Diss. ETH No. XXX

# Smartphone Security: New Applications and Challenges

A thesis submitted to attain the degree of

Doctor of Sciences of ETH Zurich

presented by

Claudio Marforio

Master of Science in Computer Science,
ETH Zurich

born on 10.03.1986

citizen of Italy

accepted on the recommendation of

Prof. Dr. Srdjan Čapkun, examiner
Prof. Dr. N. Asokan, co-examiner
Prof. Dr. David Basin, co-examiner
Prof. Dr. Patrick Traynor, co-examiner

2015

*To succeed, planning alone is insufficient.*
*One must improvise as well.*
— Asimov, Foundation

# Abstract

Smartphones have become daily companions and are carried around by billions of people. They are used for the most diverse tasks and users trust them with both private and business data. As with any technology that reaches such a scale and holds so much information about its users, smartphones enable new applications but also pose new challenges. On the one hand, the power and versatility of these platforms can be used to enhance the security of our day to day activities. On the other hand, smartphones are becoming more and more the targets of attacks.

In this thesis we look at both aspects of smartphone security. We first propose two ways in which smartphones can enhance the security of daily life operations. Then, we look at sophisticated attacks that can circumvent the security mechanisms deployed on smartphones.

In the first part of this thesis, we introduce Sound-Proof, a two-factor authentication mechanism for the web. Sound-Proof uses short audio recordings to verify the proximity of the user's smartphone and computer. Only when the two are close to each other is the login attempt authorized. This solution is transparent to the user, who does not have to interact with his smartphone, thus matching the experience of password-only authentication. It is also easily deployable, as no extra software is required on the computer from where the user is logging in. We then propose a secure location-based two-factor authentication system for payments at points of sale. In this case we consider a strong adversary that can compromise the user's mobile operating system. To overcome this adversary, we propose novel secure enrollment schemes for the trusted execution environment of the user's smartphone.

In the second part of this thesis, we first investigate application phishing attacks. We look at various countermeasures and identify personalized security indicators as an attractive solution. We perform the first user study to assess their effectiveness in the new context of mobile application phishing. Our results show an increase in attack detection rates compared to previous studies on the web and give reason to believe that personalized security indicators can be used effectively on smartphones. We then look at application collusion attacks. In particular, we implement and evaluate various overt and covert communication channels that two applications can use to exchange information on smartphones. We provide evidence as to why some channels cannot be easily prevented and conclude that application collusion attacks remain an open problem.

# Sommario

Da qualche anno a questa parte, miliardi di persone portano sempre con sé e interagiscono con uno smartphone. Questi dispositivi hanno molteplici utilizzi, da quelli lavorativi a quelli più personali. Come spesso accade con tecnologie che raggiungono tali livelli di utilizzo e che gestiscono una grande quantità di informationi private, gli smartphone possono essere utilizzati per nuove operazioni ma pongono anche nuove sfide nell'ambito della sicurezza. Da una parte, le capacità di questi apparecchi fanno sì che possano essere utilizzati per migliorare la sicurezza delle nostre operazioni giornaliere, dall'altra essi vengono presi di mira dai criminali informatici con sempre maggior frequenza.

In questa tesi approfondiamo i seguenti aspetti della sicurezza degli smartphone: in primo luogo proponiamo due modalità con cui gli smartphone possono migliorare la sicurezza delle operazioni di ogni giorno; in seguito analizziamo alcuni sofisticati attacchi che aggirano le misure di sicurezza implementate su questi apparecchi.

Nella prima parte della tesi introduciamo Sound-Proof, un sistema di autenticazione a due fattori per il web. Sound-Proof usa due brevi registrazioni audio per verificare che il computer e lo smartphone dell'utente siano vicini tra di loro. Solo quando questi dispositivi sono vicini il sistema autorizza l'accesso. Questa soluzione risulta pratica per l'utente in quanto quest'ultimo non deve interagire con il proprio smartphone, ma semplicemente inserire l'identificativo e la password come d'abitudine; allo stesso tempo, non deve installare alcun software sul proprio computer. Proseguiamo questa prima parte della tesi introducendo un sistema di autenticazione a due fattori per i pagamenti con carta di credito presso i negozi, basato sulla posizione GPS dello smartphone. In questo contesto consideriamo un attaccante potente che sia in grado di compromettere il sistema operativo dello smartphone dell'utente. Per combattere questo tipo di attaccante proponiamo due nuovi sistemi di inizializzazione per la modalità sicura del dispositivo.

Nella seconda parte della tesi investighiamo le truffe di tipo "phishing" perpetuate contro applicazioni per smartphone. In particolare analizziamo diverse contromisure e identifichiamo le immagini personali di sicurezza come una soluzione accattivante al problema. Conduciamo il primo studio di usabilità atto a capire l'effettiva efficacia di questo sistema nel nuovo ambito degli smartphone. I risultati dimostrano un miglioramento nella rilevazione degli attacchi da parte degli utenti quando comparati con

sistemi simili utilizzati sul web. Per questo motivo le immagini personali di sicurezza possono essere utilizzate con successo sugli smartphone. In seguito analizziamo gli attacchi di applicazioni che colludono tra loro: in particolare, implementiamo e valutiamo diversi canali di comunicazione, sia palesi che occulti, che due applicazioni possono utilizzare per scambiarsi informazioni. Dimostriamo come alcuni di questi canali non possano essere facilmente bloccati e concludiamo che gli attacchi portati avanti da tali applicazioni rimangono un problema aperto.

# Zusammenfassung

Smartphones sind zum ständigen Begleiter von Milliarden von Menschen geworden. Sie werden für die verschiedensten Aufgaben eingesetzt und Benutzer vertrauen ihnen private wie auch geschäftliche Daten an. Wie bei jeder Technologie mit derartiger Verbreitung und derartigem Informationsgehalt, ergeben sich auch durch Smartphones neue Möglichkeiten aber auch neue Herausforderungen. Einerseits kann die Funktionalität und Vielseitigkeit dieser Plattformen genutzt werden um die Sicherheit unserer alltäglichen Aktivitäten zu erhöhen. Andererseits, werden Smartphones mehr und mehr zum Ziel von Angriffen.

In dieser Dissertation betrachten wir beide Sicherheitsaspekte von Smartphones. Zuerst schlagen wir zwei alltägliche Sicherheitsverbesserungen mit der Hilfe von Smartphones vor. Dann behandeln wir zwei komplizierte Angriffe, die die Sicherheitsmechanismen auf Smartphones umgehen können.

Im ersten Teil dieser Dissertation zeigen wir Sound-Proof, eine Zwei-Faktor-Authentifizierung für Webseiten. Sound-Proof nutzt kurze Audio-Mitschnitte, um die räumliche Nähe des Smartphones zum Computer des Benutzers zu verifizieren. Nur wenn beide Geräte nah beieinander sind, wird die Anmeldung autorisiert. Die Lösung ist für den Benutzer unsichtbar, da er sein Smartphone nicht aktiv benutzen muss und daher die gleiche Vorgehensweise, wie bei einem rein Passwort-gestützten Verfahren, hat. Die Lösung ist auch leicht einsetzbar, da keine zusätzliche Software auf dem verwendeten Computer installiert werden muss. Zudem schlagen wir eine sichere, ortsbasierte Zwei-Faktor-Authentifizierung für Zahlungen an Verkaufsstellen vor. In diesem Fall nehmen wir an, dass ein starker Angreifer das mobile Betriebssystem eines Nutzers kompromittieren kann. Um Sicherheit gegen einen solchen Angreifer zu gewährleisten, schlagen wir neue, sichere Registrierungsverfahren für die gesicherte Ausführungsumgebung des Benutzer-Smartphones vor.

Im zweiten Teil dieser Dissertation, analysieren wir zunächst Phishing Angriffe durch mobile Anwendungen. Wir betrachten verschiedene Gegenmassnahmen und stellen fest, dass personalisierte Sicherheitsindikatoren eine ansprechende Lösung sind. Wir führen die erste Benutzerstudie durch, in der die Wirksamkeit dieser Sicherheitsindikatoren, im neuen Kontext von Phishing Angriffe durch mobile Anwendungen, getestet wird. Unsere Ergebnisse zeigen einen Anstieg in erkannten Angriffen, verglichen mit vorherigen webbasierten Studien, und somit den möglichen Nutzen

von personalisierten Sicherheitsindikatoren auf Smartphones. Wir zeigen ausserdem, dass Kollusion zwischen mobilen Anwendungen möglich ist. Genauer entwickeln und bewerten wir verschiedene offene und verdeckte Kommunikationskanäle, die von zwei Anwendungen genutzt werden können, um auf Smartphones Informationen auszutauschen. Wir zeigen warum manche dieser Kanäle nicht einfach geschlossen werden können und folgern, dass Kollusion zwischen Anwendungen ein offenes Problem bleibt.

# Acknowledgments

Throughout the years that have passed while working on what now composes this thesis I have made friends, learned a lot and had great fun.

I would like to express my deepest gratitude to my advisor, Prof. Srdjan Čapkun, for his support and guidance during my studies. He provided me with motivation and a push to achieve more that helped my development as a researcher and as a person. A friendship is the best outcome.

I would like to thank my co-advisors, Prof. N. Asokan, Prof. David Basin and Prof. Patrick Traynor for being on my dissertation committee and dedicating time to reading and evaluating this thesis.

This thesis would not exist if not for my co-authors Dr. Ramya Jayaram Masti, Nikos Karapanos, Dr. Claudio Soriente, Dr. Kari Kostiainen, Hubert Ritzdorf, and Prof. Aurelién Francillon. The unending meetings and discussions enriched all of our publications.

Many thanks go to Dr. Ramya Jayaram Masti, Nikos Karapanos, Dr. Kari Kostiainen, Dr. Claudio Soriente, and Dr. Elizabeth Stobert for their help in improving this thesis.

The System Security Group is first a home and a family to all of its members. I would like to remember first my office mates with whom I shared so many days and laughs: Aanjhan Ranganathan, Dr. Ghassam Karame, Dr. Davide Zanetti, Dr. Elli Androulaki, Dr. Srdjan Marinovic, Dr. Claudio Soriente, and Nikos Karapanos. Even more laughs have been shared during lunches with everyone: Prof. Kasper Bonne Rasmussen, Dr. Boris Danev, Prof. Aurelién Francillon, Arthur Gervais, Dr. Ramya Jayaram Masti, Dr. Kari Kostiainen, Luka Malisa, Daniel Moser, Hildur Olafsdottir, Prof. Nils Ole Tippenhauer, Barbara Pfändner, Prof. Christina Pöpper, Dr. Joel Reardon, Hubert Ritzdorf, Dr. Reza Shokri, Dr. Elizabeth Stobert, and Der-Yeuan Yu.

Over the years I have made great friends which I am sure will remain so, scattered throughout the world, for a long time: Davide Zanetti, Ramya Jayaram Masti, Claudio Soriente, Nikos Karapanos, Srdjan Marinovic and Aanjhan Ranganathan. Titles are not needed this time.

