calls, SMS, etc.) and 2) allow that the user install and use a wide range of applications like email, games, social networks, banking applications, among others.

While the major part of applications contains meaningless or no information about the user, the information contained in key applications like email or banking applications is personal and valuable to the user. Considering the high value of the information contained in those applications and since textual passwords have been proven insecure, the authentication in these applications usually requires two authentication factors or alternative types of passwords as an attempt to make them safer.

Some of the problems related to textual passwords consist on the following: short passwords are easy to guess and crack. Long passwords solve this problem but they are hard to remember. Textual passwords also may contain patterns that could be exploited with some attacks.

An alternative to textual passwords is graphical passwords that use the fact that images are easier to remember than text. These passwords are resistant to the majority of attacks performed on textual passwords. However, other types of attacks could be performed, like Shoulder Surfing, where an attacker can see the images chosen by the user and is able to reproduce them later.

Another possible solution is to use biometrics like fingerprint, voice and facial recognition as well as behavior biometrics. However, some of these techniques require specific and sometimes expensive hardware components to work making it a problem if the phone does not have this type of hardware installed.

Some solutions use location as a second authentication factor. These systems aim to authenticate the user if the distance between the user and a certain location is short enough. The major drawback of these systems is that the user may only authenticate himself if he is in the range of a specific location, or, in other solutions, if he carries an object.

The general solution adopted by systems like Gmail to overcome the problem of a login using only a textual password is to generate a One-Time Password and send it to the user. Then, the user will have to input the received password.

All these additional authentication factors require the user to introduce information or perform actions he would not perform otherwise, introducing an undesired overhead especially considering mobile applications, where the user does not want to be introducing lots of text or clicking on lots of buttons.

3.3 System Architecture

The objective of the proposed solution consists on creating a unique user profile based on unique features of Android smartphones and features regarding user behavior like installed applications, memorized networks or memorized accounts. The generated user profile will be used as a second authentication factor in an authentication system. This way, the proposed system does not require the user to perform additional tasks in order to prove his identity since the profile will provide by itself everything the system needs to authenticate the user.

The unique user profiles will be stored in a server that is also responsible for verifying the similarity between two profiles, deciding whether a received profile should be accepted or rejected. The matching algorithm will be explained in Section 3.3.1. If a profile is rejected the third-party's fallback mechanism is activated. These situations need to be analyzed with caution because they correspond to users logging with different smartphones or users that changed a huge part of their smartphone information. This way, this system is an improvement to the state of the art authentication processes since this system requires fewer user actions in the majority of the cases and behaves exactly like the state of the art systems in the worst cases, introducing no overhead.

The architecture of the system described by the proposed solution is the following:

- Profiling Server - the server that contains all the information regarding the created user profiles. This server is responsible for receiving the profiles, storing them and verifying whether the received profiles are similar enough to the stored ones.
- Third-Party Server - third party server that contains all the information and functionalities provided by the third-party. In this work, considering the example of a banking third-party application and since banking functionalities are out of the scope of this work, this server consists only of a simple authentication server that stores the username and password of each user of the banking application. This server is also responsible for defining the fallback mechanism used when the received profiles are not accepted.
- Information Collector Application - the application that runs on the phone and is responsible for collecting the information, creating the profile and communicating with both servers.

There are some possible alternative architectures that does not require the usage of a Profiling Server, making the Information Collector Application communicate directly with the Third-Party Server. Some examples include an architecture that stores and validates the profiles on the smartphone, and an architecture who stores and validates the profiles on the Third-Party Server.

However, these types of architectures have some serious security issues. Storing and validating the profiles on the smartphone is insecure, since malware with root capabilities would become able to bypass the profile verification, allowing the attacker to log in the system.

Storing the profile information and profile validation on the Third-Party Server is also problematic, since the same smartphone profile would be replicated in every used service, and once one of these services would become compromised by an attacker, the attacker would be able to use the stolen profiles to authenticate in every service associated with the profile. Another security issue would be privacy issues, since every service provider would know the profile of a user.

Considering all these problems, it becomes easy to understand that the Profiling Server is a key component in the solution's architecture and cannot be removed. It is very important that the profile storage and validation is done in an external and secure party such as the Profiling Server, helping both the client and the third-party server to trust each other. Even though the Profiling Server knows every user profile, privacy issues are as relevant in this system as in the state of the art authentication systems, since Google and Apple, for example, store even more information about their users and this fact does not prevent people of using smartphones with Android or iOS operating system.

When implementing the system, an SMS code verification was defined as the fallback mechanism of the third-party part of the application and for the profiling server part of the application. This mechanism was chosen due to being the most commonly used mechanism by banking applications nowadays. This facilitates the comparison between the proposed solution and the existing solutions, as well as it helps the users to understand the system since the fallback mechanism is already known to them.

The Information Collector Application collects all the required information (described in Section 3.3.1) using the Android API, calculating for each retrieved attribute the respective hash. The Information Collector Application then calculates the profile HMAC using a symmetric AES key stored in the device's TEE.

The system has three main phases: the Registration phase, the Bootstrap phase and the Login phase. The Registration and Bootstrap phases occur only once and have the common objective of registering the user in both the Third-Party Server and in the Profiling server, initializing all the required information. The Login phase is the phase that occurs every time and has the objective of verifying if the information sent by the Information Collector Application is stated as valid by the Profiling Server, confirming this way that the login is performed by the rightful user.

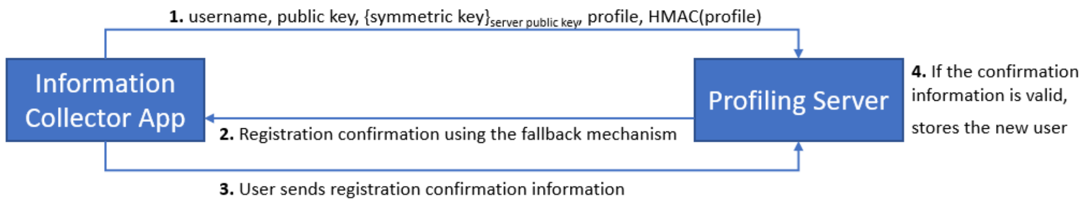
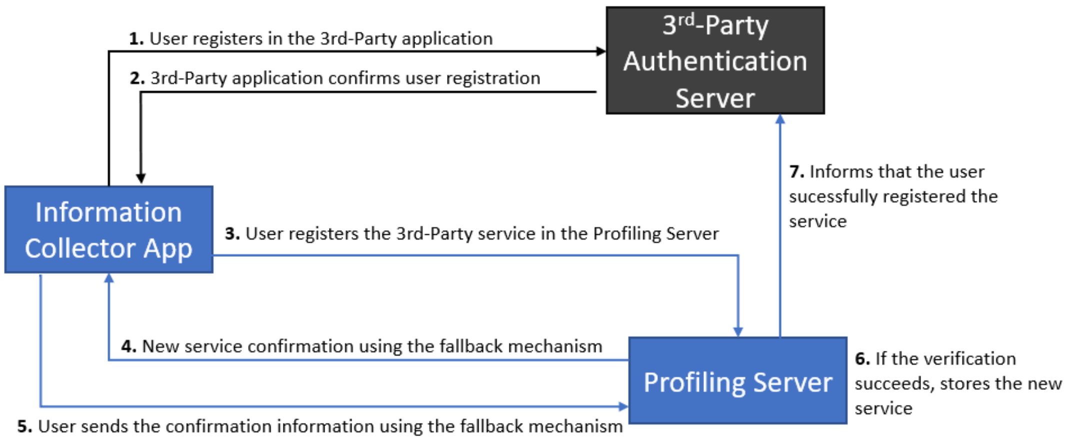

1. username, public key, {symmetric key}server public key, profile, HMAC(profile)
Information Collector App
Profiling Server
2. Registration confirmation using the fallback mechanism
3. User sends registration confirmation information
4. If the confirmation information is valid, stores the new user

Figure 3.3: Registration phase of the system

Whenever a user joins the system, the Registration and the Bootstrap phases occur. The Registration phase can be seen in detail in Figure 3.3. The Registration phase consists on the user registering himself in the Profiling Server.

The Information Collector Application starts by generating an asymmetric RSA key pair and a symmetric AES key to be used to create the profile. Then, the Information Collector Application sends to the Profiling Server the username, the public key of the user, the symmetric key used to create the profile encrypted with the server public key, the freshly acquired profile and its calculated HMAC.

The server sends an SMS code (since it is the defined fallback mechanism) to the smartphone to make sure that the rightful user is registering in the system. If the user replies with the correct code, the server will then store all the received information, registering the user.

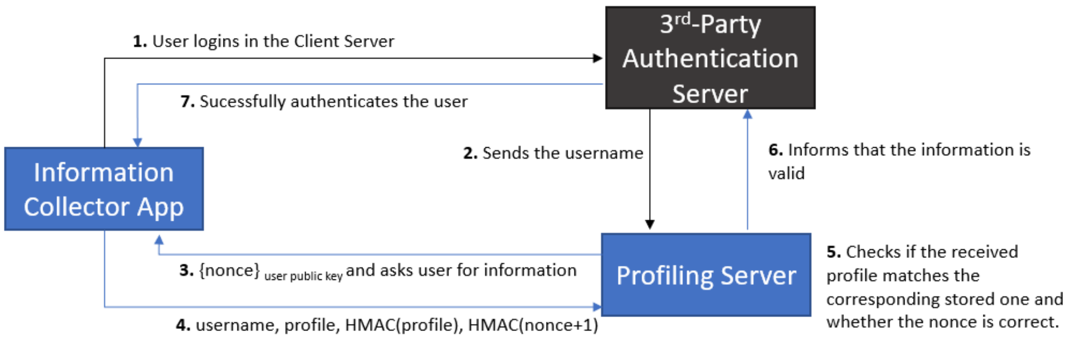


3rd-Party Authentication Server
1. User registers in the 3rd-Party application
2. 3rd-Party application confirms user registration
Information Collector App
3. User registers the 3rd-Party service in the Profiling Server
7. Informs that the user sucessfully registered the service
4. New service confirmation using the fallback mechanism
Profiling Server
6. If the verification succeeds, stores the new service
5. User sends the confirmation information using the fallback mechanism

Figure 3.4: Bootstrap phase of the system

The Bootstrap phase can be seen in detail in Figure 3.4. This phase occurs when the user joins a new service, provided by the Third-Party. In this case registration is done online for simplicity, but in real cases that phase could be replaced by the user going to the Third-Party to ask for registration and receiving later a letter from the Third-Party with his username and password to be used within the application. The user starts by registering himself in the Third-Party Server, receiving a confirmation of his registration. Then the user notifies the Profiling Server that he is now using the stored profile in the new service.

The Profiling server will then send an SMS code (since it is the defined fallback mechanism) to the smartphone to confirm the registration of the new service. If the user replies with the correct code, the server will then store the new service and notify the Third-Party Authentication Server that the user has successfully registered the service.



3rd-Party Authentication Server
1. User logins in the Client Server
7. Sucessfully authenticates the user
Information Collector App
2. Sends the username
6. Informs that the information is valid
3. {nonce} user public key and asks user for information
Profiling Server
5. Checks if the received profile matches the corresponding stored one and whether the nonce is correct.
4. username, profile, HMAC(profile), HMAC(nonce+1)

Figure 3.5: Login phase of the system

The Login phase can be seen in detail in Figure 3.5. The Information Collector Application starts to communicate with the Third-Party Authentication Server, sending the user credentials. If the credentials have a match in the storage of the Third-Party Authentication Server, the Third-Party will communicate the username to the Profiling Server.

The Profiling Server will send to the user the nonce that should be used to ensure the freshness of the login. The user will receive a push notification asking whether he wants to collect the information and login or whether the login request was just a mistake or an attacker trying to impersonate the user. If the user validates the login request, the Information Collector Application will acquire the most recent profile by reading all the necessary information from the smartphone and sends it to the Profiling Server alongside the username, the profile HMAC and the HMAC of a known variation of the most recent received nonce.

The Profiling server will check if the username is valid and if the received profile matches the calculated HMAC. The Profiling Server will also verify if the received profile is similar enough to the stored one to be accepted and the server will also verify if the received nonce is the most recent one or if the message was a replay of a previous one or even a forged one. If all these parameters check, the server has now authenticated the user. The Profiling Server notifies the Third-Party Authentication Server that the user is valid. The Third-Party Authentication Server will then successfully authenticate the user, allowing the user to use the Third-Party application.

All the communications are done using HTTPS to ensure confidentiality and integrity of the messages. Each message is also sent along with its digital signature to ensure mutual authentication and non-repudiation. This way, it is easy to detect whether the messages come from the expected sender or an attacker attempted to impersonate the sender and whether the message's integrity is ensured or an attacker attempted to change the message.

3.3.1 Profile Verification

In this work, a profile is a set of the hashes of attributes (described in Table 3.1), where each attribute corresponds to a different feature of the smartphone. Since Android is a customizable Operating System and certain information may change over time, three types of attributes were identified with, with each type requiring a different type of verification. The three types of attributes are the following:

- Static Attributes - information that mostly concerns the smartphone itself and therefore cannot be changed.
- Dynamic Attributes - information that concerns the Operating System (OS) of the smartphone and the Software Development Kit (SDK) version or the Subscriber Identity Module (SIM) card of the user. While some of this information is very unlikely to change over time, change is still a possibility.
- Dynamic List Attributes - attributes that do not consist of a single instance of information but relate to a set of different values. An example is the attribute *Installed Applications* that consists on a set

Name	Type	Description
IMEI	Static	Unique global identifier of the smartphone
MAC Address	Static	Smartphone's network interface unique identifier
Screen Resolution	Static	The screen size of the smartphone
Software Version	Dynamic	Software version number of the device
SIM Operator	Dynamic	The identifier of the SIM Operator
SIM Operator Name	Dynamic	The name of the SIM Operator
SIM Country ISO	Dynamic	The ISO country code for the SIM's provider country code
SIM Serial Number	Dynamic	The serial number of the SIM card
IMSI Number	Dynamic	The unique subscriber identifier of the SIM card
Location	Dynamic	The location of the user calculated using the IP address
OS Version	Dynamic	The version of the Operating System in use
SDK Version	Dynamic	The Android SDK version of the Operating System in use
Device Name	Dynamic	The name of the device
Keyboard Language	Dynamic	The language of the keyboard
Memorized Networks	List	The SSID of the networks memorized by the smartphone
Google Accounts	List	The Google accounts memorized by the smartphone
Memorized Accounts	List	The other accounts memorized by the smartphone
Input Methods	List	The list of keyboards installed in the smartphone
Installed Applications	List	The list of applications installed in the smartphone

Table 3.1: Retrieved attributes and its type

of package names of each installed application in the smartphone.

The Location attribute does not intended to contain the precise location of the user, which would have the same disadvantages has the Two-Factor Authentication schemes that use the location of the user to authenticate the user in the system. Instead, this attribute retrieves the location of the user with a big granularity (country-wise). This attribute is only retrieved to increase the complexity of generating a false profile, since a remote attacker that spreads malware through the globe would only be able to effectively attack users in his own country, otherwise this attribute will mismatch.

Due to privacy questions as well as facilitating the profile verification, each attribute consists of the hash of the real value of the attribute. Since lists are exceptional cases, each list attribute consists of a list of hashes where each hash is calculated using the real value of each different element of the real list. This way, only the smartphone of the user deals with the real information, making it impossible to know the real values of each attribute.