I will forever be thankful to my girlfriend Alessia, the *real* doctor in the family, for all the love and support she has given me throughout these years. I extend my gratitude to my family in its entirety, but especially to my mother Donatella, to Mariano and to my father Rinaldo, who have taught me so much.

# Contents

# List of Figures

# List of Tables

# Chapter 1
# Introduction

Smartphones are mobile devices carried around by billions of people everyday and used for both personal and business activities, more often than not on the same device. It is common for people to use their smartphones for social-media interaction (e.g., Facebook, Twitter, Instagram), for their day-to-day private life (e.g., making and receiving calls and messages from relatives or friends, taking pictures, accessing their online banking), as well as for their business activities (e.g., receiving and composing e-mails, reading work-related documents, accessing their corporate functions through a VPN).

The introduction of smartphones followed years of technological advancements in both hardware and software components and, among other aspects, many security concepts introduced for other platforms were ported and refined for this new architecture. Examples are limiting access to peripherals through a well-defined set of APIs and permission controls, application sandboxing techniques, the permission-based security mechanism used by Android [127] and Windows Phones [194] and the signature-based third-party applications distribution used by iOS [10] and Windows Phones. Over multiple iterations, different vendors have borrowed security principles from one another and ended up having similar security guarantees for the applications running on the mobile OS as well as for the data stored by users.

Hardware developments have further enabled secure applications for modern smartphones. Technologies like TI M-Shield [25], ARM TrustZone [16], and GlobalPlatform standards for SIM applets deployment [121] have made their way into many devices available today. Hardware and software vendors, as well as the research community have tapped repeatedly into these technologies to propose and provide secure services like payment systems [218], keyless access to cars [48], and secure ticketing for events and public transport [249, 250].

As a two-factor authentication mechanism, smartphones can increase the security of day-to-day activities such as online banking and web logins. Nonetheless, they still suffer from a series of security problems and privacy concerns that have been raised and studied in the past years by both the research community and industry. In this thesis, we look at two different aspects of smartphone-related security: first we look at how current smartphones can be used as a second authentication factor in different scenarios

in a secure way; then we look at the security guarantees offered by current smartphones and how they can be both attacked and enhanced.

In the first part of this thesis we show how modern smartphones can strengthen the security of various services, such as web logins and payments at points of sale. In particular, smartphones can serve as a usable and secure two-factor authentication token, transparent to the user: something that might eventually make the adoption of two-factor authentication mechanisms widespread on the web. We then consider a strong attacker model and we present an ARM TrustZone-based solution to strengthen the security of credit-card payments at points of sale. Through the use of novel enrollment schemes for the secure code running on a user's phone, our solution withstands an adversary that can carry out targeted attacks and who has potentially infected the user's device at the time of enrollment.

While smartphones can aid in securing some aspects of people's daily lives, they are still open to some attacks. This holds, irrespective of which security mechanism smartphone operating systems use or if the underlying hardware supports system-wide security features, like ARM TrustZone. Furthermore, in many scenarios the added security benefits brought by smartphones rely on the alertness of the user.

In the second part of this thesis, we investigate security problems of widely used smartphone platforms and applications. In particular, we look at phishing attacks, commonly seen on the web, that have also appeared in the form of application phishing attacks against smartphone users. We borrow a solution proposed for the web, namely the use of personalized images at the time of login, and integrate it into smartphone applications. While this solution showed limited success on the web [171, 232] we study its effectiveness in this new setting. Finally, we show how decade-old attacks against multi-tier security systems, namely the use of covert channels to exfiltrate private information, still apply to current mobile platforms, and then propose some countermeasures.

## 1.1 Part I: Securing Applications Using Smartphones

We explore how smartphones can be used as a secure and usable two-factor authentication token for different application scenarios. In our work we focused on web authentication and payments at points of sale. We advance the state-of-the-art in terms of usability, deployability and security. While increasing security through the use of a second authentication factor, our solutions ensure that the user interaction is not changed or are evaluated

through a user study, and that the hardware and software requirements are met by platforms available and widespread today.

On the web, two-factor authentication protects online accounts even when passwords are leaked. Most users, however, prefer password-only authentication. One reason why two-factor authentication is so unpopular is the extra steps that the user must complete in order to log in [47, 141, 273]. Currently deployed two-factor authentication mechanisms require the user to interact with his phone or a dedicated two-factor token to, for example, copy a verification code to the browser. Two-factor authentication schemes that eliminate user-phone interaction exist, but require additional software to be deployed on every device from which the user wants to login.

With Sound-Proof, we propose a usable and deployable two-factor authentication mechanism as no interaction between the user and his phone is required. Furthermore, Sound-Proof is deployable with today's technologies and does not require any software to be installed on the computer from which the user is logging in. In Sound-Proof, the second authentication factor is the proximity of the user's phone to the device being used to log in. The proximity of the two devices is verified by comparing the ambient noise recorded by their microphones. Audio recording and comparison are transparent to the user, so that the user experience is similar to the one of password-only authentication. Sound-Proof can be easily deployed as it works with current phones and major browsers without plugins. We built a prototype for both Android and iOS. We provide empirical evidence that ambient noise is a robust discriminant to determine the proximity of two devices both indoors and outdoors, and even if the phone is in a pocket or a purse. We conducted a user study designed to compare the usability of Sound-Proof with Google 2-Step Verification. Participants ranked Sound-Proof as more usable and the majority would be willing to use Sound-Proof even for scenarios in which two-factor authentication is optional. Furthermore, users particularly liked how fast Sound-Proof is in comparison to a code-based two-factor authentication solution.

Following on our work with Sound-Proof and web authentication, we look at a different scenario, namely payments at points of sale. In this setting, we consider a stronger attacker model, one where the adversary can carry out targeted attacks and compromise the victims' smartphone applications and operating system.

We propose a novel location-based two-factor authentication solution for modern smartphones that withstands such attacks. We demonstrate our solution in the context of points of sale transactions and show how it can be effectively used for the detection of fraudulent transactions caused

by card theft or counterfeiting. Our scheme makes use of Trusted Execution Environments (TEEs), such as ARM TrustZone, commonly available on modern smartphones. Following the same goals as for Sound-Proof, namely security, usability and deployability, our solution does not require any changes in the user behavior at the point of sale or to the deployed payment terminals. In particular, we show that practical deployment of smartphone-based two-factor authentication requires a secure enrollment phase that binds the user to his smartphone TEE and allows convenient device migration. We then propose two novel enrollment schemes that resist targeted attacks and provide easy migration. We implement our solution with available platforms and show that it is indeed realizable, can be deployed with small software changes, and does not hinder user experience.

## 1.2 Part II: Smartphone Attacks and Countermeasures

In the second part of this thesis, we focus our attention on the security of smartphones themselves. First, we look at the user's role of the security equation and try to aid him in detecting an application phishing attack. Then, we show how applications that are inconspicuous when analyzed on their own, might actually turn out to be malicious and collude to exfiltrate personal data despite all the security mechanisms implemented on smartphones.

In mobile application phishing attacks, a malicious mobile application masquerades as a legitimate one to steal user credentials. Such attacks are an emerging threat with examples reported in the wild [100, 236, 291]. We first categorize application phishing attacks in mobile platforms and identify possible countermeasures. We show that personalized security indicators can help users to detect all types of phishing attacks. Indicators can be easily deployed, as no platform or infrastructure changes are needed. However, personalized security indicators rely on the user's alertness to detect phishing attacks. Previous work in the context of website phishing has shown that users tend to ignore the absence of security indicators and remain susceptible to attacks [171, 232]. Consequently, personalized security indicators have been deemed an ineffective phishing detection mechanism. We revisit their reputation and evaluate the effectiveness of personalized indicators as a phishing detection solution in the new context of mobile applications.

We report the results of a user study on the effectiveness of personalized indicators as a phishing-detection mechanism. Our results show that personalized indicators can help users detect phishing attacks in mobile applications and we suggest that their reputation as an ineffective anti-phishing mechanism should be reconsidered. We further discuss possible reasons that lead to our results and some limitations of our study.

Application collusion attacks are another mechanism that malicious applications can use to exfiltrate users' private data. In such attacks two malicious applications exchange information through the use of overt or covert channels. Users are not made aware of possible implications of application collusion attacks, quite the contrary, on existing platforms that implement permission-based security mechanisms. Rather, users are implicitly led to believe that by approving the installation of each application independently, they can limit the damage that an application can cause.

We implement and analyze a number of covert and overt communication channels that enable applications to collude and therefore indirectly escalate their permissions. We measure and report the throughput of the implemented communication channels, thereby highlighting the extent of the threat posed by such attacks. We show that while it is possible to detect a number of channels, most techniques do not detect all the possible communication channels and therefore fail to fully prevent application collusion. Our research shows that attacks using covert communication channels remain a real threat to smartphone security and an open problem for the research community.

## 1.3 Related Publications

The work presented in this thesis is based on the following publications, co-authored during my doctoral studies at ETH Zurich.

- C. Marforio, R. Jayaram Masti, C. Soriente, K. Kostiainen and S. Čapkun. Evaluation of Personalized Security Indicators as an Anti-Phishing Mechanism for Smartphone Applications. *Under Submission*
- N. Karapanos, C. Marforio, C. Soriente, and S. Čapkun. Sound-Proof: Usable Two-Factor Authentication Based on Ambient Sound. In *Proc. USENIX Security*, 2015
- C. Marforio, N. Karapanos, C. Soriente, K. Kostiainen, and S. Čapkun. Smartphones as Practical and Secure Location Verification Tokens for Payments. In *Proc. Network and Distributed System Security Symposium (NDSS)*, 2014

- C. Marforio, N. Karapanos, C. Soriente, K. Kostiainen, and S. Čapkun. Secure Enrollment and Practical Migration for Mobile Trusted Execution Environments. In *Proc. ACM workshop on Security and Privacy in Smartphones and Mobile devices (SPSM@CCS)*, 2013

- C. Marforio, H. Ritzdorf, A. Francillon, S. Čapkun. Analysis of the Communication between Colluding Applications on Modern Smartphones. In *Proc. Annual Computer Security Applications Conference (ACSAC)*, 2012

## 1.4 Thesis Outline

In this thesis, we investigate the security enhancements that the widespread use of smartphones can enable, as well as attacks and countermeasures for different security mechanisms employed on smartphones themselves. The first part of this thesis is dedicated to the former theme, investigating smartphones as two-factor authentication tokens in different scenarios. We focus on the security, deployability, and usability of the proposed solutions. In the second part of this thesis we look at application phishing attacks as well as personal data exfiltration through covert channels. A detailed outline of the thesis is as follows.

In **Chapter 2**, we introduce the architecture and operation of modern smartphones from the hardware and software perspective. We present an overview of the different hardware components, with a focus on the ones later used in our work. We then present the security mechanisms deployed on modern smartphones. For more detailed background information relevant to each part of the thesis, we refer the reader to the shorter background sections in each research chapter.

**Part I: Securing Applications Using Smartphones**

**Chapter 3** introduces the first part of this thesis, which focuses on applications whose security can be enhanced through the use of smartphones. In **Chapter 4**, we overview research as well as deployed systems that propose or employ two-factor authentication. We detail smartphone-based systems with a particular interest in their security, deployability and usability. In **Chapter 5**, we introduce Sound-Proof, a two-factor authentication mechanism for web login that leverages the ambient audio to verify the proximity of the first (e.g., the user's laptop) and the second (e.g., the user's phone) authentication factor. The driving factor of Sound-Proof is its immediate deployability and usability. In **Chapter 6**, we take a look at TEE solutions

available for smartphones and in particular ARM TrustZone. We present the problem of secure enrollment of applications running within the secure world, and possible solutions deployable with minimum changes to the system. We then introduce a secure two-factor authentication mechanism for payments at points of sale that is deployable and transparent to the user.

**Part II: Smartphones Attacks and Countermeasures**

**Chapter 7** introduces the second part of this thesis, which focuses on security on smartphones, attacks and countermeasures. In **Chapter 8**, we overview attacks on smartphone operating systems that enable an attacker to disrupt functionality or steal the user's data. We then provide a summary of proposed techniques to reduce or mitigate such attacks. In **Chapter 9**, we present a taxonomy of application phishing attacks, a technique borrowed from the web. We discuss possible solutions and focus on application indicators. In particular we are interested in finding out if mobile application indicators can better prevent phishing attacks than their web counterparts. Finally, we present a mechanism to securely set up application indicators despite the presence of an active attacker on the victim's phone. In **Chapter 10**, we show how overt and covert communication channels can be implemented on modern smartphones to enable application collusion attacks. We demonstrate several of these channels ranging from high-throughput but easy-to-detect, to low-throughput but harder-to-detect. We show how current countermeasures can be defeated, allowing an attacker to exfiltrate information in a slow yet effective manner.

In **Chapter 11** we present the closing remarks of the work presented in this thesis and conclude.

# Chapter 2
# Background on Smartphones

In this chapter we introduce the architecture of modern smartphones with regard to both their hardware and software, as well as the mechanisms used to deploy applications onto them. Due to the amount of sensitive private data stored on smartphones, specific security measures have been implemented by vendors. In particular, smartphone vendors have implemented security mechanisms aimed at preventing malicious software from running on their operating systems or controlling what potentially malicious applications can do. These mechanisms have so far limited widespread malware infections that, in contrast, have plagued personal computers for decades. Additional mechanisms provide a way for users to manage which applications have access to which components and services of their smartphones. This form of control allows careful users to protect their privacy by choosing which private information to share with whom and which applications have access to restricted system components (e.g., access to the microphone, or to the pictures taken with the camera).

We note that different hardware and software vendors introduce slight variations into the basic concepts described in this chapter. While keeping the descriptions generic we will also highlight some differentiation factors between smartphone vendors. We will try to give an as up-to-date as possible view of the different system components. In this fast-evolving market, hardware and software vendors keep on increasing the security mechanisms used in their smartphones and in the whole ecosystem around them. Finally, we will focus our attention only on Android and iOS smartphones, the most used platforms in the current market (accounting to approximately 97% of the global marketshare [152]). Other popular smartphone platforms provide similar security architectures and mechanisms (i.e., Microsoft WindowsPhone [194] and RIM's Blackberry [35]).

Overall we try to present the reader with details that are of interest to the rest of this thesis and will help in understanding the following research chapters. The rest of this chapter is structured as follows, we first present the typical hardware architecture of a modern smartphone in Section 2.1. We will see how we exploit some of the hardware components of smartphones in the first part of this thesis, where we use smartphones to secure our daily operations. We then introduce the different software components and highlight the security mechanisms provided by the operating system in Section 2.2. Finally, we provide an overview of the two main distribution

strategies for applications for Android and iOS devices in Section 2.3. In the second part of this thesis we will understand how attackers can overcome the security mechanisms implemented on current smartphones to carry out sophisticated attacks to steal users' credentials or private data.

## 2.1 Smartphone Hardware

A standard mobile device architecture (as shown in Figure 2.1) has two processors. The *application processor* runs the mobile OS (e.g., Android) and the applications on top of it. The vast majority of devices use ARM as the architecture for their processors. This is due, mainly, to ARM cores small footprint and reduced power consumption while offering powerful multi-core options. Modern ARM processors, from ARMv7 onwards, typically support both the ARM and the Thumb-2 instruction sets [17]. The original iPhone, in 2007, was one of the first smartphones to use an ARM processing core. Ever since, the majority of modern smartphones have used ARM cores as their main processing unit. The more recent ARM cores (since ARMv6) also support a system-wide security mechanism called ARM TrustZone. We defer the discussion on ARM TrustZone to Section 6.3 as it will be the focus of that research chapter.

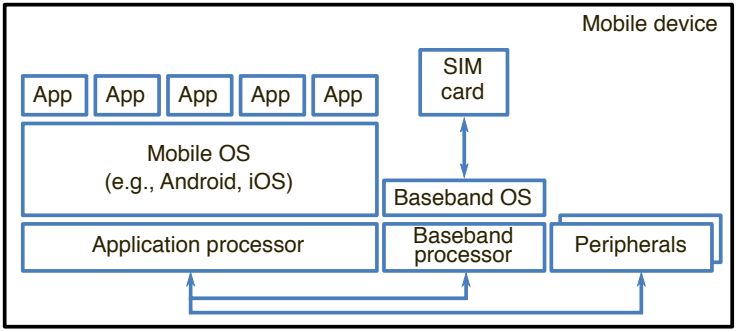



Figure 2.1: Architecture overview of a modern smartphone. The application processor is separated from the baseband processor that handles network operations and communicates with the SIM card. Applications run on top of a mobile operating system (e.g., Android or iOS).

The mobile OS that runs on the application processor has direct access to the peripherals found on the device and mediates this access to the unprivileged applications running on top. A common set of peripherals

found on modern smartphones consists of a wireless adapter (implementing both WiFi as well as Bluetooth functionality, such as the Broadcom BCM4354 [40]), a GPS receiver (such as the Broadcom BCM47521 [41]), one or more gyroscopes and accelerometers (such as the InvenSense's ICM-20608-G [154]), one or two integrated cameras, and a set of microphones and speakers. A physical keyboard and pointing device are typically omitted and a software-based implementation that shows on the screen is preferred. This plethora of peripherals is one feature that distinguishes smartphones from other mobile devices and personal computers and enables many different applications, some of which malicious, as well as some interesting security solutions.

A *baseband processor*, running the baseband OS, handles cellular communication and mediates communication between the application processor and the SIM card. Each SIM card has a unique identifier called IMSI (International Mobile Subscriber Identity), used to negotiate with a base station to grant access to the mobile network. The baseband OS is the responsible for implementing the protocols that govern the mobile networking space. For example, it must implement the stacks used for GSM [96], GPRS [94], EDGE [93], LTE [95]. The application processor and the baseband processor interact by exchanging messages, typically through a shared memory region. Device manufacturers are free to integrate any baseband OS of their choosing. Typically such operating systems are small microkernel-based real-time operating systems customized for baseband processors. Notable examples are: Nucleus RTOS [191], ThreadX [98] and OKL4 [113]. Due to the fact that baseband OSs are interfacing directly with the network providers and that they have full access to the smartphone hardware, they are typically well tested and undergo strict code audits to ensure that they are free of bugs. While some attacks against baseband processors have been found [70, 271], their numbers are relatively small compared to bugs found in more complex operating systems.

## 2.2 Smartphone Software

We now briefly describe the software architecture of Android and iOS operating systems. We then focus on their security features. We first introduce the features generically and then focus on more details for the Android OS, which is also used in the rest of this thesis as the main smartphone OS for discussion.

Smartphone operating systems are based on a monolithic kernel, such as the Linux kernel [125] (for Android) or a hybrid kernel such as Mach [12]

(for iOS). The kernel implements common functionality (such as process and memory management, filesystem access, drivers to access the peripherals). On top of the kernel, the OS features a layer of software that is both the foundation for developers to develop their applications (so called Software Development Kits, or SDKs) as well as a set of pre-installed privileged utilities to manage the system. Examples of these utilities are a network manager, a way to configure the many preferences of the system, a centralized notification center, and so on. The development framework dictates the running environment as well as the main development language of the operating system. Finally, a number of applications come bundled with the OS. Examples include an internet browser (on iOS an offspring of Safari, based on WebKit [14], and on Android a mobile version of Chrome, based on Blink [136]), an e-mail client, a calendar application, an address book application, and so on.

We now introduce how third party applications can be developed on both Android and iOS and then focus our attention on the security features provided by both architectures.

**Android Application Development**

Android applications are Dalvik executables [128], where Dalvik is a small implementation of Java specifically tailored for ARM processors and optimized for mobile platforms. Each application is developed in a type-safe language very similar to Java, and is run in a virtual machine (the so-called ART: Android runtime). Developers can also develop applications in C or C++ by using the Android NDK (Native Development Kit) and providing an interface to communicate data back and forth with the Dalvik application. Developing against the NDK allows for fast ARM-optimized code and also allows developers to use any C/C++ library, like opengl [133], or ProjectNe10 [22] used to access ARM Neon optimized routines. Figure 2.2 shows an overview of the main software components of an Android device.

Applications are developed using a set of Dalvik classes and XML files which define a plethora of parameters (e.g., string values, localization information, color combinations) as well as GUI descriptions. These values are either compiled in the application at compilation time or parsed at runtime by the OS to, for example, draw the GUI of the application on the screen. Developers can also draw GUI elements programmatically. Applications can have a number of components, and, most notably, can be split into *Activities*, the foreground processes that interact with the user, and *Services*, the background processes that can be used to perform long running operations or are active when an application goes into background.

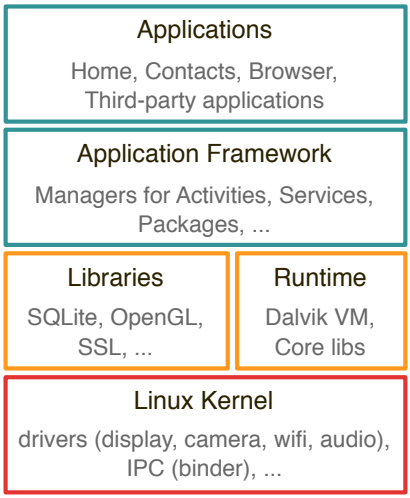



Figure 2.2: Android software components. The Linux kernel sits at the lowest level and manages drivers and other OS components. The runtime environment runs Dalvik executables, which can interface with system libraries. The application framework allows applications to use standard components such as activities and services. Finally, on top, applications are the front end to the user.