To improve the security of the system, an HMAC of the whole profile is calculated and send alongside the profile, preventing an attacker from generating a similar profile or brute forcing the hashes to know the real information. The symmetric key used to generate the calculated HMACs is securely stored in Android's Keystore, which stores the keys in the smartphone's TEE, whenever the smartphone is capable of storing information in the TEE.

Regarding profile verification, the main idea is quite simple: static attributes cannot change or else we are in the presence of a different device than the one used to register the user. The system should accept changes in dynamic attributes and store the most updated information whenever a profile is accepted. However, given how unlikely it is to change the major part of the dynamic attributes at the same time, when a big part of the dynamic attributes are changed it is more likely that we are in the presence of a different device than in the presence of the registered device. This way, it is necessary to define a certain threshold to determine whether the number of changed attributes could be accepted or not.

List attributes are once again a special case. It is very frequent that a user changes only a small part of the list instead of the full list, and a verification that consists only of checking whether two lists are exactly the same does not consider those cases, therefore being required thresholds that will help to determine whether two lists are similar enough for the profile to be accepted or not. The only thing considered regarding changes in dynamic attributes is the number of changed attributes. Lists verification requires a more accurate approach in order to check whether the major part of a list remains unchanged and whether the number of changes is not significant. This is based on the idea that a user is more likely to add new information than just replace the existing one. For example, a user is more likely to uninstall a couple of applications and install a larger number of applications from one login to another than uninstalling half of his applications and replacing it by an equal number of applications.

With all these ideas in mind, the verification algorithm is the following:

- If two profiles are exactly equal verification succeeds.
- If static attributes are changed the verification fails.
- The threshold for dynamic attributes is defined as 75%. This means that considering the 12 dynamic attributes and the 4 lists the server will only accept up to 4 changed attributes. Is important to note that whenever the SIM card is changed or the Operating System is updated the verification will most likely fail which is an acceptable outcome due to the nature of the changes while an everyday use of the phone that includes new networks, applications and accounts will not be rejected unless those lists are heavily changed.
- If the verification has not failed in some other points, lists will be verified against thresholds to determine whether the verification could be accepted. If a list fails this verification, the whole verification of the profile fails, which is a reasonable situation since it is quite abnormal to see a user completely change his installed applications, memorized networks or memorized accounts. List verification is done considering the common elements and the new elements. The percentage of common elements must be above the threshold and the percentage of new elements must be below 1 - threshold. The threshold for Installed Applications, Input Methods and Memorized Networks is 75% and the threshold for Google Accounts and Memorized Accounts is 66%.

The threshold for the lists was defined after a preliminary test phase with around 16 users that is described in detail in the Evaluation Chapter. In this test phase, the median number of each list was

Attribute	Median number (per user)
Google Accounts	1
Input Methods	2.42
Installed Applications	53
Memorized Accounts	4.5
Memorized Networks	27.26

Table 3.2: Mean number for each attribute after First Test Phase

calculated, the results can be seen in Table 3.2.

Given these numbers, considering an example of a user with 50 installed applications, 25 memorized networks and 5 memorized accounts (Google Accounts and Input Methods are discarded from this example since a change in them will make the verification fail, which is acceptable given how unlikely these attributes are changed), the same user must have at the next login 37 installed applications out of a potential 62 installed applications, 19 networks out of a potential 31 networks and 3 accounts out of a potential 6 accounts.

Finally, whenever a profile has suffered some changes but is still accepted by the verification the server will update the stored information replacing the old attributes by the changed ones. This way, the stored profile always corresponds to the information sent in the most recent successful login, making sure that a changed value is only processed once and that whenever the changes are not significant enough the system will be able to authenticate the user by itself, not requiring the user to perform further actions, accomplishing that way the objectives of this work.

Note that even though an attacker may try to slowly change the attributes that compose the profile, whenever the rightful user logs in his information will always prevail, which means that attributes changed by an attacker would change to the real value every time the rightful user logs in.

3.4 Implementation of the Solution

The two servers described in the System Architecture section were implemented as two HTTPS Java servers that use MySQL databases to store all the relevant information in a persistent way. The decision to implement the servers in Java was done based on both servers being relatively simple, based on how easy it is to implement and test an HTTPS server in Java 8 and due to the fact that Android runs in Java and some libraries could be used either in the application side and in the server side.

The certificate of the Profiling Server is hardcoded in the Information Collector Application, making sure this way that the application always comes with the right certificate and there are not possible Man-In-The-Middle attacks that change the certificate that would be sent by the server.

Unit tests were designed using JUnit. The objective of these tests was to test the storage and the server's connection to the storage and test whether the server handlers would give the proper responses to the requests made by the Android application. Server logs were created using the Java Logging package. These logs contain information about all the messages received by the server, about the corresponding server reply and also information about Profile Verification, namely the attributes that changed.

The Information Collector application was created using Android 4.4 as the minimum version of the Operating System but an Android 6 application was also done with few adaptations from the original work. The reasoning behind choosing Android 4.4 as the minimum version is that a huge percentage of Android users still use Android 4.4, Android 5 or Android 5.1. If the minimum version targeted Android 6 this huge percentage of users would be unable to use the application, which would be unrealistic for an application that is trying to simulate a banking application. The application uses Google Volley to communicate with the server, using JSONRequests to send the information.

The adaptations done in Android 6 consisted in explicitly requiring the user consent to use some strong permissions like READ_PHONE_STATE and GET_ACCOUNTS and also storing the AES key in the Keystore. The Keystore present in Android 4.4 does not allow symmetric key storage, so the application uses a bit of a workaround that consists on storing the symmetric AES key encrypted by the private RSA key in a file, since asymmetric keys are stored in the Keystore in all versions of the Keystore. Android 6 also introduced a change that requires applications to ask for user consent in order to obtain certain strong permissions like the ones noted above instead of just having permissions declared in the Manifest. Whenever the user registers for the first time he is requested to give consent for the application to use both permissions. Once the consent is given the application will be able to work without problems all the time.

Regarding the profile collection, the application requires the following permissions declared in the Manifest:

- CHANGE_WIFI_STATE - permission required to programmatically turn on and off the WiFi, explained below why this is necessary.
- INTERNET - permission required to contact the HTTPS servers and external APIs.
- ACCESS_WIFI_STATE - permission required to retrieve the list of memorized networks.
- ACCESS_NETWORK_STATE - permission required to retrieve the IP and MAC addresses.
- GET_ACCOUNTS - permission required to retrieve the list of memorized accounts.
- AUTHENTICATE_ACCOUNTS - permission required to retrieve the list of memorized accounts.
- READ_PHONE_STATE - permission required to retrieve information related to the smartphone like the IMEI and SIM card information.

- READ_PROFILE - permission required to retrieve some information related to the SIM card.
- READ_CONTACTS - permission required to retrieve some information related to the SIM card.
- WRITE_EXTERNAL_STORAGE - permission required to store all the necessary information in the smartphone.

Note that the WRITE_EXTERNAL_STORAGE is only used in devices that are equipped with Android 5.1 or below. This happens due to, as explained above, these versions of the Android Operating System not allowing symmetric keys to be stored in the device's TEE. The version of the application directed to Android 6.0 and above does not requires this permission.

The collection of information is implemented by an Android service that is started by the application whenever the user logs in or registers and the service is stopped whenever the application is dismissed or closed. This service uses Android's Telephony Manager to retrieve all the information related to the SIM card of the smartphone, the IMEI and the Software Version. The MAC address, the IP address (which is used to calculate the location) and the Service Set Identifier (SSID)s of the memorized networks are retrieved using Android's WiFi Manager.

The Operating System version is read directly from the String VERSION.RELEASE and the SDK version is read directly from the String VERSION.SDK_INT. The memorized accounts are retrieved using Android's Account Manager, and all accounts are iterated in order to separate the Google Accounts from the other types of accounts. The screen resolution is retrieved using Android's Window Manager which has methods that return the screen's width and height. The keyboard language and input methods list is retrieved using Android's Input Manager with the keyboard language being read from the Locale of the input method in use. Finally, the list of installed applications is retrieved using Android's Package Manager and filtering out the system applications.

Both the asymmetric RSA key pair and the symmetric AES key used are stored using the Android Keystore, which securely stores the mentioned keys in the device's TEE whenever possible. In this case, all the operations performed using these keys are performed inside the device's TEE and all the key material generated during these operations are stored in the device's TEE as well. This means that the operations and the key material used are not stored in normal world memory at the anytime, which means that an attacker is not able to obtain information even if he is capable of reading the device's memory.

3.4.1 Problems found during the Implementation phase

After doing a preliminary test phase with real users it was possible to find three major problems that stood out. The first one was detected while calculating the median number of memorized accounts, it was immediately noticed that only few users had memorized accounts stored in the database. Even though this situation is possible, given that only few applications store user accounts in the smartphone, it is unlikely that the major part of the users would not have any stored accounts, especially since some highly used social networks like Facebook store the corresponding user accounts in the smartphone. After noticing that some of the users without stored accounts corresponded to users using Android 6, it was found that the application lacked asking for user consent to use the permission GET_ACCOUNTS, which is required in Android from version 6 onwards to access stored accounts in the smartphone, among other information.

The second major problem was a misimplementation of the fallback mechanism on the server, whenever the retrieved profile did not match the stored one the server and the user asked the application to retrieve a new profile, introducing the SMS verification code, the server would try to find a stored profile with the same IMEI as the new profile acquired that should be stored. Since some minor bugs were being fixed whenever they were found and the users were installing the application multiples times during the preliminary test phase, the keys stored in the TEE would be overwritten with every new installation making it impossible for the server to match the fresh profile's IMEI with an existing profile, preventing that way users from updating their profile. This situation was particularly problematic since it would also prevent users from having more than one profile stored and in the eventuality that the keys are erased from the TEE (which may happen if the user clears the application data, for example) the user would also be prevented from creating a new profile.

The third major problem was the reason of most logins failing. While looking at the logs generated in the preliminary test phase in order to understand what made some logins fail that it was possible to notice that some users would send a list of memorized networks in one login and send an empty list in the following login, sending a full list again on the following login.

After talking to some of these users trying to understand if they were removing memorized networks and adding then a list of different ones, users claimed to rarely delete networks and when asked they also claimed that some logins were done using wireless networks and another logins were done using mobile data.

After looking in the documentation of Android, it became clear that whenever users logged in using mobile data with their smartphone's wireless function turned off the application was unable to retrieve the list of memorized networks, since wireless must be turned on in order to retrieve the memorized networks. Since it is possible in all Android versions so far to change the WiFi state using the WiFi Manager, all it took to fix the problem was adding the permission CHANGE_WIFI_STATE to the Manifest file and then using the WiFi Manager to turn on the WiFi functionality (if it was turned off) right before retrieving the profile, turning it back off again if the user was not using it whenever the users closes the application.

The application also suffered a substantial change in the workflow regarding rejected profiles. This change was not due to some specific problem but rather a design decision to improve the workflow of the application, requiring fewer actions from the user and helping the users relate more the application with the existing alternatives. Previously, whenever a profile was rejected, the user would be presented with a button that would allow the application to collect a new profile, send it to the server and requiring the user to confirm the creation of the new profile by correctly introducing a code that would be received via SMS. In order to make the application work more closely to existing alternatives, the button phase was eliminated. Whenever a profile is rejected the user receives a code via SMS that the user should input. The application will then acquire the new profile and send it to the server alongside the inputted code. If the code matches the expected one, the server will then store the new user profile. In the existing alternatives the user receives an SMS code in each login without being required to press buttons for it to be sent. With this change in the workflow the user is immediately authenticated or automatically receive an SMS code, having this way a better perception on the improvement in the number of performed actions whenever the profile is accepted.

3.5 Attacker Models

As presented in Chapter 2 with the works of Cooijmans et al [13] and Loftus and Bauman [25], the proposed solution is not vulnerable to common attacks since it does not require any input from the user (which can be retrieve with a keylogger) or flawed biometric authentication such as fingerprints. As described by Android and further confirmed by Cooijmans et al, the keys stored in the TEE are also secured against being extracted from the device, even though they can be extracted from the corresponding application sandbox.

Taken into account the attacker models introduced by Cooijmans et al (which are also used in the works of Bouazzouni [10] and Sabt [28]) the only relevant attack vector is an attacker who extracts the keys into his own malicious application. Since the proposed solution uses the Android Keystore, the attacks that are possible to perform against the Bouncy Castle Keystore are not considered. As described by Cooijmans et al, extracting keys directly from the device is impossible.

Considering once again the results of Cooijmans et al and the attack vector described above, the ”Malicious app attacker” model is further discarded since the attacker described by the model is unable to perform the attack, since keys cannot be read using a different application if the device is not rooted. It is further irrelevant to consider both the models ”Root attacker” and ”Intercepting root attacker” since the only difference between the two models is the capability of intercepting user input which is not relevant in the proposed solution. The only usage of user input in the implemented solution is done in the fallback mechanism, which security is out of the scope of this work.

Therefore, the model of the attacker considered in the security evaluation of the system is the attacker described by Cooijmans et al as the ”Root Attacker”. This is a type of attacker who has root capabilities and is able to run applications with root permissions. This attacker is also able of inspecting the Android file system. It is assumed that the device was previously rooted or the attacker has gained root capabilities through a malware capable of Privilege Escalation attacks.

Chapter 4

Evaluation

Contents

4.1 Tests with Users

In order to test the application with real-world users, group of users were selected that showed availability in downloading the application and using it on a regular basis. Those users were also required to fill-in a preliminary inquiry whose results will be presented in Section 4.1.1. As explained in Chapter 3, during the first weeks of the First Test Phase the application was fixed as several bugs that made the application crash were found.

The application was deployed without major bugs to the users in the beginning of June 2017 and the users were asked to perform logins until July 2017. Every login was recorded and the results will be presented in Section 4.1.2.

As explained in Chapter 3, several problems were found while analyzing the resulting logs. There was a bug that made the application crash whenever a new profile was required. Accounts were not being retrieved in Android 6 onwards since it was missing explicit user consent to use the required permissions. There was also a problem regarding people who logged in using both wireless networks and mobile data, whenever the user logged in using mobile data with the wireless function turned off the system was unable to collect the memorized networks, which resulted in the system sending a blank list in those cases which made the verification fail since the memorized networks list was significantly different.

After fixing those problems a Second Test Phase was deployed. Users received the final version of the application in the end of July 2017 and were asked to perform logins until mid-September 2017. Every login was recorded and the results will be presented in Section 4.1.3. It is also important to note that the Second Test Phase had fewer users using the application, which is explained by users being less available during Summer or even forgetting to login as often as possible.

Besides the bugs that were fixed, the verification thresholds were also changed between phases. These thresholds were changed using the results of the First Phase and were used in the Second Phase in order to validate whether the newly defined thresholds would allow less false-negatives.

4.1.1 Preliminary Inquiry

The preliminary inquiry contained the following set of questions:

- Questions about smartphone usage - these questions were important to characterize the users, creating an estimate of application usage, most used types of applications and how frequently users change the installed applications, the memorized networks and the memorized accounts.

- Questions about the value of information contained in the smartphone - users were asked whether they would login in a friend's smartphone, in which applications would they login, the types of applications with information considered sensitive by users and whether users login in sensitive applications only in known networks or in public networks as well. The location of those logins was also considered in the questions.

- Questions about Two-Factor Authentication - users were asked whether they use Two-Factor Authentication, in which applications they use it and reasons for not using it whenever possible.
- Questions about the proposed solution - users were asked whether the proposed solution would be interesting and what they like the most in a Two-Factor Authentication scheme.