The system, indeed, allows background applications to perform any kind of task, like monitoring the GPS coordinates, record sound through the microphone and perform any network activity.

We refer the interested reader to the Android developer portal for a complete overview of Android system components, development practices and possible applications [125].

#### Apple iOS Application Development

iOS applications are compiled binaries implemented in Objective-C, an object-oriented dialect of C. Starting with iOS 7, applications can also be developed in Swift, a new open-source object-oriented language developed by Apple [13]. Through Objective-C glue code applications are able to directly use C/C++ libraries and have direct access to ARM functions through the direct use of assembly code. Developers make extensive use of the CocoaTouch runtime framework to interact with system components and the user interface. The latter can be implemented either programmatically

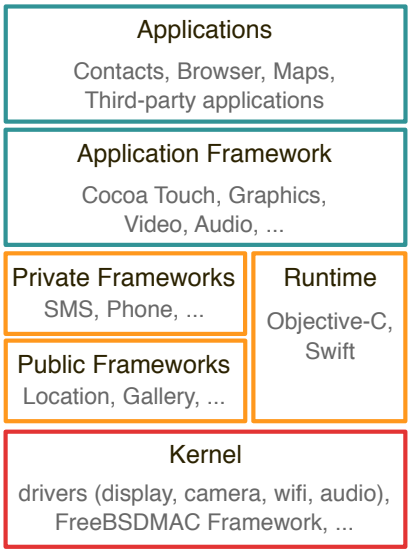



Figure 2.3: iOS software components. The kernel is at the lowest level and manages drivers, OS components and the security mechanisms. The runtime environment runs Objective-C and Swift applications. Applications can use the public libraries and the application framework components.

or through XML-based files. Figure 2.3 shows an overview of the main software components of an iOS device.

In contrast to Android applications, iOS applications are monolithic. In order to run longer-running tasks, each process can spawn multiple threads. The application lifecycle is also different from Android's in that, once in the background, applications are typically fully suspended or can continue to operate for a finite period of time (a number of seconds or minutes, at most), mostly in order to preserve battery life. A handful of exceptions to this rule are applications that require GPS updates (e.g., a mapping application or an activity logger), perform VoIP functionality (e.g., Skype) or play music (e.g., Spotify). With the recent release of iOS 9, Apple allows for two applications to run concurrently on some devices (e.g., the iPad Air 2, the iPad Pro and the iPad mini 4) and for both to display content on the screen. Apart from when they are in the foreground, applications can execute arbitrary code only upon receiving a *silent* push notification. When this happens the application has approximately 30 seconds to, for example, fetch data from a server.

We refer the interested reader to the iOS developer portal for a complete overview of iOS system components, development practices and possible applications [9].

### 2.2.1 Security Features

In terms of security, each OS features different mechanisms that we will now present in more detail.

**Secure Boot.** Modern smartphones employ, in most cases, a standard secure boot chain. Android manufacturers can develop their own version, with a potentially slightly modified strucutre. Some enable a modified (and unsigned) kernel to run right out of production (albeit typically voiding the phone warranty), and others lock the platform and require hacks, so-called *device rooting*, before an unsigned kernel can be booted. Apple's iOS devices, on the other hand, all follow the same procedure, which we now detail. Upon device boot, the application processor runs code from the Boot ROM (a read-only memory region burnt-in at hardware manufacturing time) which consitutes the hardware root of trust. Apart from the boot code, the Boot ROM also contains the Apple Root CA public key, which is used to verify the integrity of the Low-Level Bootloader (LLB). The LLB, when it has finished running its tasks, in turns verifies the integrity of the next bootloader (iBoot) which finally verifies and boots the iOS kernel. For any unsigned kernel to boot, the phone must be rooted. Exploits for each new kernel version must be discovered in order for the modified kernel to be booted up correctly. It is common that rooted devices modify the framework running on top of the kernel (or some kernel extensions), rather than the kernel itself.

Similar to the application processor, the baseband processor and the code running in the trusted execution environment (if any) follow a similar procedure to make sure that the code that runs at the lowest level on a device is verified and has not been modified.

**Application Sandboxing.** Each application running on top of the operating system and developed using the system SDK is sandboxed in its own execution environment. The sandbox makes sure that at runtime the application cannot access code or data used by another application (memory isolation). In Android, memory isolation is typically achieved by starting each application in its own virtual machine. The system then performs two levels of access control enforcement: *(i)* the middleware

component controls IPC calls and *(ii)* each application is assigned a locally unique Linux UID and the kernel enforces access to low-level resources based on these. In contrast, iOS performs memory isolation through the use of kernel-level process-based isolation to protect the address space of each application and of other system resources. Both platforms support address space layout randomization (ASLR), so that memory regions are randomized at launch, both for system processes and third party applications [10, 127]. Furthermore the use of ARM's Execute Never (XN), which marks memory pages as non-executable improves the overall memory protection. On iOS, only Apple-approved (and, to the best of our knowledge, only Apple-developed) applications can overcome this limitation to, for example, enable the JavaScript just-in-time compiler of the system browser.

**Storage Isolation.** Applications data integrity and confidentiality is protected when at rest. On Android, each application is assigned its own unique user identifier (Linux UID), as if it were a different user on the system. Storage isolation is then implemented as file-system permissions. Application files are created by default as owned by that particular user on the system and are, hence, accessible only by it. An application can also create world-readable and world-writable files that can then be accessed by any other application. On systems that provide an external storage medium (e.g., smartphones that allow the user to expand the internal storage with an SD-card), any data stored in the external storage can be accessed by any application. This is mainly due to the fact that external storage is typically formatted as FAT which does not support Unix access-control bits.

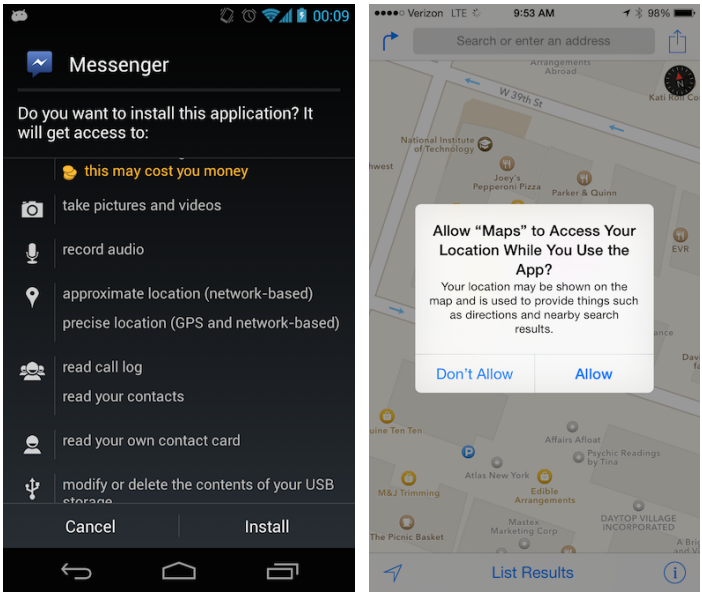
On both iOS and Android, third party applications can further store their data in encrypted form through the use of special APIs [10, 127] that make use of device-specific and application-specific keys. On iOS devices the Data Protection framework combines a device-specific key together with the user's passcode (i.e., either a 4-digit PIN or a longer alpha-numeric password) to generate per-application or per-file keys to keep stored data in an encrypted form. These operations happen transparent to the developer aided by the OS as well as the hardware. While an iOS application's data is stored in encrypted form automatically, on Android devices the developers decide what to store encrypted and how. Android provides a plethora of hardware-accelerated encryption routines available through the Java API.

**Permission-based Architecture.** Applications running on top of smartphone operating systems do not have direct access to peripherals or stored

user's data. Instead, all access is mediated by the kernel, the operating system and the upper-layer framework. When mediating access to peripherals (such as the microphone, or the camera) as well as to data (such as contacts or GPS location) the OS performs access control checks to make sure that the application accessing the private information has indeed been granted access to it (what is called a permission).

Android and iOS have a different approach to permissions. Android has roughly 138 permissions (at the time of this writing, for API level 21). For example, applications require permissions to access the internet, to read the contacts, to manipulate the pictures stored in the photogallery, or to receive location information. We refer the reader to the information available with Google for further details [127]. In contrast, permissions on iOS devices are very coarse-grained. At this time, the system requires explicit permissions to access: the microphone, the camera, photos, location information, contacts, calendars and reminders, the motion activity sensor (on iPhone 5s and later), social media accounts, HomeKit and HealthKit and Bluetooth sharing [10]. Finally, the user has to explicitly allow applications to receive push notifications.

Permissions are further handled differently by the two platforms both from the developer as well as from the user's perspective. On Android, the developer has to specify in the application's *manifest* file (which also specifies the application ID, name, and other details) all the permissions required by its application to run. The manifest is parsed on the marketplace to show to the user which permissions are required. Upon installation it is parsed by the Android OS to grant the correct set of permissions to the process. Up until Android 6.0, released in late 2015, users could only accept or deny (the latter resulting in the application not being installed) all the permissions required by a specific application at install time. In order to help users make sense of the potentially long list of permissions required by an application, the OS bundles fine-grained permissions into broader categories (as shown in Figure 2.4 (a)). This mechanism has changed recently and users can now revoke previously granted permissions to an application by going in the system settings. On iOS devices, the developer does not have to specify permissions while developing the application. At run-time, the operating system blocks the first access to any protected resource and prompts the user with a system dialog to deny or grant access to the resource (as shown in Figure 2.4 (b)). Again, the user is able to revoke a previously granted access to a resource by going into the system settings.




(a) (b)

Figure 2.4: The permission dialog for both Android version 4.3 (a) and iOS version 7 (b) shown to the user. On Android the dialogue appears at install time. If the user does not approve the required permissions the application is not installed. On iOS the user is prompted with a dialogue to allow the access to a specific resource at run-time.

**Application Signatures.** On both Android and iOS, applications are signed packages. On the Apple platform, only software signed by a valid Apple-issued certificate will be permitted to be installed and run. This mechanism extends the chain of trust to third-party applications and prevents unsigned code and self-modifying code from executing. Developers are required to sign their applications with an Apple-issued certificate that is released only after verification of the individual or organization requesting it. This allows Apple to bind each application with a particular identity, discouraging the creation and distribution of malicious code through their marketplace. All the signature checks are performed at runtime by the iOS kernel before starting an application. On Android, in contrast, applications can be signed with self-signed certificates that anyone can produce. In fact, the main reason behind signing applications on Android, is not to prevent

unsigned code from running on the platform (which is possible), but rather to maintain a same-origin principle for application updates. On Android, it is possible to ship an update to an application only if said update is signed by the same key used for the previous version.

**Trusted Execution Environments.** Smartphones have different Trusted Execution Environments (TEEs) available. In general, a TEE is any environment that is able to store secrets (e.g., private keys, passphrases) and run code in isolation from the main operating system. All smartphones have a SIM card, which can run small applets [120] (known, since before the advent of smartphones, as SIM applications). Such applets come pre-installed on the SIM card and have to be endorsed by the carrier operator in order to be deployed to customers' SIM cards.

Another option to perform operations in a trusted environment is ARM TrustZone, which is available on ARM cores since the Cortex-A series [16, 23]. We will go into more detail on how ARM TrustZone works in Chapter 6. As an overview, a TrustZone-enabled device supports two execution modes whose isolation is controlled and enforced in hardware. The *normal world* executes the main operating system (e.g., Android or iOS), while the *secure world* executes a smaller, typically more secure, operating system (e.g., an L4-variation on Apple's iPhones [10], or a custom Trustonic OS on some versions of the Samsung Galaxy family [258]). On previous phones and earlier smartphones, the TrustZone technology was more tied down by the device manufacturer and used for SIM locks and similar features [167]. Although the software running in the secure world potentially has access to all resources of the device (unlike software running on SIM cards), one of the main design goals is to keep this software small and verifiable in order to prevent bugs in this higher-privilege execution mode.

Although some proposals have been made in order to allow third-party developers to tap into the potential of smartphone TEEs [164, 165, 166], at the time of this writing the software in the secure world is mostly controlled by device manufacturers. Apple uses the secure world (the *secure enclave*, in Apple's terminology) to store encryption keys and to perform fingerprint matching from its Touch ID peripheral. Samsung proposed their KNOX platform [258] to enable businesses to store credentials and encryption keys in the secure world of TrustZone-enabled devices.

**Secure Peripherals.** Starting with the iPhone 5S and on some Android models (e.g., the Samsung Galaxy S5) hardware manufacturers have started

to embed secure peripherals. By secure peripherals we mean peripherals that are not accessible from applications directly. For example a fingerprint reader that both provides added security to the device as well as is securely integrated with the rest of the hardware. In this space, Apple's Touch ID stores the scanned image(s) into the encrypted memory of the secure enclave and the memory region is wiped as soon as the Touch ID sensor is deactivated. Only software running in the secure enclave is able to process the scanned copies of the fingerprint which are never accessible from the rest of the system. While fingerprint-based access (and, in general biometric-based access control) is prone to false positives and negatives and can be circumvented [53], the intention is to force users to enable passcodes on their phones, which in turns enables stronger secure storage. The general idea being that a larger population using a passcode (and more people using stronger passcodes, since they are required to type them in less frequently) is better than leaving devices unprotected for longer periods of time or missing a passcode altogether.

## 2.3 Distribution Markets

Smartphone vendors have adopted a controlled system to let users install third-party applications in the form of *marketplaces*. Apple's AppStore [11] and Google Play [131] are the most secure way (and in the case of Apple the only way) to install applications on a smartphone. This distribution model has some security advantages which we now outline in brief.

First of all, applications developed by third parties are submitted to the marketplaces and vetted before publication. Apple has a tight control model in which applications are subjected to static and dynamic analysis to check that they do not contain any potentially malicious code or that they use undocumented or private APIs. Then they are tested by a team of people to make sure that applications conform to visual guidelines but also to test for obvious bugs or problems created by each application. The whole testing procedure takes between one and two weeks in most cases, and terminates with the application being prepared for download by customers or being rejected with some motivation. The Android market follows a similar procedure, although applications tend to be published more quickly. Static and dynamic analysis (there is evidence that applications are tested in an emulator [124, 204]) is performed in the background and the application is pulled from the market should anything malicious be detected. Although these mechanisms do not fully prevent malware from making its way to a large number of customers, they are providing a first line of defense

against malware. In fact it is observed that the amount of malware present on smartphones is significantly smaller, compared to other (more open) systems [174, 257].

Second, the marketplace managers (i.e., Apple and Google, although smaller ones exist such as from Amazon [4]), that have all applications in a single repository can perform large-scale analysis to detect potentially malicious applications. This has, for example, led to the discovery of some malware masquerading as legitimate banking applications and trying to steal user credentials [100].

Another security advantage of the distribution model of smartphone applications is that it allows for continous and fast upgrades. This is true for third-party applications that can be updated (potentially fixing security bugs) and that will in turn be automatically downloaded and installed by the majority of the user base (this option is typically on by default but could be switched off, if desired by the user). Similarly, OS (and firmware) software updates are enabled over-the-air (OTA), something that again makes adoption of security fixes fast. For example, at the time of this writing, the iOS 8 (introduced in September 2014) adoption rate is 41% and iOS 9 (introduced in September 2015) is at 52%. On Android the numbers are lower, due to a more varied landscape in terms of devices: Android 4.4 (introduced in October 2013) is at 39%, Android 5.0 (introduced in November 2014) is at 21%.[1]

Finally, applications installed through marketplaces can be remotely removed from devices or disabled in case malicious activity is detected. Although some consider this activity a violation of user's privacy (the act of remotely disabling applications), it is also a strong security mechanism in case malicious software indeed finds its way through to users.

**Sideloading.** Although installation of applications through marketplaces is the recommended and typical way for users to install software on their smartphones, on Android it is also possible to install unsigned software from other places. This operation, known as sideloading, is potentially an attack vector.

On Apple devices it is not possible to install any third party application unless it is signed and comes from the AppStore. The only possibility to install and run unsigned content is to root or jailbreak the one's device. This operation disables the security checks performed by the operating

[1]The most up-to-date numbers can be found at `https://developer.apple.com/support/app-store/` for Apple devices and at `https://developer.android.com/about/dashboards/index.html` for Android devices.

system at runtime and hence, similar to Android, is a potential security risk for end users.

## 2.4 Summary

We have given a broad overview of smartphone platforms both in terms of hardware and software. In particular, after a generic introduction, we focused on the details that will be most useful to better appreciate the rest of this thesis. We will see how the hardware configurations of current smartphones enable solutions where they are used as an effective and usable two-factor authentication mechanism for daily operations. The second part of this thesis will focus on the software security mechanisms deployed today by device vendors and how they can be circumvented to steal users' private data.

# Part I

# Securing Applications Using Smartphones

# Chapter 3
# Introduction

The increasing availability of always-connected smartphone devices has led to their use for a multitude of applications. In this first part of the thesis, we focus on research and deployed mechanisms that use smartphones to enhance the security of other applications, such as unlocking cars, entering buildings and ticketing.

In the solutions that we present in the following chapters we always have similar design goals; security, usability and deployability. Security plays an important role both in two-factor authentication for the web and in payments at points of sale. On the web, large-scale leaks of database passwords [77, 158, 172] render two-factor authentication a strong mechanism to prevent attackers from gaining unrestricted access to users' online profiles and data. When it comes to payments at points of sale, targeted attacks against stolen credit cards information account for over 1 billion Euros of frauds in the Euro zone alone [97]. Similarly, a two-factor authentication mechanism can be used to prevent these fraudulent transactions.

The typical use-case for two-factor authentication on the web is to perform an extra verification of a user trying to log in to a web server. In general, the idea is that the user supplies his username and password (what he knows) and then inputs a one-time-code that proves the possession of an external device (the second factor). While for most sites this happens only upon login, services that require stronger security guarantees might ask the user to supply the information generated on the external device multiple times (e.g., when issuing a bank transfer). In Figure 3.1 we show a schematic view of a login transaction in both cases when two-factor authentication is absent and when it is present.

While security plays an important role, we also stress usability and deployability as further goals of our work. A plethora of two-factor authentication mechanisms have been proposed for web authentication but a recent study [215] has found that only 6.4% of users actually use two-factor authentication for their e-mail accounts. While the scenario is different where two-factor authentication is mandatory (e.g., for banking institutions), the situation still looks grim in scenarios where the second factor is optional (e.g., access to e-mails, social networks, coding websites, online games, and online shopping portals).

One of the problems with current two-factor authentication schemes for the web is that they require user interaction [24, 88, 129, 286], for example

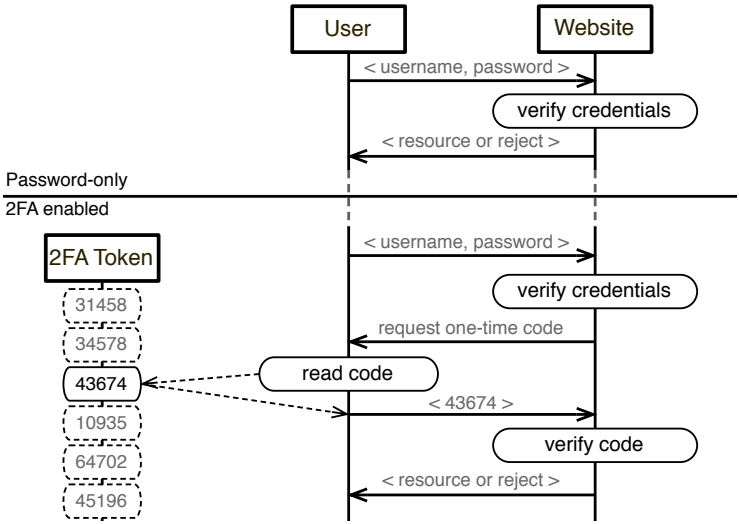



Figure 3.1: Overview of a web login operation in the absence (top) and presence (bottom) of a generic two-factor authentication mechanism.

to type into the browser a one-time-code received through SMS or from an application (as shown in Figure 3.1). Users have to find their smartphones, unlock them, open the application or wait for the SMS message to arrive, and then copy the code in order to log in. Proposed solutions that do not require user interaction leverage system resources that are not currently available to web browsers, such as bluetooth [63] or direct access to the wireless network card of the system [241].

In Chapter 5 we introduce Sound-Proof, a scheme that provides two-factor authentication for web logins by checking the proximity of the user's smartphone to the computer where he is logging in. We perform such a check by comparing audio simultaneously recorded by the smartphone's and the computer's microphones. If the two audio samples match, the user is successfully logged into the system. Sound-Proof is completely transparent to the user, making the login process similar to password-only authentication in terms of user experience. We evaluate Sound-Proof in different scenarios, both indoors and outdoors and show its applicability as well as its security against an attacker that is not co-located with the victim (as for most two-factor authentication schemes). In a user-study that compared Sound-Proof with Google 2 Step Verification, users ranked Sound-Proof considerably higher in terms of its usability. Similarly, participants

appreciated the reduced time that Sound-Proof imposed on the overall login procedure, compare to the code-based approach. Finally, as Sound-Proof requires access to only the microphone on both devices it is deployable on major browsers as well as on Android and iOS smartphones.

In contrast to web authentication, improving the security of payment systems at points of sale requires a solution that can withstand a more targeted attack carried our by an attacker who has a high incentive to perform a fraudulent transaction. However, deployability and usability are still of paramount importance. On the one hand, users are not willing to change their behavior when purchasing goods with credit cards (that is, sign or enter a PIN code) nor are they willing to spend more time than necessary to finalize the purchase. On the other hand, payment terminals have restricted hardware support and offer only restricted interactions with the customer. Upgrading payment terminals to support a new secure scheme would be a long and costly process, impacted by requirements for standardization, proofs of correct implementation, and deployment.