The inquiry was filled by 13 users. The full results of the Preliminary Inquiry can be seen in Appendix A. In this Section only the most interesting and relevant results will be addressed.

The most used types of applications are Messaging Applications and Social Networks, as can be seen in Figure 4.1.

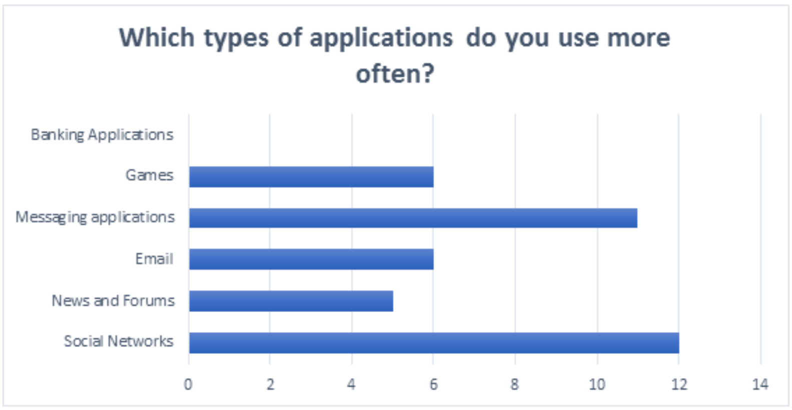


Figure 4.1: Most used types of applications

As can be seen in Figures 4.2, 4.3 and 4.4, users claim to install applications, and store networks and accounts with a relatively low frequency, which turned out to be false. Users install more frequently applications than they store networks. Users store more frequently networks than accounts.

Regarding the sensitivity of the information, Banking Applications, Social Networks, Email and Messaging Applications contain the most valuable information to the users, as seen in Figure 4.5. Users also claimed to not have major concerns with their location when using applications that contain sensitive information. Users showed a major concern with the types of networks used to access sensitive applications, with the major part of the users claiming to login using mobile data or their own private network.

Users use Two-Factor Authentication quite often, especially in sensitive applications, with the most preferable scheme used being codes sent via SMS or email, as can be seen in Figures 4.6 and 4.7.

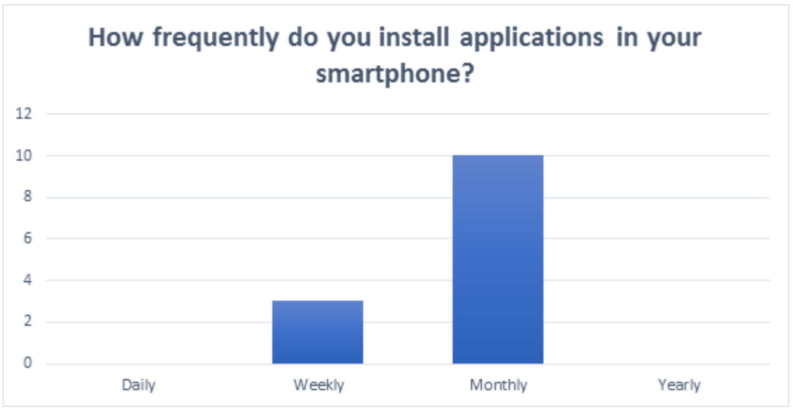


Figure 4.2: Application installation frequency

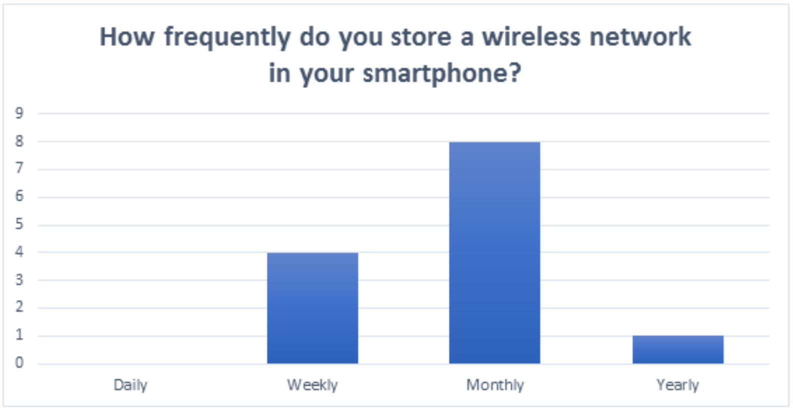


Figure 4.3: Network storage frequency

The biggest obstacles in using Two-Factor Authentication for every application consists of the fact that those schemes slow down the authentication process, requiring the users more actions than they desire, as can be seen in Figure 4.8. Another reason to not use Two-Factor Authentication whenever possible is that the information contained in some applications is not worthwhile protecting with a slow form of authentication that requires lots of steps in order to increase its security.

Regarding the proposed solution, users showed desire in having a fast authentication system that requires few actions, as can be seen in Figures 4.9 and 4.10. When asked if the information contained in the smartphone would be unique enough, users expressed some uncertainty, as can be seen in Figure 4.11. It is important to note that users want to control the information sent to the system, or at least, they do not want their information leaked to other parties. Users are also quite certain that information contained in a smartphone is hard to falsify.

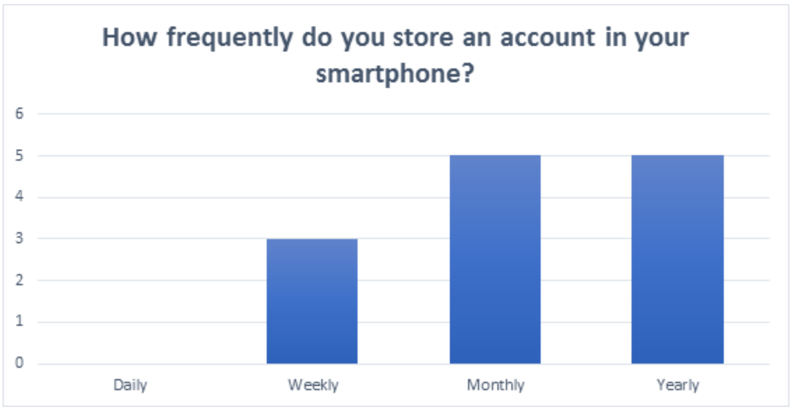


Figure 4.4: Account storage frequency

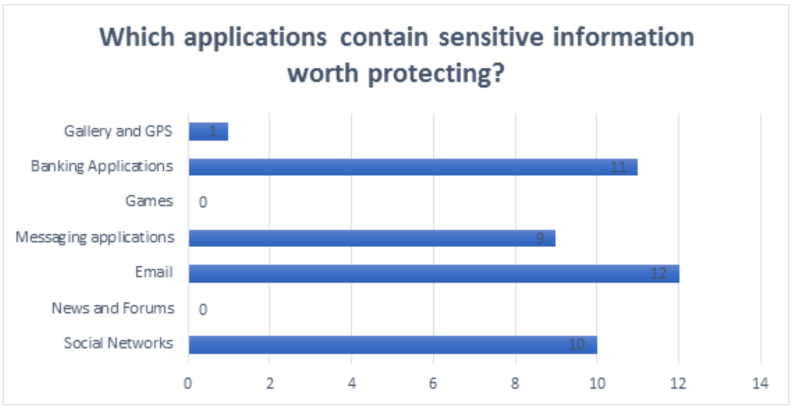


Figure 4.5: The applications that contain the most sensitive data

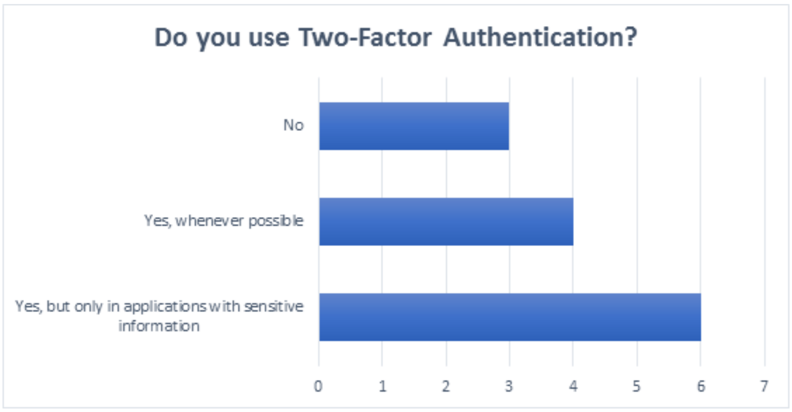


Figure 4.6: Usage of Two-Factor Authentication

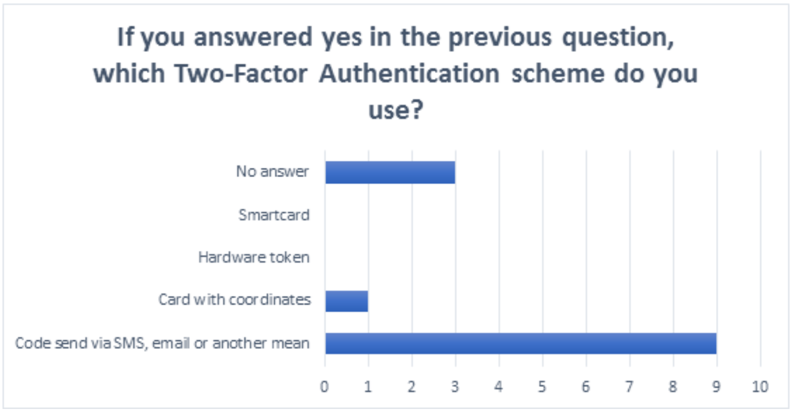


Figure 4.7: Most used Two-Factor Authentication schemes

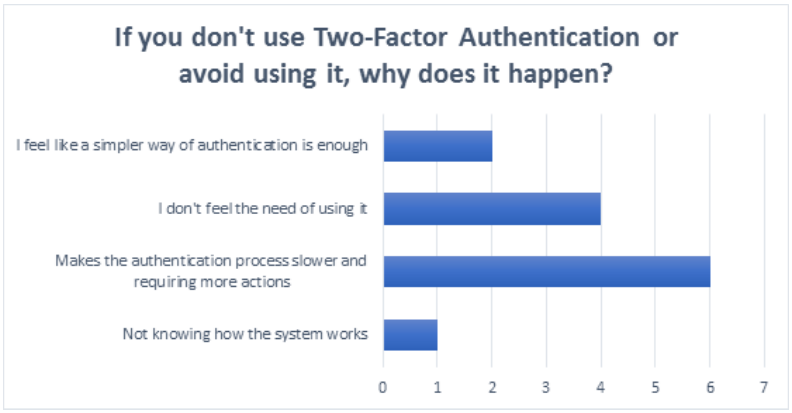


Figure 4.8: Major obstacles to avoid using Two-Factor Authentication schemes

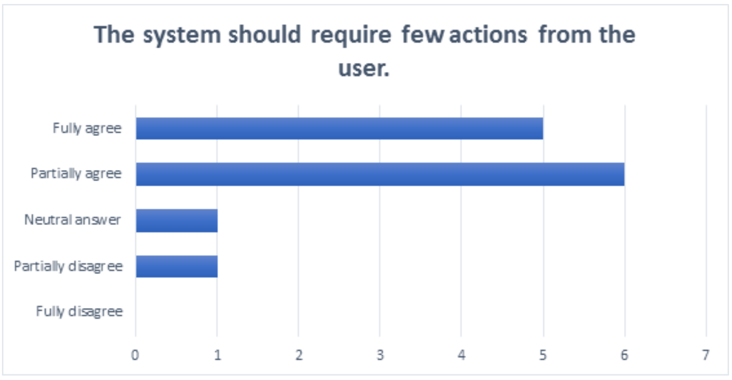


Figure 4.9: Preferred types of authentication, as reported in the Preliminary Inquiry

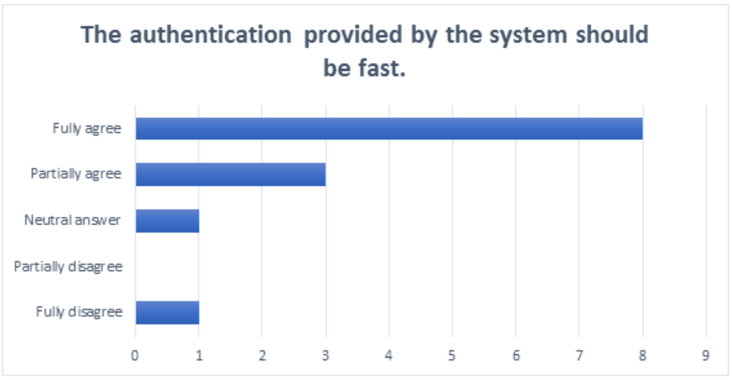


Figure 4.10: Preferred types of authentication, as reported in the Preliminary Inquiry

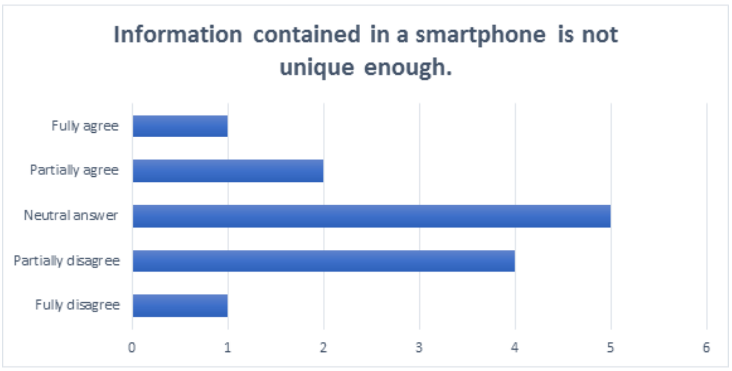


Figure 4.11: How unique is the information contained in smartphones, as reported in the Preliminary Inquiry

4.1.2 First Test Phase

In the First Test Phase, 19 users participating performed a total of 78 logins. Despite the number of logins being relatively low considered the number of users participating, it is still useful to estimate how the profile changes over time. The median value of logins is 4 per user, which roughly gives an average of 1 login per week per user.

The results are described in the Figures 4.12 and 4.13: out of the 78 recorded logins, only 44 were considered correct by the system, whereas 34 logins failed in profile verification. This was further investigated, and the major problems have been described in Chapter 3 and the beginning of Section 4.1.