In Chapter 6 we propose a two-factor authentication solution for payments at points of sale that does not require the user to interact with his smartphone nor require extra software or hardware on the terminals. Our solution is secure against a strong attacker model that can mount targeted attacks and can compromise the user's phone remotely or fully compromise any phone to which he has physical access. We rely on the system-wide security offered by ARM TrustZone to perform a location verification when the user performs a payment at a point of sale. We propose two novel enrollment schemes for the application running in the secure world of the user's smartphone that guarantee security without relying on the trust-on-first-use assumption. Finally, we implement the proposed schemes on available hardware and show their deployability. Through a field study we verify that our proposed solution thwarts targeted attacks and further show how it can be applied to other scenarios that have similar requirements, such as entrance to buildings and public transport ticketing.

# Chapter 4
# Related Work

In this chapter we review related work in the area of two-factor authentication. We first look at traditional systems that look at providing a second authentication factor for the web. That is, systems where the second authentication factor is required for the user to login to a website. We then evaluate related work in the field of strengthening the security of payments, both online and at points of sale.

## 4.1 Two-Factor Authentication for the Web

**Hardware Tokens.** Hardware tokens range from the RSA SecurID [86] to recent dongles [286] that comply with the FIDO U2F [107] standard for universal 2FA. In the first case, the user is asked to input a fixed-length one-time code that is displayed on the RSA SecurID token (e.g., a 6 digit number) into the login website. The number displayed changes at fixed time intervals (typically 30 seconds). These tokens are usually tamper resistant and require a synchronization phase at manufacturing time where a secret is shared between the hardware token and the service provider.

The new generation of hardware tokens, such as Yubico [286], is easier to use but requires the token to be physically connected to the PC used to login (e.g., through a USB connection). Upon login, the user has to press a button (or touch a particular part of the token). This ensures that the user is in possession of the hardware token and is actually performing the login operation (the user action demonstrates the user intention to login).

Solutions based on hardware tokens require the user to carry and interact with the token and may be expensive to deploy because the service provider must ship one token per customer. In the case of standardized solutions (e.g., solutions that comply with the FIDO U2F [107] specification) only one token is required for any service provider. Anyway, in this case, the browser and the system through which the user is performing the login must support this new hardware peripheral.

In [38] the authors propose an additional factor to the main three ones. That is, they explore the usage of the social network of a user, someone he knows, as the fourth authentication factor. Overall they propose this fourth factor, in the form of vouching for a user, to be used in emergency situations where the other factors (i.e., passwords and hardware tokens) are not available.

**Software Tokens.** With the advent and widespread use of smartphones the solutions based on hardware tokens have been migrated to use applications that the user installs and configures on his personal mobile devices.

Google 2-Step Verification [129] is an example of a 2FA mechanism based on one-time codes, that uses software tokens on phones. The verification code is retrieved either from an application running on the phone or via SMS. Similar to the RSA SecurID, this mechanism requires the user to copy the verification code from the phone to the browser upon login. Similar solutions are also offered by other application vendors such as AgileBits 1Password [153] and Authy [24].

Researchers have also proposed to use the smartphone to provide cues to a user while he interacts with a graphical password on his computer [230]. In particular the smartphone would instruct the user on which parts of the graphical password he has to click in order to perform the login. This solution requires the user to interact with his smartphone every time he performs a login operation and changes the traditional password-only authentication system considerably.

In order to simplify what the user has to do upon login, similar to Yubico, Duo Push [79] and Encap Security [88] prompt the user with a push message on his smartphone with information on the current login attempt. The push message includes the username, the server being accessed, and the IP address of the client with approximate geolocation information. Simply tapping the notification will allow the user to login. In some countries, a similar approach is also used to confirm online transactions through the use of the Transakt application [91] when using a credit card for an online purchase.

Compared to hardware tokens, these solutions incur smaller costs to the service providers that do not have to manufacture and ship the hardware tokens to their users. The user is required to own a smartphone and install an application. Nonetheless, both solution spaces require the user to interact with his phone to authorize the login or the online transaction.

### 4.1.1 Reduced-Interaction 2FA

Standard solutions for code-based 2FA we just described are deployed by different online services (e.g., Google, Facebook, Github, Microsoft, Apple). For these vendors 2FA is an optional mechanism that users have to opt-in to use. Banking websites that allow online transactions are using 2FA mechanisms similar to the ones presented (through hardware tokens or SMS-based delivery of one-time codes). For such services 2FA is a mandatory requirement. Recent research has highlighted the very low

adoption rate that 2FA mechanisms suffer from when they are optional [47, 215]. One of the reasons that could negatively impact 2FA adoption is its poor usability. In particular users are not used to move away from a password-only authentication system and find use of 2FA cumbersome [141, 273]. We now look at proposed solutions that try to minimize the interaction between the user and the second authentication factor, making the added security mechanism more usable.

**Short-range Radio Communication.** PhoneAuth [63] is a 2FA proposal that leverages unpaired (unauthenticated RFCOMM) Bluetooth communication between the browser and the phone, in order to eliminate user-phone interaction. The choice to use unpaired bluetooth communication eliminates the requirement to pair the user's smartphone with all the devices from which he wishes to login to a particular website. The Bluetooth channel enables the server (through the browser) and the phone to engage in a challenge-response protocol which provides the second authentication factor.

The proposed solution requires the web browser to expose a Bluetooth API, something that is currently not available. A specification to expose a Bluetooth API in browsers has been proposed by the Web Bluetooth Community Group [270]. It is unclear whether the proposed API will support the unauthenticated RFCOMM or similar functionality which is required to enable seamless connectivity between the browser and the phone. However, if the Bluetooth connection is unauthenticated, an adversary equipped with a powerful antenna may connect to the victim's phone from afar [274] and login on behalf of the user, despite the second authentication factor being enabled. Prevention of range-extension attacks could be achieved via distance-bounding protocols [221]. However, today's phones and computers lack the hardware to run such protocols.

Authy [24], aside from the code-based solution presented before, can be configured to allow a seamless 2FA mechanism using Bluetooth communication between the computer and the phone. The idea is to have the smartphone close to the computer from where the login attempt is being made. Upon request of the second authentication factor an application installed on the user's computer queries the code from the Authy application on the user's smartphone and automatically fills the code into the browser input field. Authy does, therefore, require extra software to be installed on the user's computer or on any device from which the user wishes to login.

As an alternative to Bluetooth, the browser and the phone can communicate over WiFi [241]. This approach only works when both devices are

on the same network. The authors propose to use extra software on the computer to virtualize the wireless interface and create a software access point (AP) with which the phone needs to be associated. The user has to perform this setup procedure every time he uses a new computer to log in. Their solution also requires a phone application listening for incoming connections in the background, which is currently not possible on iOS.

Finally, the browser and the phone can communicate through near field communication (NFC). NFC hardware is not commonly found in commodity computers, and current browsers do not expose APIs to access NFC hardware. Similar to Bluetooth, the NFC communication range can be extended by an adversary equipped with a directional antenna [74, 146]. Furthermore, a solution based on NFC would not completely remove user-phone interaction because the user would still need to hold his phone close to the computer or where the NFC reader is placed.

In general the proposed short-range radio communication solutions try to limit the user interaction with the second authentication token. The idea is to increase the usability of 2FA mechanisms to foster larger user adoption. We have presented some of the pitfalls of these solutions such as range-extension attacks or the requirement of new software to be installed on all the devices from which the user performs login operations.

**Location Information.** When the user is performing a login operation, the server can check if the computer and the phone are co-located by comparing their GPS coordinates. GPS sensors are available on all modern phones but are rare on commodity computers. If the computer from which the user logs in has no GPS sensor, it can use the geolocation API exposed by some browsers [195]. The API allows to estimate the current location by querying Google Location Services with information about nearby wireless access points and the device's IP address. In indoor environments or where the GPS sensor on the phone does not have a fix, the phone may also use the geolocation API of the browser. Nevertheless, information retrieved via the geolocation API may not be accurate, for example when the device is behind a VPN or it is connected to a large managed network (such as enterprise or university networks). Similarly, if the phone does not use GPS coordinates and relies on its IP address to estimate its location, the location query is likely to return the coordinates of the data connection provider. Furthermore, geolocation information can be easily guessed by an adversary. For example, assume the adversary knows the location of the victim's workplace and uses that location as the second authentication

factor. This attack is likely to succeed during working hours since the victim is presumably at his workplace.

**Near-ultrasound.** SlickLogin [135] minimizes the user-phone interaction by transferring the verification code from the computer to the phone encoded using near-ultrasounds. The idea is to use spectrum frequencies that are non-audible for the majority of the population but that can be reproduced by the speakers of commodity computers (> 18kHz). Using non-audible frequencies accommodates for scenarios where users may not want their devices to make audible noise. Due to their size, the speakers of commodity computers can only produce highly directional near-ultrasound frequencies [228]. Near-ultrasound signals also attenuate faster, when compared to sounds in the lower part of the spectrum (< 18kHz) [15, 147]. With SlickLogin, the user must ensure that the speaker volume is at a sufficient level during login. Also, login will fail if a headset is plugged into the output jack of the computer from which the user is performing the login. Finally, this approach may not work in scenarios where there is in-band noise (e.g., when listening to music or in cafes) [147]. We also note that a solution based on near-ultrasounds may result unpleasant for young people and animals that are capable of hearing sounds above 18kHz [226].

**Other Sensors.** A 2FA mechanism can combine the readings of multiple sensors that measure ambient characteristics, such as temperature, concentration of gases in the atmosphere, humidity, and altitude, as proposed in [242]. These combined sensor modalities can be used to verify the proximity between the computer through which the user is trying to login and his phone. However, today's computers and phones lack the hardware sensors that are required for such an approach to work, making such a proposal not deployable.

## 4.2 Two-Factor Authentication for Payments

In the previous section we have summarized deployed system and research proposals that implement or improve two-factor authentication for the web. We now focus on the proposals and deployed systems that propose the use of 2FA for payments.

Payments at points of sale have received particular attention in terms of increasing their security and preventing frauds derived from credit card theft and cloning. Recently, MasterCard has proposed to use the location of the card-holder's smartphone to improve fraud detection in payments at points

of sale [109]. The proposal is presented at a high level and suggests to check the smartphone reported GPS location to the authorization server when a payment is processed. The server can compare the location reported by the phone and the known location of the point of sale where the transaction is taking place. Only if the two match (or are within a given range threshold) the transaction is authorized. In the patent there is no technical information on how to address potential smartphone OS compromise, user enrollment in the system or the migration of the user to a new smartphone.

The authors of [210] propose to mitigate fraud in payments at points of sale, using the phone location as an additional evidence to distinguish between legitimate and fraudulent transactions. In contrast to the MasterCard proposal, the bank sends a message to the user phone with the details of the transactions (including the location of the merchant and the one of the phone, as provided by the network operator) and asks the user to confirm the payment. This solution requires changes to the GSM infrastructure to provide the user with the current location of his phone, the points of sale to handle extra messages and additional cryptographic operations, and ultimately the overall user experience. Furthermore, the protocols in [210] do not account for a compromised mobile OS, nor they address enrollment. The authors do assume an adversary who can intercept any communication channel and require that the bank shares pair-wise keys with the phone, the mobile network operator, and the point of sale.

Securing online payments with two-factor authentication is the focus of a number of work that leverage hardware and software tokens to generate one-time passwords [32, 85, 213], biometric scanners [199, 200], or simply user-remembered passwords [266]. Financial institutions are deploying systems that leverage modern smartphones to replace traditional payment cards and using NFC-enabled point of sale terminals [132, 162]. Such solutions can only be used at selected stores or face large investments for hardware upgrades at merchants. Other systems that enhance the security of payment card transactions, require the user to confirm the payment through SMS messages or phone calls [261].

## 4.3 Complementary Research Topics

We conclude this chapter with related work in topics complementary to two-factor authentication.

### 4.3.1 Biometrics

While the second authentication factor proves to the service provider that the user is in possession of a second device, biometrics can be used to

strengthen the login procedure or completely replace it by focusing on something that the user itself has (as in, a physical property of the user) and that is unique.

Research as well as deployed systems have looked at using the users' fingerprints to authenticate a user to a system [223]. The most recent introduction in this field is the use of fingerprint sensors integrated within mobile devices. The overall idea is that upon registration, the user provides the fingerprint of one or multiple fingers. Upon login the user places his finger over a reader that scans his fingerprint and sends it to the server for comparison. While fingerprints are unique to each individual, recent work has shown how easy it is to recreate a person's fingerprint from an object that he touched [53, 54]. Similar attacks have been shown to work also on the latest solutions that try to evaluate the "liveliness" of the finger that is scanned, on top of performing the fingerprint scan.

Other deployed systems, for instance to grant access to buildings, range from iris scanners [99] to voice [201] and palm veins recognition solutions [111, 181]. More recent work in the area of biometric-based authentication has looked at how to authenticate a user by exploiting his unique body electrical transmission [222] or eye movements [81]. In all proposals, similar to fingerprints, a reference value for the physical property is stored on the server and matched against a new scan when the user attempts to access a protected resource. These new systems have not been shown to be easily attackable.

Finally, we mention recent work that authenticate users by performing continous verification of user's interaction patterns. In these work the user interaction with a device (e.g. a keyboard [240], a mouse [160], a touchscreen [110]) is profiled over time. Only if the interaction matches the user habitual one the user is allowed to remain logged into the system.

### 4.3.2 Security of Passwords

Most two-factor authentication mechanisms rely, as the first factor, on a user's password. The problem with password-only authentication systems is that their security comes from the strength of users passwords. Recent password databases leaks [77, 158, 172] have shown that users tend to choose weak passwords and also that users tend to re-use passwords across services [65].

Researchers have analyzed the problem extensively in recent years and have proposed a plethora of countermeasures. In general, proposals focused on client-side increased password strength [69, 143, 227, 285] and server-side password protection through new cryptographic tools [55, 161].

## 4.4 Summary

A large number of deployed systems as well as research proposals focused on two-factor authentication methods both for web logins and payments. In the first case, the user is required to provide a second authentication factor in order to access a protected online resource. In the second case, the user is required to perform additional operations or provide extra information to guarantee that the payment is indeed a legitimate one.

Both deployed systems and research proposals suffer from limited usability, in that they require the user to change his behavior when logging in or when performing a payment at a point of sale. Furthermore, some systems suffer from poor deployability as they require additional software on the user's computer or changes to the current infrastructure, something that is costly and requires a long time to be carried out.

In the following chapters we will try to overcome these limitations in both scenarios while enhancing the security of currently deployed systems.

# Chapter 5
# Sound-Proof: Usable Two-Factor Authentication Based on Ambient Sound

## 5.1 Introduction

Software tokens on modern phones are replacing dedicated hardware tokens in two-factor authentication (2FA) mechanisms. Using a software token, in place of a hardware one, improves deployability and usability of 2FA. For service providers, 2FA based on software tokens results in a substantial reduction of manufacturing and shipping costs. From the user's perspective, there is no extra hardware to carry around and phones can accommodate software tokens from multiple service providers.

Despite the improvements introduced by software tokens, most users still prefer password-only authentication for services where 2FA is not mandatory [47, 215]. This is probably due to the extra burden that 2FA causes to the user [141, 273], since it typically requires the user to interact with his phone.

Recent work [63, 241] improves the usability of 2FA by eliminating the user-phone interaction. However, those proposals are not yet deployable as their requirements are not met by today's phones, computers or browsers.

In this chapter, we focus on both the usability and deployability aspect of 2FA solutions. We propose Sound-Proof, a two-factor authentication mechanism that is transparent to the user and can be used with current phones and with major browsers without any plugin. In Sound-Proof the second authentication factor is the proximity of the user's phone to the computer being used to log in. When the user logs in, the two devices record the ambient noise via their microphones. The phone compares the two recordings, determines if the computer is located in the same environment, and ultimately decides whether the login attempt is legitimate or fraudulent.

Sound-Proof does not require the user to interact with his phone. The overall user experience is, therefore, close to password-only authentication. Sound-Proof works even if the phone is in the user's pocket or purse, and both indoors and outdoors. Sound-Proof can be easily deployed since it is compatible with current phones, computers and browsers. In particular, it works with any HTML5-compliant browser that implements the WebRTC

API [137], which is currently being standardized by the W3C [64]. Chrome, Firefox and Opera already support WebRTC, Internet Explorer plans to support it [192], and we anticipate that other browsers will adopt it soon.

Similar to other approaches that do not require user-phone interaction nor a secure channel between the phone and the computer (e.g., [63]), Sound-Proof is not designed to protect against targeted attacks where the attacker is co-located with the victim and has the victim's login credentials. Our design choice favors usability and deployability over security and we argue that this can edge for larger user adoption.

We have implemented a prototype of Sound-Proof for both Android and iOS. Sound-Proof adds, on average, less than 5 seconds to a password-only login operation. This time is substantially shorter than the time overhead of 2FA mechanisms based on verification codes (roughly 25 seconds [272]). We also report on a user study we conducted which shows that users prefer Sound-Proof over Google 2-Step Verification [129].

In summary, we make the following contributions:

- We propose Sound-Proof, a novel 2FA mechanism that does not require user-phone interaction and is easily deployable. The second authentication factor is the proximity of the user's phone to the computer from which he is logging in. Proximity of the two devices is verified by comparing the ambient audio recorded via their microphones. Recording and comparison are transparent to the user.
- We implement a prototype of our solution for both Android and iOS. We use the prototype to evaluate the effectiveness of Sound-Proof in a number of different settings. We show that Sound-Proof works even if the phone is in the user's pocket or purse and that it fares well both indoors and outdoors.
- We conducted a user study to compare the perceived usability of Sound-Proof and Google 2-Step Verification. Participants ranked the usability of Sound-Proof higher than the one of Google 2-Step Verification, with a statistically significant difference. More importantly, we found that most participants said that they would use Sound-Proof even if 2FA were optional.

## 5.2 Assumptions and Goals

**System Model.** We assume the general settings of browser-based web authentication. The user has a username and a password to authenticate to

a web server. The server implements a 2FA mechanism that uses software tokens on phones.

The user points his browser to the server's webpage and enters his username and password. The server verifies the validity of the password and challenges the user to prove possession of the second authentication factor.

**Threat Model.** We assume a remote adversary who has obtained the victim's username and password via phishing, leakage of a password database, or via other means. His goal is to authenticate to the server on behalf of the user. In particular, the adversary visits the server's webpage and enters the username and password of the victim. The attack is successful if the adversary convinces the server that he also holds the second authentication factor of the victim.

We further assume that the adversary cannot compromise the victim's phone. If the adversary gains control of the platform where the software token runs, then the security of any 2FA scheme reduces to the security of password-only authentication. Also, the adversary cannot compromise the victim's computer. The compromise of the computer allows the adversary to mount a Man-In-The-Browser attack [207] and hijack the victim's session with the server, therefore defeating any 2FA mechanism.

We do not address targeted attacks where the adversary is co-located with the victim. 2FA mechanisms that do not require the user to interact with his phone cannot protect against targeted, co-located attacks. For example, if 2FA uses unauthenticated short-range communication [63], a co-located attacker can connect to the victim's phone and prove possession of the second authentication factor to the server. We argue that targeted, co-located attacks are less common than non-selective, remote attacks. Furthermore, any 2FA mechanism may not warrant protection against powerful attackers. For example, if 2FA uses verification codes, a determined attacker may gain physical access to the phone or read the code from a distance [27, 28, 219].

We do not consider Man-In-The-Middle adversaries. Client authentication is not sufficient to defeat MITM attacks in the context of web applications [163]. We also do not address active phishing attacks where the attacker lures the user into visiting a phishing website and relays the stolen credentials to the legitimate website in real-time. Such attacks can be thwarted by having the phone detect the phishing domain [63, 212]. This requires short-range communication between the phone and the browser.

However, seamless short-range communication between the phone and the browser is currently not possible.

**Design Goals.** When designing a two-factor authentication solution for the web we had the following design goals.

- *Usability.* Users should authenticate using only their username and password as in password-only authentication. In particular, users should not be asked to interact with their phone — not even to pick up the phone or take it out of a pocket or purse.

- *Deployability.* The 2FA mechanism should work with common smartphones, computers and browsers. It should not require additional software on the computer or the installation of browser plugins. A plugin-based solution limits the usability of the system because (*i*) a different plugin may be required for each server, and (*ii*) the user must install the plugin every time he logs in from a computer for the first time. The mechanism should also work on a wide range of smartphones. We therefore discard the use of special hardware on the phone like NFC chips or biometric sensors.

## 5.3 Background on Sound Similarity

The problem of determining the similarity of two audio samples is close to the problem of audio fingerprinting and automatic media retrieval [52]. In media retrieval, a noisy recording is matched against a database of reference samples. This is done by extracting a set of relevant features from the noisy recording and comparing them against the features of the reference samples. The extracted features must be robust to, for example, background noise and attenuation. Bark Frequency Cepstrum Coefficients [142], wavelets [29] or peak frequencies [268] have been proposed as robust features for automatic media retrieval. Such techniques focus mostly on the frequency domain representation of the samples because they deal with time-misaligned samples. In our scenario, we compare two quasi-aligned samples (the offset is less than 150ms) and we therefore can also extract relevant information from their time domain representations.

In order to consider both time domain and frequency domain information of the recordings, we use one-third octave band filtering and cross-correlation.