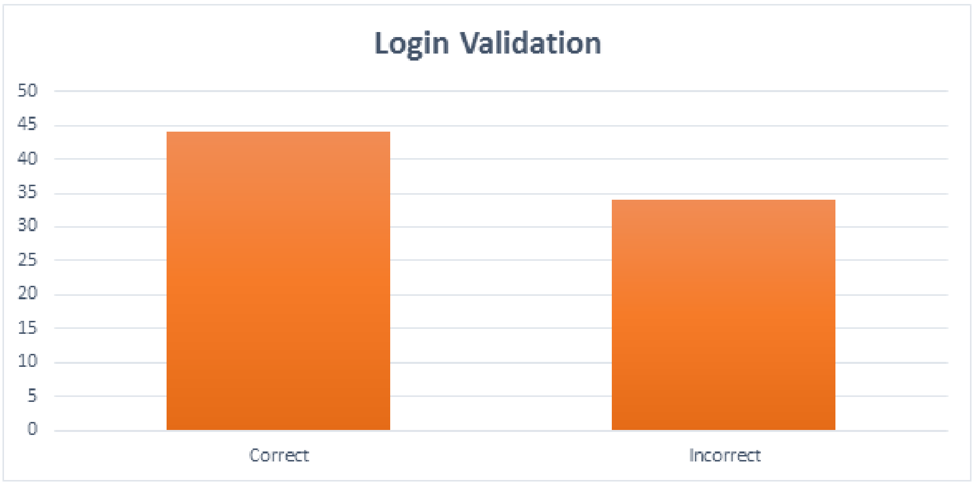


Figure 4.12: First Test Phase Login Validation

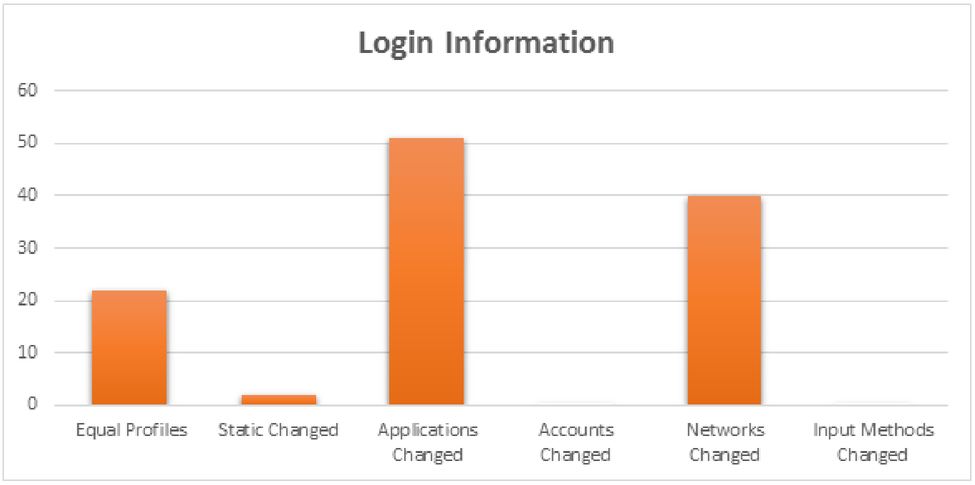


Figure 4.13: First Test Phase Login Information

Regarding the types of changed attributes, it is important to note that half of the correct logins corre-

Attribute	Median number (per user)
Google Accounts	1
Input Methods	2.42
Installed Applications	53
Memorized Accounts	4.5
Memorized Networks	27.26

Table 4.1: Median number for each attribute after First Test Phase

sponded to profiles exactly equal to the expected profile. Both logins that failed due to a static attribute being changed corresponded to a change in the Software Version, which can be considered as the attribute being wrongly classified, since it is somehow frequent to change.

Considering the large number of logins with changes in the networks (40 logins) and the problem that prevented the system from retrieving networks when using mobile data, it is easy to understand that the rate of correct logins had the potential of being higher.

Another important remark is the fact that there was not a single change on accounts recorded, which as explained before is also due to the fact that a large part of the user base was using Android 6 or higher and the application was not asking the required user consent to use the permissions needed to retrieve the stored accounts.

Considering only the logins where dynamic attributes have changed, which accounts to 54 logins, since in 22 logins nothing has changed and in 2 dynamic attributes were not considered due to static attributes being changed, it is important to note that Applications were changed in 51 of these logins and Networks were changed in 40 of these logins which accounts for 94.4% and 74.1% logins, respectively.

In order to make use of the data provided by the users to adjust the values and to estimate with more precision the percentage of changes in installed applications and memorized networks the following mean values were calculated by looking at the profiles stored in the database. The results can be seen in Table 4.1.

The mean values are somehow easy to understand, Android smartphones require one Google Account in order to allow the user to download applications from the store and use some features. Users usually use their existing Google account or create one when changing to a new phone but they usually do not associate another Google account to the device, since the major part of applications allow users to use more than one account and these accounts are not stored in the device but in the corresponding applications.

The values of Input Methods and Installed Applications are also easy to understand considering that some manufacturers install by default several keyboards with different languages and character sets and also install some manufacturer specific applications like browsers, custom definitions, bloatware and other branded software which increases the number of applications, especially considering that in most cases those applications cannot be removed without root permissions.

The number of Memorized Accounts is also easy to understand, since some social networks (Facebook stores one for its application and another if the user uses Facebook Messenger) and messaging applications (WhatsApp) store accounts in the device. Given the problems regarding retrieving the stored accounts in Android 6 and onwards it was only possible to obtain the accounts of 4 users out of the 19 users that tested the application. This way, it was hard to make a good estimate of the median number of accounts but despite that the number seems like an acceptable guess.

The oddest number is the high value of memorized networks, but it may be explained by users who connect to diverse public hotspots and never clean the list of memorized networks afterwards.

As explained in Chapter 3, after fixing client application problems the value of the define thresholds was also revisited taking into account the estimated medians calculated after the First Test Phase. The threshold of memorized accounts and input methods was changed from 75% to 66% to accept a bigger number of changes in both lists.

The number of Google accounts was ignored since changes in this attribute are not only less frequent than changes in another attributes but it is also hard to effectively track them. It cannot be assumed that one user will only have a Google account since it is possible to store more than one account of this type but it is highly unlikely that a user stores more than one Google account in the device. The attribute cannot be considered static but it is almost impossible to define an acceptable threshold, therefore it is easier to just use the defined fallback mechanism to deal with changes regarding Google accounts.

4.1.3 Second Test Phase

In the Second Test Phase, only 12 users chose to participate, which is rather unfortunate since the testing period was bigger than the period of the first phase and the application was not displaying any bugs or problems.

As explained in Chapter 3 the workflow of the application was also changed making the application behave more closely to the existing solutions, which would help users understand the purpose of the application and the advantages of the solution.

Similarly to the first phase, 78 logins were recorded, which translated into a mean value of 6.5 logins per user during a 6 weeks period, which corresponds to one login per week per user.

The results are described in Figures 4.14 and 4.15 and can be considered a huge success: out of the 78 recorded logins, only 5 were considered incorrect by the system, one of them with a failed static attribute, which means that the calculated probability of the fallback mechanism being activated which requires further actions from the user is only 6.41% which translates into a huge advantage when compared to existing Two-Factor Authentication schemes that require lots of actions from the user like the SMS code verification, biometrics, among others.

The system still needs further testing since a larger user base and more regular logins might find

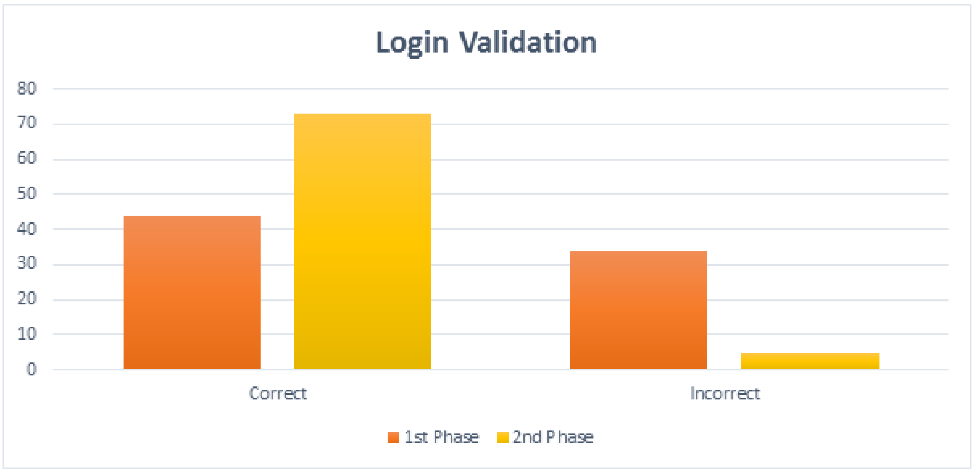


Figure 4.14: Second Test Phase Login Validation

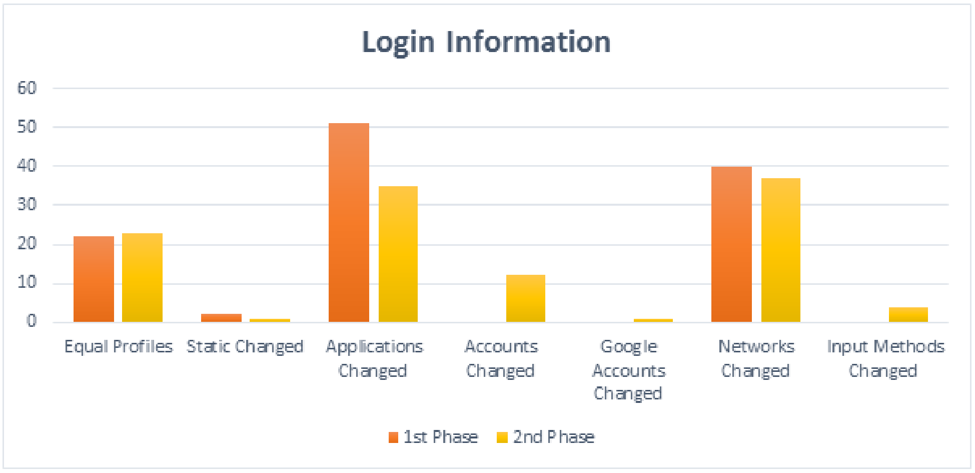


Figure 4.15: Second Test Phase Login Information

some unexpected problems that need attention but the results are definitely promising especially keeping in mind that daily logins will have a smaller number of changes than weekly logins which means that the profiles will always be closer to those expected. One interesting experiment is to tighten the thresholds, which are completely realistic at the moment but there is still room for being more strict.

The 5 failed logins had the following problems: one of the failed logins corresponded to a change with the MAC address attribute which is a strange result since the user made several logins before the failed one with a MAC address and then it completely changed and stayed constant in the following logins. It is hard to understand how and why the user changed his MAC address but the system behaved as it should.

The remaining four failed logins consisted on a user that lost the key used to create the registration profile which can happen if the user clears the application data, two users that changed a large number of memorized networks and a user that changed his stored Gmail account (note that he did not an extra

Attribute	Median number (1st Phase)	Median number (2nd Phase)
Google Accounts	1	1
Input Methods	2.42	2.25
Installed Applications	53	49
Memorized Accounts	4.5	3.66
Memorized Networks	27.26	27.92

Table 4.2: Comparison of the medians of each attribute in both login phases

account, but he rather changed the existing one for another).

Out of the 73 correct logins, only 23 were exactly the same as the expected profile, which is a lower percentage than in the first phase but the difference is also easy to understand since accounts were not being retrieved in the first test phase, for example.

Considering the 54 logins where dynamic information has changed, applications were changed in 35 logins which makes 64.8% of the logins, networks were changed 37 times which makes 68.5% of the logins, accounts were changed 12 times which makes 22.2% of the logins and input methods were changed 4 times, making 7.4% of the logins.

Despite only 12 users using the application in the second test phase, some of them were new users and it was interesting to calculate the median number for each attribute to compare to the values obtained in the first phase to see whether they remain constant or changed a lot. Another problem was that the estimate of the mean number of accounts was done using only 4 users in the first phase, therefore the mean values of the dynamic attributes had to be calculated again after the second test phase. The values can be seen in Table 4.2.

The values are in general close to the values calculated in the first phase. As expected the number of Google accounts is still one per user which makes sense since even the user that changed the Google accounts attribute replaced one account for another.

The number of Installed Applications and Memorized Networks is close to the estimates calculated in the first phase which is also consistent to the fact that these two attributes are the ones that change more frequently. The thresholds should remain the same or even tighten since the number of changed applications and networks is usually not very significant.

The number of Input Methods and Memorized Accounts changed the most given that those lists are significantly smaller than the others and require further testing and validation. The number of Input Methods did not change much (from 2.4 to 2.25) but is getting closer to 2, which requires attention and a possible different threshold than the defined one. The current threshold of 66% would not allow a user with 2 input methods stored to add another, for example.

Similarly, the number of Memorized Accounts decreased almost 1 account per user (from 4.5 to 3.66) which creates the possibility of a change in the threshold. A user with 3 stored accounts would not be able to add 2 new accounts at a time without triggering the fallback mechanism.

4.2 System Security

4.2.1 Threat Analysis - STRIDE model

STRIDE is a Microsoft threat classification model that contains six threat categories: **S**poofing Identity, **T**ampering Data, **R**epudiation, **I**nformation Disclosure, **D**enial of Service, **E**levation of Privilege.

4.2.1.A Spoofing Identity

There are two ways for an attacker to attempt Spoofing User Identity attacks in the system developed in this work. Since the profile is created using a symmetric key stored in the smartphone, an attacker wishing to spoof the identity of a rightful user must be capable of creating fake profiles for a given user, or in alternative, replaying old messages sent from the respective user to the server.

Since the server generates a different nonce for every user action and checks if the received messages contain the right nonce, discarding the messages if the nonce is invalid, replay attacks are not possible in the system.

Regarding the creation of fake user profiles, in order to successfully create a profile that is accepted by the server an attacker needs access to the symmetric key stored in the user's smartphone since it is used to create the profile and used by the server to verify the profile. The attacker also needs to use user information to make the forged profile match the stored profile.

The attacker may obtain the required information about the user through malware installed in the rightful user's smartphone or through brute force attacks if the symmetric key is known by the attacker. In the latter case, an attacker can also make use of some social engineering knowledge to infer relevant information. For example, if the attacker knows that a user is an active Facebook user and knows the user's account name (which is public in some social networks) an attacker may add that information to the profile.

Even though the symmetric key is stored in the Android Keystore which uses the TEE whenever the smartphone's hardware supports it, some attacks may still be possible, especially if the smartphone is rooted. These attacks will be explained in detail later, taking into account realistic attacker models.

4.2.1.B Tampering Data

Tampering Data attacks are not possible in the system since the communication between user and Profiling server is done through HTTPS, which protects against message tampering. The messages exchanged in all communications are also signed to guarantee the authenticity of the message, which also makes it easy to detect tampered messages.

4.2.1.C Repudiation

Repudiation is an issue in the system since, as said above, all messages exchanged in every communication between user and Profiling server are signed by the sender.

4.2.1.D Information Disclosure

Information disclosure is in practice impossible in the system, since the profile is a set of hashes and all communications are done through HTTPS. An attacker would need to break the HTTPS encryption to obtain the profile and then the attacker would need to break the hash function.

In the case where the attacker obtains access to a profile, he would need the symmetric key to either brute force the profile or break the HMAC function. As discussed above, attacks against the Android Keystore storage will be further discussed taking into account attacker models. The symmetric key is also shared with the server but it is sent encrypted with the server's public key, so in order for an attacker to obtain the key he would need to break both HTTPS encryption and then RSA encryption.

The Profiling server stores all the necessary information in an encrypted MySQL database which also mitigates information disclosure attacks against the server storage. Further attacks against the server are also not considered, with the server being assumed secure.

Furthermore, most of the information that could be disclosed in the highly unlikely events described is uninteresting or even harmless, revealing very little about the user. The major threat with information disclosure is the possibility of spoofing identity attacks since the attacker would know the relevant user information and the key used to create the profile.

4.2.1.E Denial of Service

It is possible to flood the Profiling server with fake requests or replays of old HTTPS requests which delays or even denies the server from responding to rightful requests made by the users. This problem can be mitigated, even if not completely eliminated, by the use of a scalable architecture for the server implementation and deployment.

4.2.1.F Elevation of Privilege

Elevation of Privilege attacks are possible in the Android Operating System and those attacks are particularly threatening to the system since they may allow an attacker to obtain access to the symmetric key used to create profiles.

4.2.2 Security Evaluation of the System

After concluding the Threat Analysis of the System, it is possible to note that the most relevant attacks consist of an attacker obtaining access to the keys stored in the smartphone, which would make an attacker capable of creating a false profile, signing messages, obtaining the nonces sent by the server or even disclose the information contained in profiles.

The information is stored on the Profiling Server using encrypted databases, which means that an attacker is not capable of retrieving the keys or the profile information directly from the server without being able to break the encryption of the databases.

Therefore, the security of the system is dependent on an attacker being able to fully control the keys stored in the Android Keystore.

4.2.2.A Assumptions

In order to evaluate whether an attacker is able to subvert the system, the following assumptions are made:

- The server is secure and trusted, it is impossible for an attacker to retrieve information stored in the server. It is important to note that even if the symmetric keys are stolen from the server, all that is required to make sure the devices are not compromised is to force all users to reset their keys, making the stolen keys useless.
- Denial of Service attacks are not considered, due to not being relevant to the keys stored itself but to denying the server to properly respond to the users.
- Communications are assumed to be secure. It is assumed that attackers cannot break HTTPS encryption, RSA asymmetric key encryption and *SHA1withRSA* digital signatures. An attacker is unable to eavesdrop messages, tamper messages by forging the digital signature and incapable of decrypting the symmetric key shared between user and server.

4.2.2.B Attacker Models

Considering the Attacker Models defined in Chapter 3 and the Threat Analysis done in this chapter, the major security threat to the proposed solution consists on an attacker with root capabilities who is able to extract the keys from the Android Keystore and use it on his own malicious application.

In that case, given that the attacker would have the key and the capability of reading the smartphone's information, the attacker would become able to slowly start to deface the profile of the rightful user by creating false profiles. It is important to note that the malicious application may send the retrieved key to a server controlled by the attacker, but even in that case the attacker would not be able to completely change the profile without triggering the fallback mechanism.

The most impactful attack that can be performed with this information is an attacker which is somehow able to emulate the victim's static smartphone attributes and uses the key earned through the attack to impersonate the real user. Another variant of this attack would be the attacker stealing the smartphone and using it to authenticate, however this attack is mitigated if the user reports the theft and the profile is deactivated.

However this attack is extremely powerful it is unclear how easy it is for an attacker to create malware capable of retrieving keys stored in the Android Keystore and emulate another device. It is important to note as well that the attacker needs root capabilities to perform this attack which is only obtainable through another type of malware if the device does not have root enabled by default or if the user did not enabled it.

Even though the damage may be considerable, the victim does not need to replace his own smartphone to solve the problem but rather the victim just needs to disable root, wipe all the data in the smartphone and request a new key through the fallback mechanism.

Due to Android Keystore being restructured in Android version 6.0 and the lack of recent studies regarding the mentioned vulnerabilities it is unclear whether these vulnerabilities have been fixed. The TEE prevents keys from extraction if the device is not rooted, but it fails to prevent keys from being extracted from a root device. Therefore, it is recommended that root should be disabled unless the user really needs root capabilities, since it can be exploited by attackers to perform a wide range of attacks against the device.

4.3 Evaluation Summary

Considering the objectives defined in Chapter 1, it is possible to conclude that:

- Fast Authentication that requires few actions - Considering the best case, the system is able to successfully authenticate the user in a short period of time without requiring any actions from the user. Using the results of the Second Test Phase, the best case happens 93.6% of the times which roughly means that in every 10 logins the user is required to use the fallback mechanism once, which means that 9 out of 10 times the proposed solution is better than the existing solution in the number of required actions. Regarding the speed of the system, profile verification is optimized in a way that the system takes more time with more changes, but even in the case of numerous changes the system takes few seconds to authenticate the user.

- Security - The only possible attack is a targeted malware that uses root permissions to retrieve the keys used to create the profiles. Despite this attack being extremely impactful, existing solutions are also vulnerable to several types of malware especially if root permissions are available to the malware. Solutions like the SMS code also have the drawback of being targeted by malware that is capable of reading the user input, being that way vulnerable to cases where an attacker receives the code as the user inputs it and introduces first than the user. Therefore, the proposed solution is at least as secure as the existing alternatives.

- CPU and Memory usage - The system does not store anything in memory since the keys are stored in the TEE and the cryptographic operations done there as well. CPU usage is also very low since the application only calls defined Android APIs to retrieve all the necessary information. This can be seen in Figures 4.16 and 4.17, which were retrieved using the CPU and Memory Monitor of the Android Studio, while performing 3 consecutive logins.
- Network usage - The system only exchanges a couple of HTTPS messages with the server, which reduces any overhead introduced into the network. This can be seen in Figure 4.18, which was retrieved using the Network Monitor of the Android Studio, while performing 3 consecutive logins.
- Battery usage - Since the application only runs the collector service when needed and stops it when the user exits the application, the overhead in battery usage is minimal. Location is not calculated using GPS which also helps avoiding reducing the battery life.

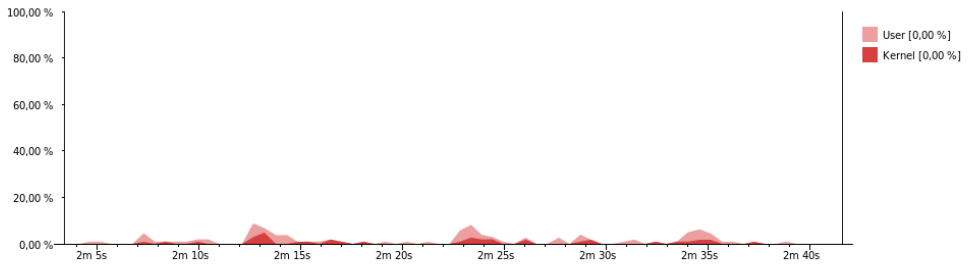


Figure 4.16: Application CPU usage, measured with Android Studio's CPU Monitor

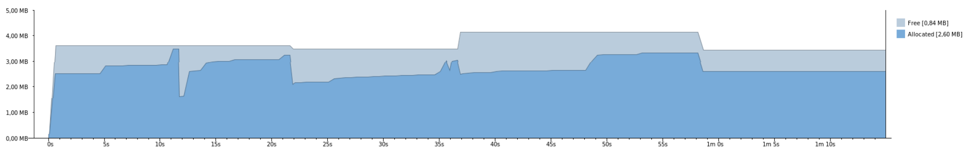


Figure 4.17: Application Memory usage, measured with Android Studio's Memory Monitor

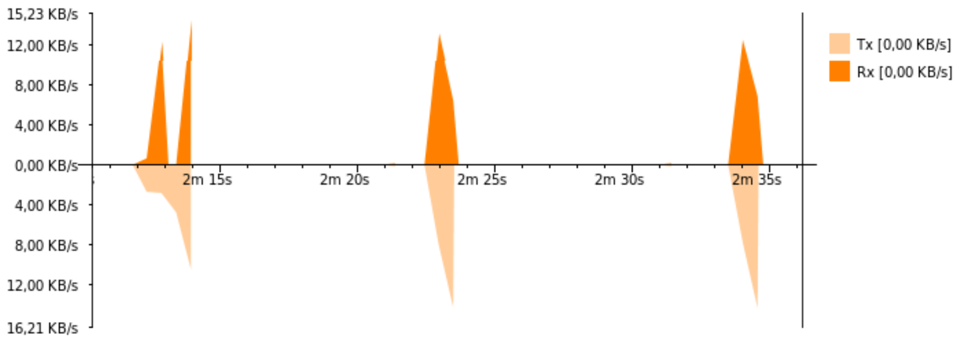


Figure 4.18: Application Network usage, measured with Android Studio's Network Monitor

Chapter 5

Future Work

Considering the scope of the work and the work that was done, there are three major directions to follow in the future. These directions can all be combined into a final product since they all complement the system without being mutually exclusive, however implementing some of these ideas means that the others would have to be adapted in order to work since the implementation would change the existing system. The directions are the following:

- Ask for parts of the profile instead of the whole profile - this direction is perhaps the most immediate to accomplish in a near future. It consists in defining some fixed attributes that should be sent all the time and a set of attributes that may or not be asked by the system.

  This idea is based on the premise that a subset of the 20 retrieved factors would be able to uniquely authenticate the user in a secure way and would further increase the security of the entire system since an attacker that is able to break one message once does not get an idea of the whole profile but only a small part of it.

  The implementation of this idea would require a thorough study on all the retrieved factors in order to create subsets of factors that could possibly verify a profile. Randomly asking for subsets would not work since it may consist of several factors who are highly likely to suffer changes making it hard to verify a profile. Implementing this idea would probably require changes to the retrieved factors and the fallback mechanism and the Profiling server would need to suffer deep changes in the way it works.

- Machine Learning algorithms to support the decision of verifying a profile - this direction is perhaps the most relevant to the work performed as it provides a different type of verification that does not rely only on counting the common elements from profile to profile but a different type of verification that would take into account the profile of the user. The biggest application of this extension to the system would be directly related to list attributes, since other attributes are harder to evaluate and predict. This solution works even with the usage of the hashes since the hash of each category would generated the same constant output.

  A possible use case would be the system considering not only the name of each installed application but also its type, based on the premise that a user that has different applications of a type is more likely to install or uninstall applications from the same type than from a different one. Android 8 will make it possible for applications to state their type in the Manifest file, making it possible to know the declared type of each installed application in the smartphone using Android's Package Manager.

  If machine learning algorithms were implemented the algorithm would be able to understand that even though the major part of installed applications was changed, the main type of the applications are the same corresponding to the same user.

Another possible use case takes into account the fact that some attributes are linked to others. For example, if a user travels to a different country, joins a wireless network there and logs in, profile verification will notice changes in both the memorized networks and the location. While in the current verification system these changes could imply that the verification would fail if more attributes were changed, this is a legitimate situation where the fallback mechanism would be activated and the user has not done anything wrong.

The use of Machine Learning algorithms could detect that those attributes changing at the same would imply a travel to a foreign country and possibly accept the profile if the remaining changes were not significant enough. Investigation in this direction would imply to think in further security questions. The idea is to help the user in not being forced to use the fallback mechanism in a regular usage of the system but these situations would possibly be exploited by some attackers and must be taken into account with precaution.

- Extend the solution to accept profiles from multiple devices - the system already accepts multiple profiles per user, however an interesting and challenging approach would be to store only one profile and use multiple devices to authenticate the user. The main idea would be to extract key user configuration features in order to check whether these features are present in a different smartphone or not.

  This idea is based on the premise that a user is highly likely to install the same set of applications in the smartphones he regularly uses, similarly memorizing a set of accounts and networks as well that is present in all the smartphones as well.

  An example of this premise is a user who actively uses social networks and has a stored account for each of these applications. This user also has memorized his own home wireless network, the network from his office and the network from some friends' houses. If the system would be somehow capable of detecting all these common elements and if the number of common elements was higher than a given threshold the system would be able to accept a profile retrieved from a different smartphone than the one used to register.

  The biggest challenge towards implementing a solution based on this idea consists in securely sharing the symmetric key used to create the profile among the devices used by the same user, since if different keys are used the system is not able to understand which are the common values.

Chapter 6

Conclusion

Online authentication has been done through the years using a username and a textual password. While this system is relatively simple, since everything the user needs is to remember his username and password, this system has been proven insecure due to problems regarding textual passwords.

Among other problems, short passwords are easy to crack using brute force attacks and in some cases easy to guess through social engineering attacks. A particularly simple case happens when the password is something related to the user. Long passwords are harder to guess and crack but it is also harder for the users to remember, especially if the user uses different passwords for different services.

In mobile devices, some solutions to this problem include the generation of a One-Time Password that is sent to the user through SMS or through an application. The user is then required to introduce the received One-Time Password in order to authenticate himself in the desired system.

Even though these systems solve many problems related to textual passwords and can be considered secure, users in general avoid using this type of authentication. This happens due to the users being required to perform additional actions like checking, copying or memorizing the code and inputting it in a matter of seconds.

A preliminary inquiry done with smartphone users (whose results are described in Chapter 4 and Appendix A) show that even though users consider the information contained in the smartphone valuable they avoid using Two-Factor Authentication whenever possible since it increases both the duration of the authentication process and the number of required actions.

The solution described in this work consists in a new Two-Factor Authentication scheme that uses smartphone characteristics and its user configuration, like SIM card information, installed applications, memorized accounts and networks, in order to create unique user profiles that will be used to authenticate the user.

In the proposed solution a profile is a set of hashes calculated for each individual attribute, the whole profile is then used to calculate an HMAC using a symmetric key stored in the device's TEE. The profile in plain text and its HMAC are sent alongside to the Profiling Server. That way, the privacy of the profile is guaranteed since the client is the only one who can really know the value of the attribute, since there is no way of inverting a hash function. The authenticity of the message is also guaranteed since only the rightful user may send the right HMAC.

There are two types of attributes: static attributes and dynamic attributes. Static attributes are attributes related to the device's hardware (like for example the screen resolution) or with the device itself (IMEI, MAC address) and are not supposed to change. Dynamic attributes are attributes related to the device configuration made by the user and can change, examples being SIM card information, installed applications, operating system version, among others.

Whenever a client tries to login, if the username and password are the correct ones, the third-party application will call the information collector application which will retrieve all the necessary information to create the profile, sending the profile information to the Authentication Server.

The Authentication Server is then responsible for the verification of the received profile, since dynamic attributes are by definition capable of being changed over time, the server must be able to accept minor changes in the profile.

Whenever static attributes are changed profile verification fails, since it means that the device is not the same as the device used to register the profile. Regarding dynamic attributes, there is a defined threshold of 75% which means that in every login, 11 out of the 16 dynamic attributes should remain constant.

Since lists like installed applications, memorized accounts and networks are special dynamic attributes since they are more likely to suffer changes. Its verification must be adapted as well. Instead of just checking if the lists are equal, the solution checks if 75% of the list remains constant (66% in the cases of memorized accounts and input methods) and whether the percentage of new information is not greater than 25% of the size of the new list.

If the profile verification is passed, the new profile is stored in the server, making sure that the server is always updated with the most up-to-date information and the user will then be able to use the third-party application. If the verification fails, the fallback mechanism of the third-party application is activated and the user will need to provide information to ensure his identity. The selection of the fallback mechanism is defined by the third party, being out of the scope of this work.

After two testing phases with smartphone users the correctness of the solution was evaluated. The first phase was extremely helpful to detect several problems which made the verification fail more than it should, accounts were not being retrieved in Android 6 onwards and the list of memorized networks was being sent empty whenever the user was using mobile data with the WiFi capability turned off.

The results after the second test phase were really positive, out of 78 logins made by 12 users over a month, only 5 of those logins failed the verification. This introduces an improvement in terms of requiring actions performed by the user, since in only about 1 login out of 10 the user is required of use the fallback mechanism. However, the results indicate that the solution is functional and works as intended. Further testing is still necessary with a larger group of users and a greater number of logins.

Regarding the security of the system, after analyzing all the threats, the only identified attack is targeted malware that abuses root capabilities to steal the key used to create the profile. Since there is no definite solution against malware and the security of the solution is dependent on how Android secures the keys in the TEE, the solution described in this work is at least as secure as the existing alternatives.

The solution also accomplishes the objectives of using the resources of the smartphone wisely. The solution does not make an abusive use of the CPU or the Memory since information is stored in the TEE and some secure operations are done there as well and the overhead introduced in the network is minimal since the application only exchanges a couple of messages with the server.

Regarding future work that could be done to improve the system, there are three major direction that can be followed. The first one consists of sending a subset of the profile instead of the whole profile, which would make replay attacks even harder to execute and also reduce the information sent. This improvement is based on the idea that only a small part of the profile may be enough to authenticate the user.

The second improvement consists in using machine learning algorithms in the profile verification, to support the decision of accepting a profile or rejecting it. This improvement would allow verification to accept or reject a profile based on the types of changes made and not only based on a constant percentage of attributes. An example would be a user that actively tries new games in his smartphone. If that user would uninstall several games and install a considerable number of new games, then verification would fail. However, with machine learning algorithms the system might detect that several gaming applications were uninstalled and installed, which fits the user profile.

Finally, the last improvement consists on studying if the system can be adapted to use one profile per user for all devices. Currently, each device used by the user is stored within a different profile associated with the user's account. Assuming that a user installs the same subset of applications and memorizes the same subset of networks and accounts in all his devices, verification could be changed to look for the same subset in all devices in order to authenticate the user.

The major contribution introduced by this work consists in a Two-Factor Authentication system that is fast and requires no actions from the user besides starting the system if everything works correctly, and requires as many actions as any existing solution in the worst case, which happens about 1 out of 10 times on average.

Another contribution introduced by this work consists in using smartphone characteristics and user profiling to authenticate instead of using these techniques to perform intrusion detection.

Bibliography

[1] URL: https://developer.android.com/guide/platform/index.html.

[2] URL: https://www.arm.com/products/security-on-arm/trustzone.

[3] URL: https://developer.android.com/training/articles/keystore.html.

[4] Sagar Acharya, Apoorva Polawar, and P Pawar. Two factor authentication using smartphone generated one time password. *IOSR Journal of Computer Engineering (IOSR-JCE)*, 11(2):85–90, 2013.

[5] Murad Ali, Zubair Shaikh, Muhammad Khan, and Taha Tariq. User profiling through browser finger printing. In *International Conference on Recent Advances in Computer Systems*. Atlantis Press, 2015.

[6] Fadi A. Aloul, Syed Zahidi, and Wassim El-Hajj. Two factor authentication using mobile phones. In El Mostapha Aboulhamid and José Luis Sevillano, editors, *The 7th IEEE/ACS International Conference on Computer Systems and Applications, AICCSA 2009, Rabat, Morocco, May 10-13, 2009*, pages 641–644. IEEE Computer Society, 2009. URL: http://dx.doi.org/10.1109/AICCSA.2009.5069395, doi:10.1109/AICCSA.2009.5069395.

[7] Chandrasekhar Bhagavatula, Blase Ur, Kevin Iacovino, Su Mon Kywe, Lorrie Faith Cranor, and Marios Savvides. Biometric authentication on iphone and android: Usability, perceptions, and influences on adoption. *Proc. USEC*, 2015.

[8] Károly Boda, Ádám Máté Földes, Gábor György Gulyás, and Sándor Imre. User tracking on the web via cross-browser fingerprinting. In Peeter Laud, editor, *Information Security Technology for Applications - 16th Nordic Conference on Secure IT Systems, NordSec 2011, Tallinn, Estonia, October 26-28, 2011, Revised Selected Papers*, volume 7161 of *Lecture Notes in Computer Science*, pages 31–46. Springer, 2011. URL: http://dx.doi.org/10.1007/978-3-642-29615-4_4, doi:10.1007/978-3-642-29615-4_4.

[9] Joseph Bonneau, Cormac Herley, Paul C Van Oorschot, and Frank Stajano. The quest to replace passwords: A framework for comparative evaluation of web authentication schemes. In *Security and Privacy (SP), 2012 IEEE Symposium on*, pages 553–567. IEEE, 2012.

[10] Mohamed Amine Bouazzouni, Emmanuel Conchon, and Fabrice Peyrard. Trusted mobile computing: An overview of existing solutions. *Future Generation Computer Systems*, 2016.

[11] Girish Chiruvolu. User authentication: Beyond passwords. *Emerging Trends in Online Authentication, 2015 ISACA, North Texas Event*, 2015.

[12] Mauro Conti, Luigi V. Mancini, Riccardo Spolaor, and Nino Vincenzo Verde. Can't you hear me knocking: Identification of user actions on android apps via traffic analysis. In Jaehong Park and Anna Cinzia Squicciarini, editors, *Proceedings of the 5th ACM Conference on Data and Application Security and Privacy, CODASPY 2015, San Antonio, TX, USA, March 2-4, 2015*, pages 297–304. ACM, 2015. URL: http://doi.acm.org/10.1145/2699026.2699119, doi:10.1145/2699026.2699119.

[13] Tim Cooijmans, Joeri de Ruiter, and Erik Poll. Analysis of secure key storage solutions on android. In *Proceedings of the 4th ACM Workshop on Security and Privacy in Smartphones & Mobile Devices*, pages 11–20. ACM, 2014.

[14] Rob Coombs. Securing the future of authentication with arm trustzone-based trusted execution environment and fast identity online (fido). *ARM White Paper*, 2015.

[15] Shuaifu Dai, Alok Tongaonkar, Xiaoyin Wang, Antonio Nucci, and Dawn Song. Networkprofiler: Towards automatic fingerprinting of android apps. In *Proceedings of the IEEE INFOCOM 2013, Turin, Italy, April 14-19, 2013*, pages 809–817. IEEE, 2013. URL: http://dx.doi.org/10.1109/INFCOM.2013.6566868, doi:10.1109/INFCOM.2013.6566868.

[16] Ingo Deutschmann, Peder Nordstrom, and Linus Nilsson. Continuous authentication using behavioral biometrics. *IT Professional*, 15(4):12–15, 2013.

[17] Alexandra Dmitrienko, Christopher Liebchen, Christian Rossow, and Ahmad-Reza Sadeghi. Security analysis of mobile two-factor authentication schemes. *Intel Technology Journal*, 18(4), 2014.

[18] Paul Dunphy, Andreas P. Heiner, and N. Asokan. A closer look at recognition-based graphical passwords on mobile devices. In Lorrie Faith Cranor, editor, *Proceedings of the Sixth Symposium on Usable Privacy and Security, SOUPS 2010, Redmond, Washington, USA, July 14-16, 2010*, volume 485 of *ACM International Conference Proceeding Series*. ACM, 2010. URL: http://doi.acm.org/10.1145/1837110.1837114, doi:10.1145/1837110.1837114.

[19] Jan-Erik Ekberg, Kari Kostiainen, and N Asokan. Trusted execution environments on mobile devices. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 1497–1498. ACM, 2013.

[20] Richard P. Guidorizzi. Security: Active authentication. *IT Professional*, 15(4):4–7, 2013. URL: http://dx.doi.org/10.1109/MITP.2013.73, doi:10.1109/MITP.2013.73.

[21] Shraddha M Gurav, Leena S Gawade, Prathamey K Rane, and Nilesh R Khochare. Graphical password authentication: Cloud securing scheme. In *Electronic Systems, Signal Processing and Computing Technologies (ICESC), 2014 International Conference on*, pages 479–483. IEEE, 2014.

[22] N Harini, TR Padmanabhan, et al. 2cauth: A new two factor authentication scheme using qr-code. *International Journal of Engineering and Technology*, 5(2):1087–1094, 2013.

[23] Stephan Heuser, Marco Negro, Praveen Kumar Pendyala, and Ahmad-Reza Sadeghi. Droidauditor: Forensic analysis of application-layer privilege escalation attacks on android. In *Proceedings of the 20th International Conference on Financial Cryptography and Data Security*, 2016.

[24] Christian Holz and Frank R. Bentley. On-demand biometrics: Fast cross-device authentication. In Jofish Kaye, Allison Druin, Cliff Lampe, Dan Morris, and Juan Pablo Hourcade, editors, *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems, San Jose, CA, USA, May 7-12, 2016*, pages 3761–3766. ACM, 2016. URL: http://doi.acm.org/10.1145/2858036.2858139, doi:10.1145/2858036.2858139.

[25] Ronan Loftus and Marwin Baumann. Android 7 file based encryption and the attacks against it. 2017.

[26] Roland Rijswijk-Deij and Erik Poll. Using trusted execution environments in two-factor authentication: comparing approaches. 2013.

[27] Anil H Rokade, Zafar Ul Hasan, and Sonali A Mahajan. User authentication by secured graphical password implementation. *IJIRSE) International Journal of Innovative Research in Science & Engineering*, 2014.

[28] Mohamed Sabt and Jacques Traoré. Breaking into the keystore: A practical forgery attack against android keystore. In *European Symposium on Research in Computer Security*, pages 531–548. Springer, 2016.

[29] Santosh Kumar Sahu, Asis Kumar Dalai, and Sanjaya Kumar Jena. Novel scheme for user authentication. 2014.

[30] Nuno Santos, Himanshu Raj, Stefan Saroiu, and Alec Wolman. Using arm trustzone to build a trusted language runtime for mobile applications. In *ACM SIGARCH Computer Architecture News*, volume 42, pages 67–80. ACM, 2014.

[31] Sebastian Schrittwieser, Peter Frühwirt, Peter Kieseberg, Manuel Leithner, Martin Mulazzani, Markus Huber, and Edgar R Weippl. Guess who's texting you? evaluating the security of smartphone messaging applications. In *NDSS*, 2012.

[32] Tim Stöber, Mario Frank, Jens B. Schmitt, and Ivan Martinovic. Who do you sync you are?: smartphone fingerprinting via application behaviour. In Levente Buttyán, Ahmad-Reza Sadeghi, and Marco Gruteser, editors, *Sixth ACM Conference on Security and Privacy in Wireless and Mobile Networks, WISEC'13, Budapest, Hungary, April 17-19, 2013*, pages 7–12. ACM, 2013. URL: `http://doi.acm.org/10.1145/2462096.2462099`, `doi:10.1145/2462096.2462099`.

[33] Matthias Trojahn and Frank Ortmeier. Toward mobile authentication with keystroke dynamics on mobile phones and tablets. In Leonard Barolli, Fatos Xhafa, Makoto Takizawa, Tomoya Enokido, and Hui-Huang Hsu, editors, *27th International Conference on Advanced Information Networking and Applications Workshops, WAINA 2013, Barcelona, Spain, March 25-28, 2013*, pages 697–702. IEEE Computer Society, 2013. URL: `http://dx.doi.org/10.1109/WAINA.2013.36`, `doi:10.1109/WAINA.2013.36`.

[34] Xuetao Wei, Lorenzo Gomez, Iulian Neamtiu, and Michalis Faloutsos. Profiledroid: multi-layer profiling of android applications. In Özgür B. Akan, Eylem Ekici, Lili Qiu, and Alex C. Snoeren, editors, *The 18th Annual International Conference on Mobile Computing and Networking, Mobicom'12, Istanbul, Turkey, August 22-26, 2012*, pages 137–148. ACM, 2012. URL: `http://doi.acm.org/10.1145/2348543.2348563`, `doi:10.1145/2348543.2348563`.

[35] Jain-Shing Wu, Wan-Ching Lin, Chih-Ta Lin, and Te-En Wei. Smartphone continuous authentication based on keystroke and gesture profiling. In *Security Technology (ICCST), 2015 International Carnahan Conference on*, pages 191–197. IEEE, 2015.

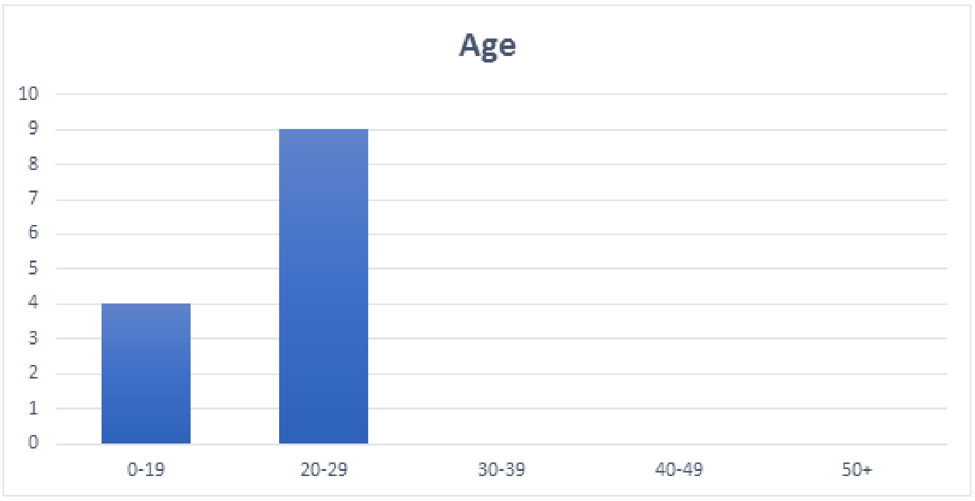
[36] Yaping Yang, Lizhi Cai, and Yanguo Zhang. Research on non-authorized privilege escalation detection of android applications. In *Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), 2016 17th IEEE/ACIS International Conference on*, pages 563–568. IEEE, 2016.

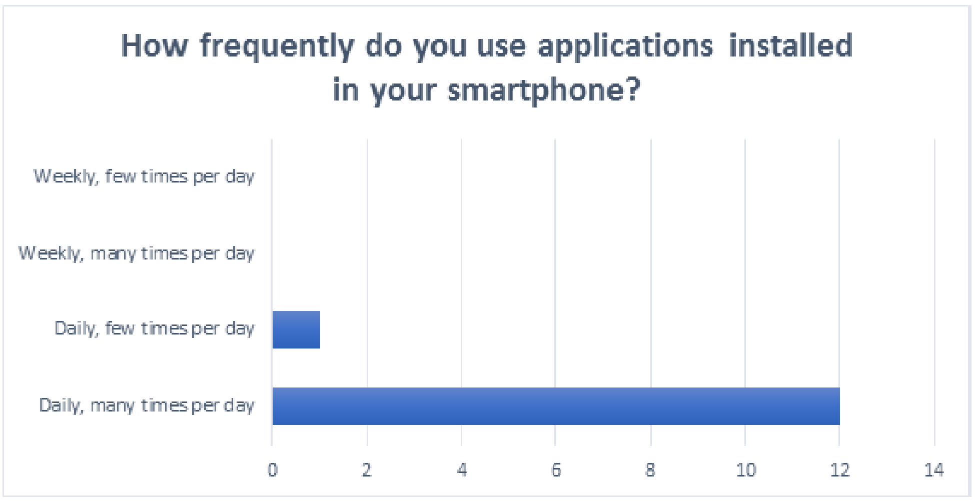
[37] Feng Zhang, Aron Kondoro, and Sead Muftic. Location-based authentication and authorization using smart phones. In Geyong Min, Yulei Wu, Lei (Chris) Liu, Xiaolong Jin, Stephen A. Jarvis, and Ahmed Yassin Al-Dubai, editors, *11th IEEE International Conference on Trust, Security and Privacy in Computing and Communications, TrustCom 2012, Liverpool, United Kingdom, June 25-27, 2012*,

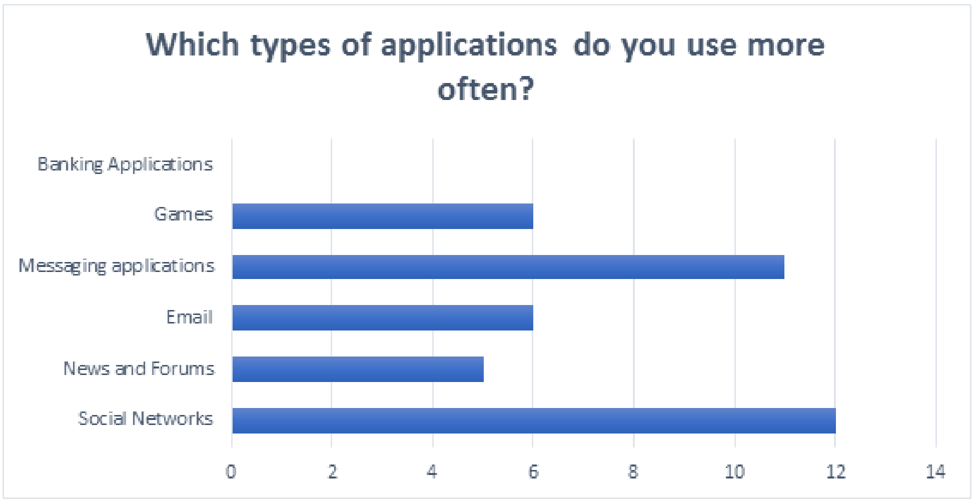
pages 1285–1292. IEEE Computer Society, 2012. URL: http://dx.doi.org/10.1109/TrustCom.2012.198, doi:10.1109/TrustCom.2012.198.

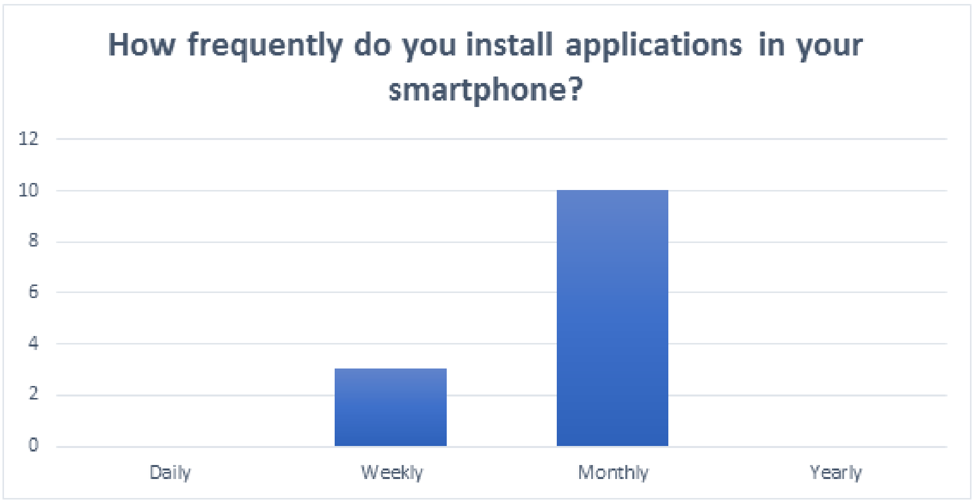
Appendix A

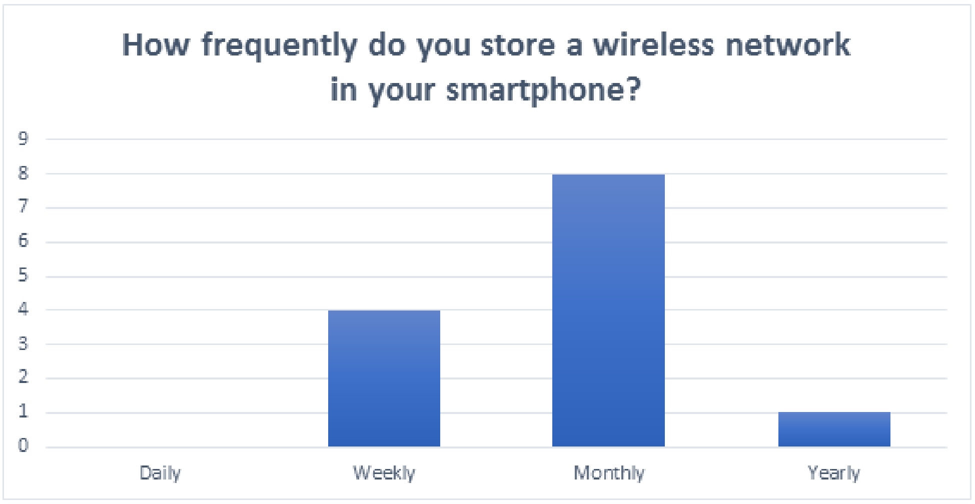
Preliminary Inquiry

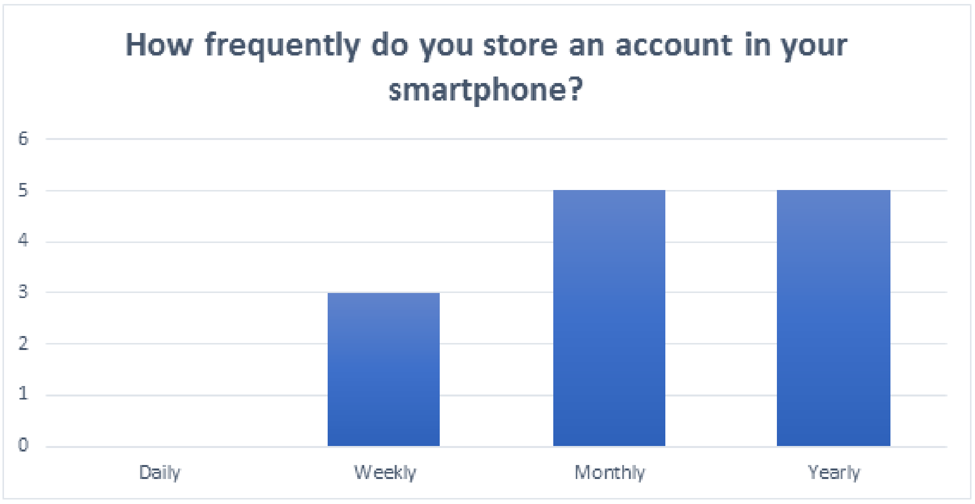




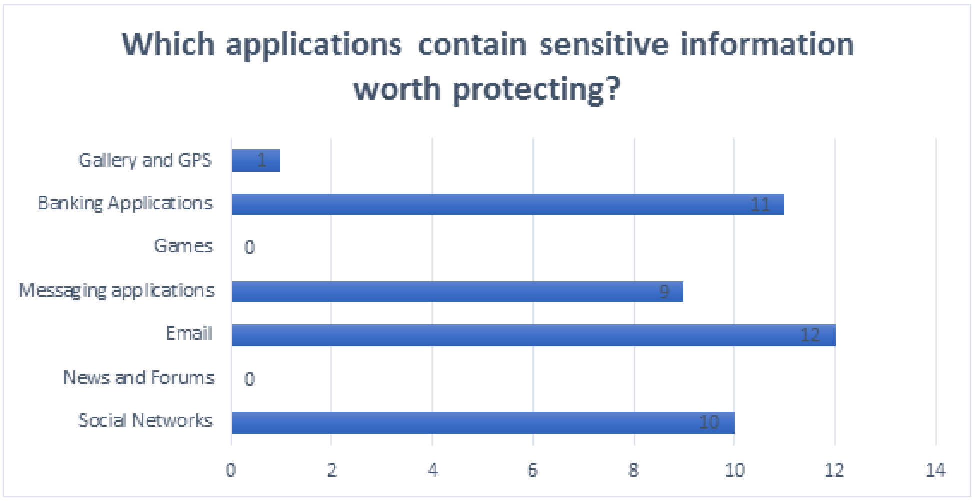

Which types of applications do you use more often?
Banking Applications
Games
Messaging applications
Email
News and Forums
Social Networks
0
2
4
6
8
10
12
14


How frequently do you install applications in your smartphone?
12
10
8
6
4
2
0
Daily
Weekly
Monthly
Yearly

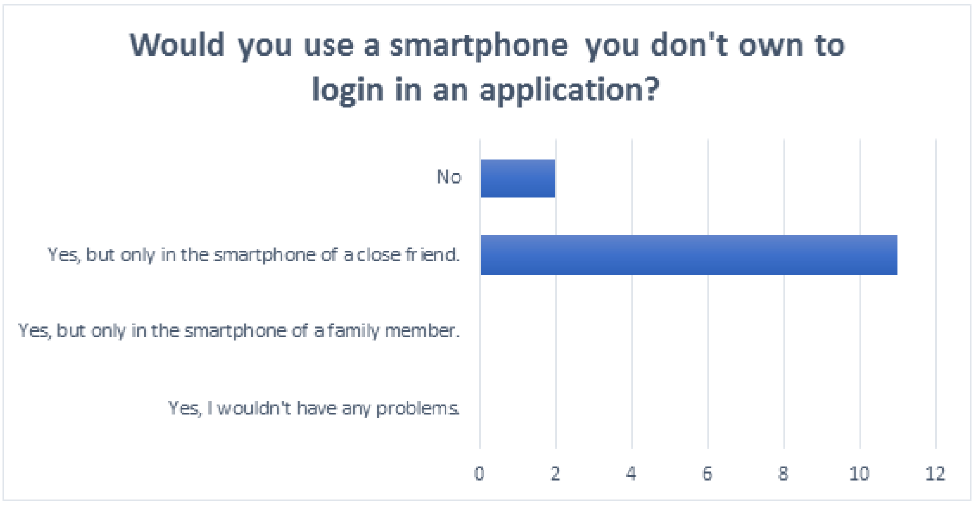

How frequently do you store a wireless network in your smartphone?
9
8
7
6
5
4
3
2
1
0
Daily
Weekly
Monthly
Yearly



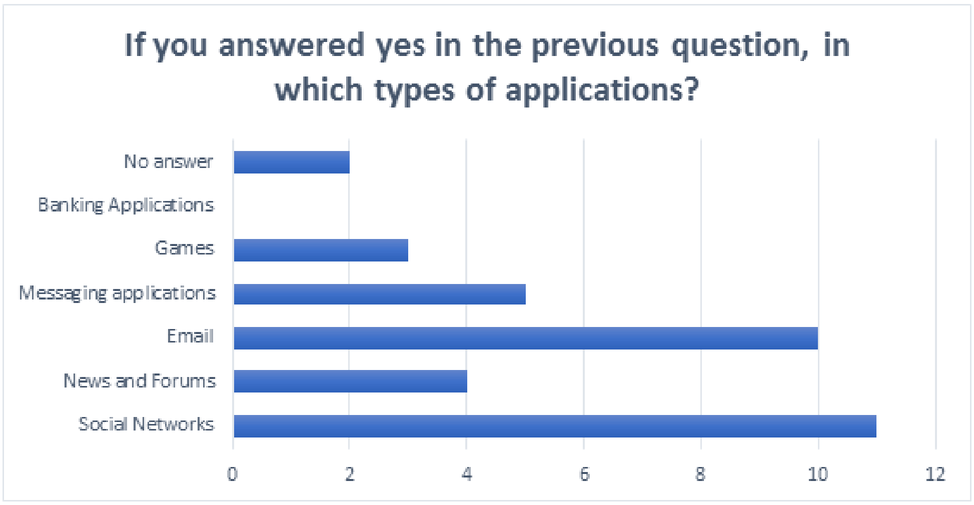
How frequently do you store an account in your smartphone?
6
5
4
3
2
1
0
Daily
Weekly
Monthly
Yearly

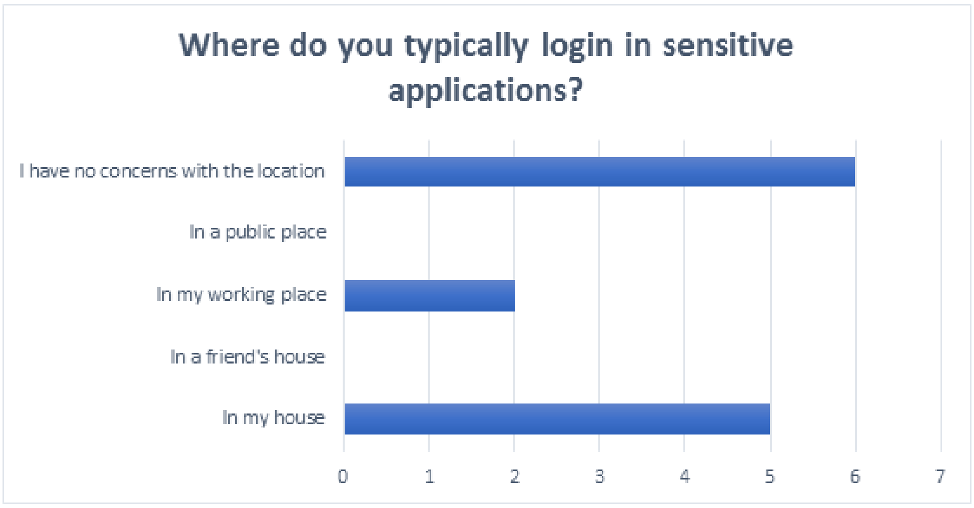


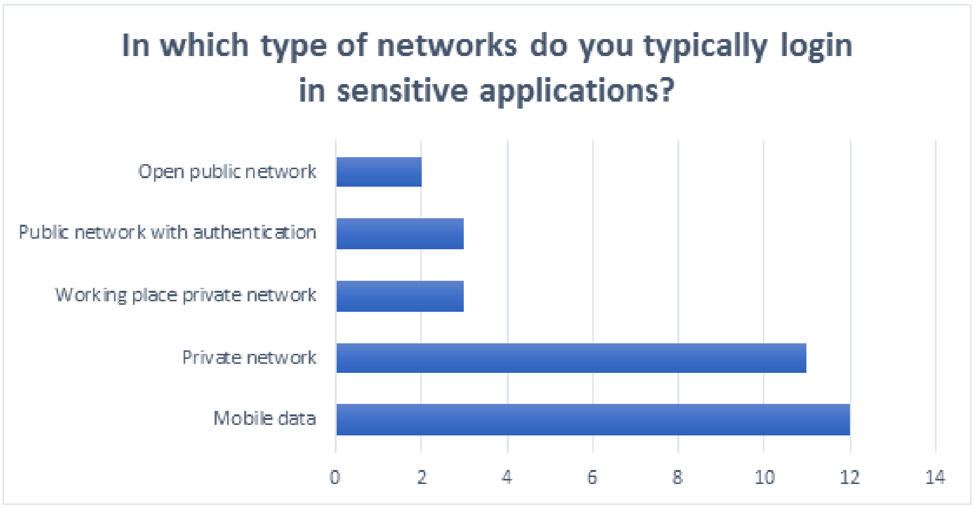
Which applications contain sensitive information worth protecting?
Gallery and GPS
1
Banking Applications
11
Games
0
Messaging applications
9
Email
12
News and Forums
0
Social Networks
10
0
2
4
6
8
10
12
14

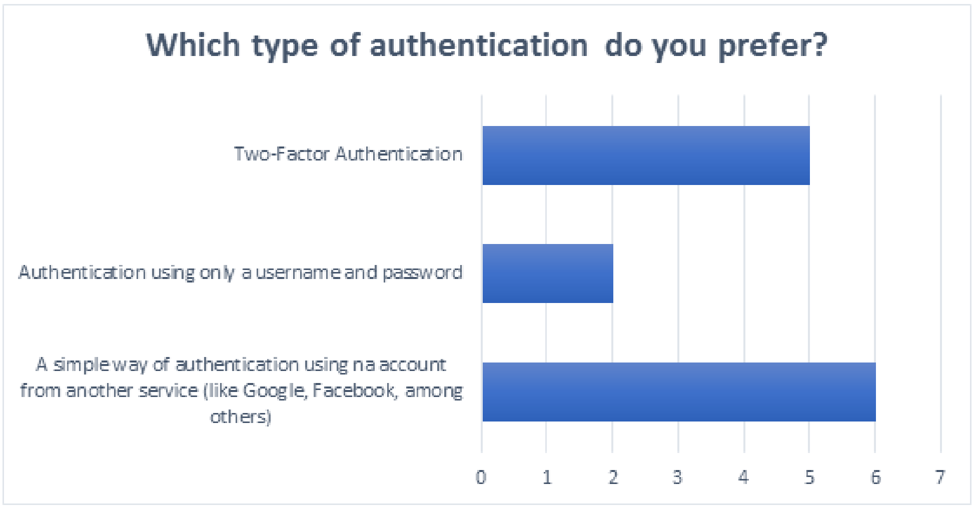


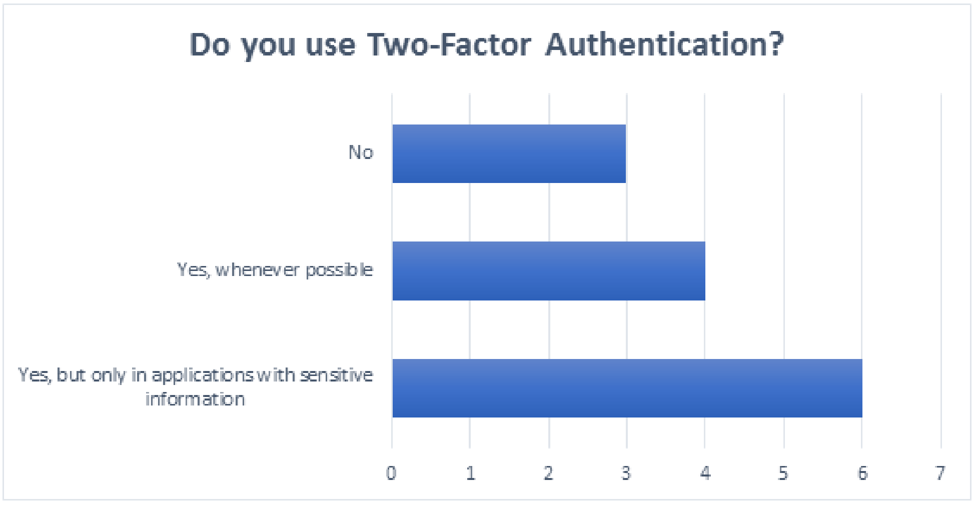
Would you use a smartphone you don't own to login in an application?
No
Yes, but only in the smartphone of a close friend.
Yes, but only in the smartphone of a family member.
Yes, I wouldn't have any problems.
0
2
4
6
8
10
12

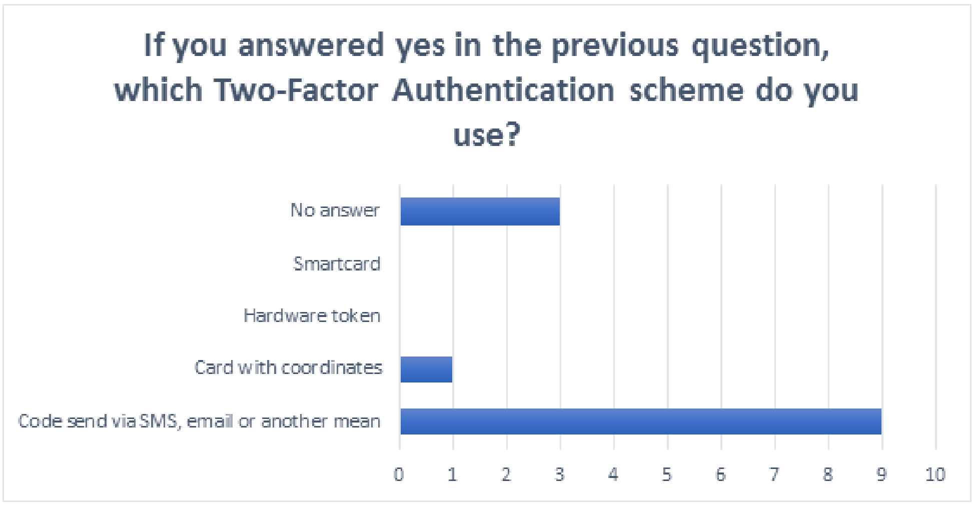

If you answered yes in the previous question, in which types of applications?
No answer
Banking Applications
Games
Messaging applications
Email
News and Forums
Social Networks
0
2
4
6
8
10
12

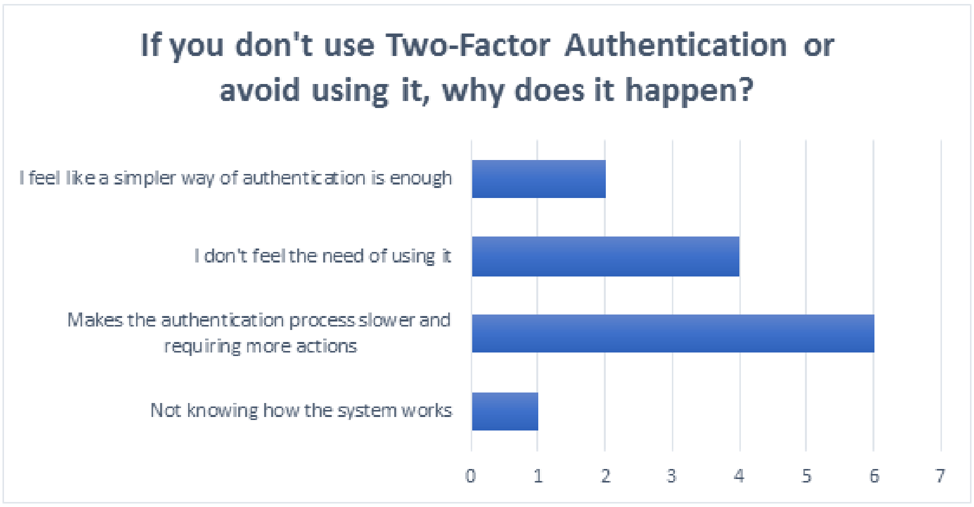

Where do you typically login in sensitive applications?
I have no concerns with the location
In a public place
In my working place
In a friend's house
In my house
0
1
2
3
4
5
6
7


In which type of networks do you typically login in sensitive applications?
Open public network
Public network with authentication
Working place private network
Private network
Mobile data
0
2
4
6
8
10
12
14

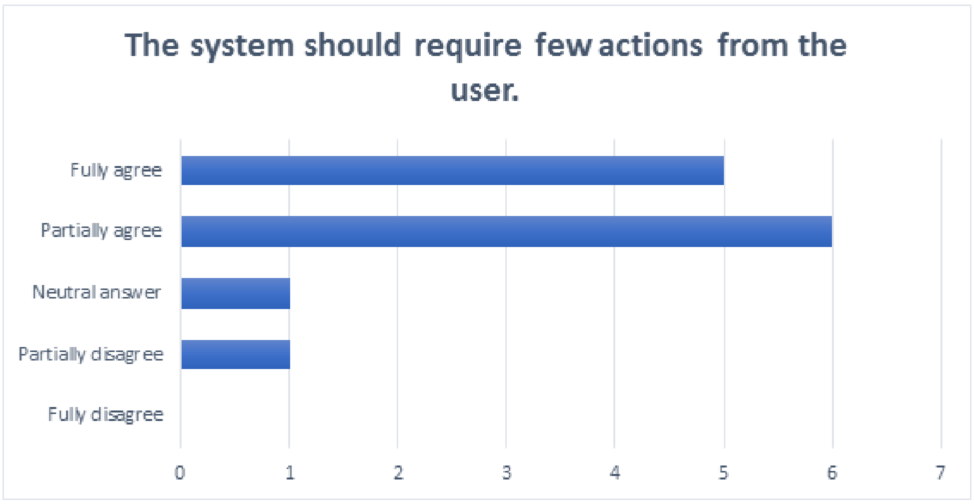

Which type of authentication do you prefer?
Two-Factor Authentication
Authentication using only a username and password
A simple way of authentication using na account from another service (like Google, Facebook, among others)
0
1
2
3
4
5
6
7

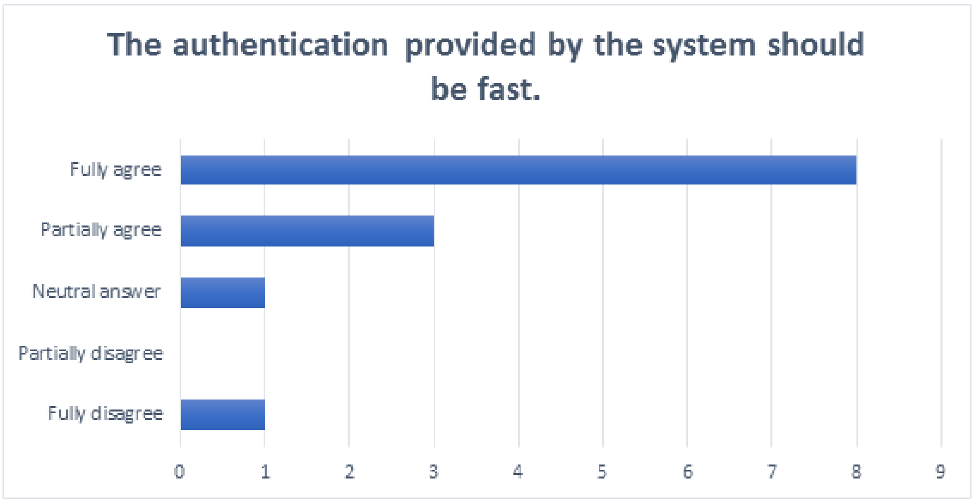

Do you use Two-Factor Authentication?
No
Yes, whenever possible
Yes, but only in applications with sensitive information
0
1
2
3
4
5
6
7

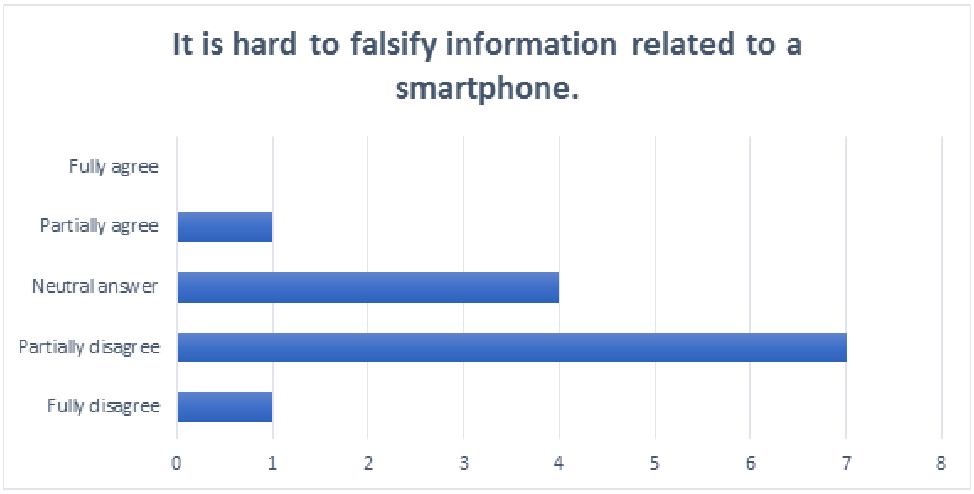

If you answered yes in the previous question, which Two-Factor Authentication scheme do you use?
No answer
Smartcard
Hardware token
Card with coordinates
Code send via SMS, email or another mean
0
1
2
3
4
5
6
7
8
9
10

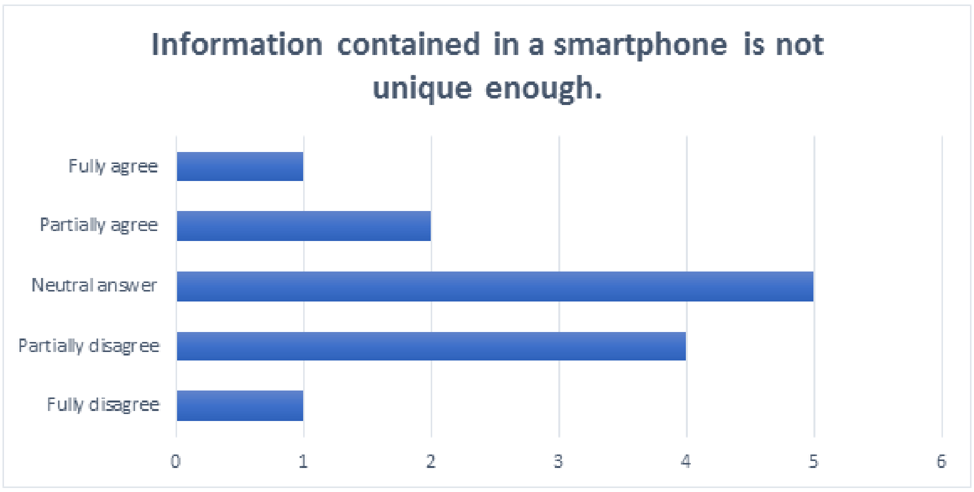


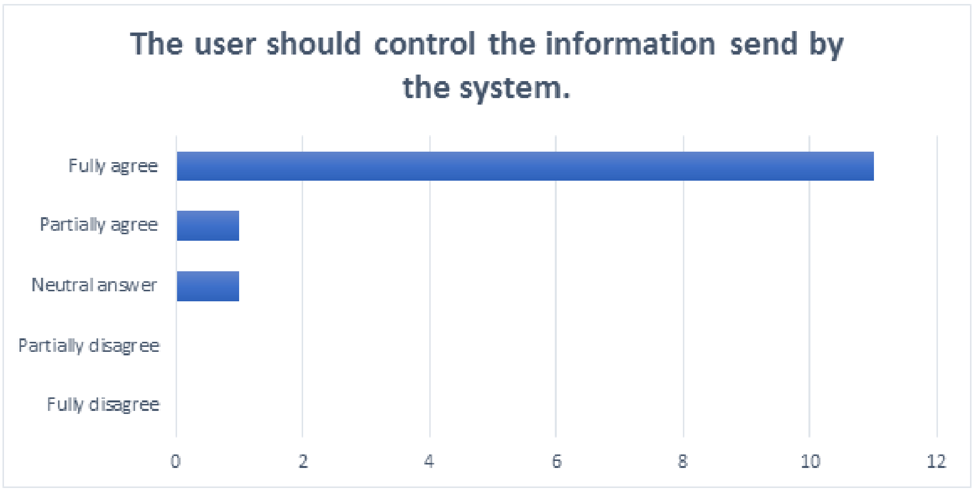
Consider an authentication system that uses information related to your smartphone and its configuration, like memorized accounts and networks, installed applications, SIM card information among others. This information will be retrieved securely and send encrypted in an automatically way to the authentication server. What do you think about the following affirmations?




The authentication provided by the system should be fast.
Fully agree
Partially agree
Neutral answer
Partially disagree
Fully disagree
0
1
2
3
4
5
6
7
8
9


It is hard to falsify information related to a smartphone.
Fully agree
Partially agree
Neutral answer
Partially disagree
Fully disagree
0
1
2
3
4
5
6
7
8


Information contained in a smartphone is not unique enough.
Fully agree
Partially agree
Neutral answer
Partially disagree
Fully disagree
0
1
2
3
4
5
6


The user should control the information send by the system.
Fully agree
Partially agree
Neutral answer
Partially disagree
Fully disagree
0
2
4
6
8
10
12