**One-third Octave Bands.** Octave bands split the audible range of frequencies (roughly from 20Hz to 20kHz) in 11 non-overlapping bands where the ratio of the highest in-band frequency to the lowest in-band frequency is 2 to 1. Each octave is represented by its center frequency, where the center frequency of a particular octave is twice the center frequency of the previous octave. One-third octave bands split the first 10 octave bands in three and the last octave band in two, for a total of 32 bands. One-third octave bands are widely used in acoustics and their frequency ranges have been standardized [252]. The center frequency of the lowest band is 16Hz (covering from 14.1Hz to 17.8Hz) while the center frequency of the highest band is 20kHz (covering from 17780Hz to 22390Hz). In the following we denote with $B = [lb - hb]$ a set of contiguous one-third octave bands, from the band that has its central frequency at $lb$Hz, to the band that has its central frequency at $hb$Hz.

Splitting a signal in one-third octave bands provides high frequency resolution information of the original signal, while keeping its time-domain representation.

**Cross-correlation.** Cross-correlation is a standard measure of similarity between two time series. Let $x$, $y$ denote two signals represented as n-points discrete time series,[1] the cross-correlation $c_{x,y}(l)$ measures their similarity as a function of the lag $l \in [0, n-1]$ applied to $y$:

$$c_{x,y}(l) = \sum_{i=0}^{n-1} x(i) \cdot y(i-l)$$

where $y(i) = 0$ if $i < 0$ or $i > n-1$.

To accommodate for different amplitudes of the two signals, the cross correlation can be normalized as:

$$c'_{x,y}(l) = \frac{c_{x,y}(l)}{\sqrt{c_{x,x}(0) \cdot c_{y,y}(0)}}$$

where $c_{x,x}(l)$ is known as auto-correlation.

The normalization maps $c'_{x,y}(l)$ in $[-1, 1]$. A value of $c'_{x,y}(l) = 1$ indicates that at lag $l$, the two signals have the same shape even if their amplitudes may be different; a value of $c'_{x,y}(l) = -1$ indicates that the two signals have the same shape but opposite signs. Finally, a value of $c'_{x,y}(l) = 0$ shows that the two signals are uncorrelated.

[1] For simplicity we assume both series to have the same length.

If the actual lag between the two signals is unknown, we can discard the sign information and use the absolute value of the maximum cross-correlation $\hat{c}_{x,y}(l) = \max_l(|c'_{x,y}(l)|)$ as a metric of similarity ($0 \leq \hat{c}_{x,y}(l) \leq 1$). The computation overhead of $c_{x,y}(l)$ can be decreased by leveraging the cross-correlation theorem and computing $c_{x,y}(l) = F^{-1}(F(x)^* \cdot F(y))$, where $F()$ denotes the discrete Fourier transform and the asterisk denotes the complex conjugate.

## 5.4 Sound-Proof Architecture

The second authentication factor of Sound-Proof is the proximity of the user's phone to the computer being used to log in. The proximity of the two devices is determined by computing a similarity score between the ambient noise captured by their microphones. For privacy reasons we do not upload cleartext audio samples to the server. In our design, the computer encrypts its audio sample under the public key of the phone. The phone receives the encrypted sample, decrypts it, and computes the similarity score between the received sample and the one recorded locally. Finally, the phone tells the server whether the two devices are co-located or not. Note that the phone never uploads its recorded sample to the server. Communication between the computer and the phone goes through the server. We avoid short-range communication between the phone and the computer (e.g., via Bluetooth) because it requires changes to the browser or the installation of a plugin.

### 5.4.1 Similarity Score

Figure 5.1 shows a block diagram of the function that computes the similarity score. Each audio signal is input to a bank of pass-band filters to obtain $n$ signal components, one per each of the one-third octave bands that we take into account. Let $x_i$ be the signal component for the $i$-th one-third octave band of signal $x$. The similarity score is the average of the maximum cross-correlation over the pairs of signal components $x_i$, $y_i$:

$$S_{x,y} = \frac{1}{n} \sum_{i=1}^{i=n} \hat{c}_{x_i,y_i}(l)$$

where $l$ is bounded between 0 and $\ell_{max}$.

### 5.4.2 Enrollment and Login

Similar to other 2FA mechanisms based on software tokens, Sound-Proof requires the user to install an application on his phone and to bind the

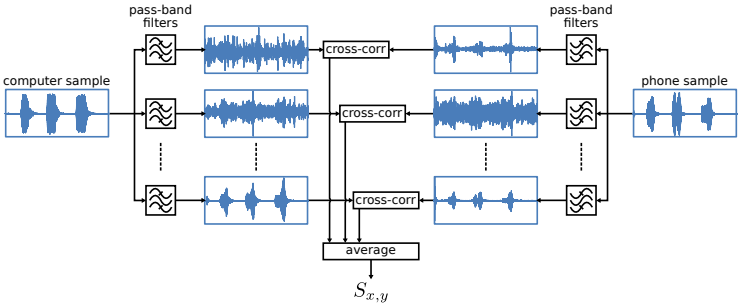



Figure 5.1: Block diagram of the function that computes the similarity score between two samples. The computation takes place on the phone. If $S_{x,y} > \tau_C$ and the average power of the samples is greater than $\tau_{dB}$, the phone judges the login attempt as legitimate.

application to his account on the server. This one-time operation can be carried out using existing techniques to enroll software tokens, e.g., [129]. We assume that, at the end of the phone enrollment procedure, the server receives the unique public key of the application on the user's phone and binds that public key to the account of that user.

Figure 5.2 shows an overview of the login procedure. The user points the browser to the URL of the server and enters his username and password. The server retrieves the public key of the user's phone and sends it to the browser. Both the browser and the phone start recording through their local microphones for $t$ seconds. During recording, the two devices synchronize their clocks with the server. When recording completes, each device adjusts the timestamp of its sample taking into account the clock difference with the server. The browser encrypts the audio sample under the phone's public key and sends it to the phone, using the server as a proxy. The phone decrypts the browser's sample and compares it against the one recorded locally. If the average power of both samples is above $\tau_{dB}$ and the similarity score is above $\tau_C$, the phone concludes that it is co-located with the computer from which the user is logging in and informs the server that the login is legitimate.

The procedure is completely transparent to the user if the environment is sufficiently noisy. In case the environment is quiet, Sound-Proof requires the user to generate some noise, for example by clearing his throat.

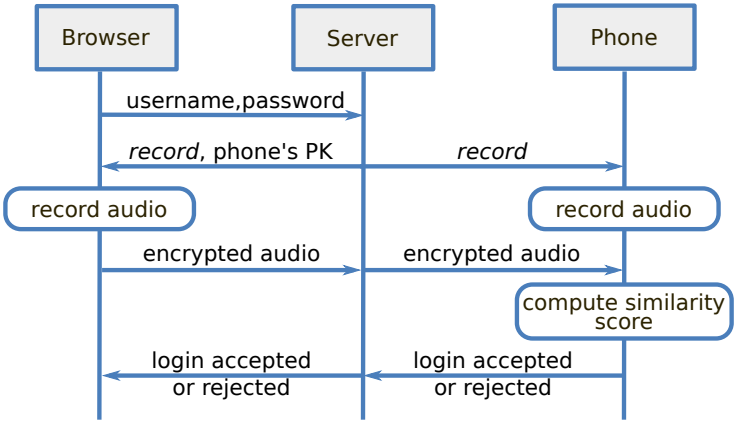



Figure 5.2: Sound-Proof authentication overview. At login, the phone and the computer record ambient noise with their microphones. The phone computes the similarity score between the two samples and returns the result to the server.

### 5.4.3 Security Analysis

**Remote Attacks.** The security of Sound-Proof stems from the attacker's inability to guess the sound in the victim's environment at the time of the attack.

Let $x$ be the sample recorded by the victim's phone and let $y$ be the sample submitted by the attacker. A successful impersonation attack requires the average power of both signals to be above $\tau_{dB}$, and each of the one-third octave band components of the two signals to be highly correlated. That is, the two samples must satisfy $Pwr(x) > \tau_{dB}$, $Pwr(y) > \tau_{dB}$ and $S_{x,y} > \tau_C$ with $l < \ell_{max}$.

We bound the lag $l$ between 0 and $\ell_{max}$ to increase the security of the scheme against an adversary that successfully guesses the noise in the victim's environment at the time of the attack. Even if the adversary correctly guesses the noise in the victim's environment and can submit a similar audio sample, the two samples must be synchronized with an error smaller than $\ell_{max}$. We also reject audio pairs where either sample has an average power below the threshold $\tau_{dB}$. This is in order to prevent an impersonation attack when the victim's environment is quiet (e.g., while the victim is sleeping).

Quantifying the entropy of ambient noise, and hence the likelihood of the adversary guessing the signal recorded by the victim's phone, is a challenging task. Results are dependent on the environment, the language spoken by the victim, his gender or age to cite a few. In Section 5.6 we provide empirical evidence that Sound-Proof can discriminate between legitimate and fraudulent logins, even if the adversary correctly guesses the type of environment where the victim is located.

**Co-located Attacks.** Sound-Proof cannot withstand attackers who are co-located with the victim. A co-located attacker can capture the ambient sound in the victim's environment and thus successfully authenticate to the server, assuming that he also knows the victim's password. Sound-Proof shares this limitation with other 2FA mechanisms that do not require the user to interact with his phone and do not assume a secure channel between the phone and the computer (e.g., [63]). Resistance to co-located attackers requires either a secure phone-to-computer channel (as in [24, 241]) or user-phone interaction (as in [79, 129]). However, both techniques impose a significant usability burden.

## 5.5 Prototype Implementation

Our implementation works with multiple browsers. We tested it with Google Chrome (version 38.0.2125.111), Mozilla Firefox (version 33.0.2) and Opera (version 25.0.1614.68). We anticipate the prototype to work with different versions of these browsers, as long as they implement the `navigator.getUserMedia()` API of WebRTC. We tested the phone application both on Android and on iOS. For Android, on a Samsung Galaxy S3, a Google Nexus 4 (both running Android version 4.4.4), a Sony Xperia Z3 Compact and a Motorola Nexus 6 (running Android version 5.0.2 and 5.1.1, respectively). We also tested different iPhone models (iPhone 4, 5 and 6) running iOS version 7.1.2 on the iPhone 4, and iOS version 8.1 on the newer models. The phone application should work on different phone models and with different OS versions without major modifications.

**Web Server and Browser.** The server component is implemented using the CherryPy [59] web framework and MySQL database. We use WebSocket [105] to push data from the server to the client. The client-side (browser) implementation is written entirely in HTML and JavaScript. Encryption of the audio recording uses AES256 with a fresh symmetric key; the symmetric key is encrypted under the public key of the phone using

RSA2048. We use the HTML5 WebRTC API [64, 137]. In particular, we use the `navigator.getUserMedia()` API to access the local microphone from within the browser. Our prototype does not require browser code modifications or plugins.

**Software Token.** We implement the software token as an Android application as well as an iOS application. The mobile application stays idle in the background and is automatically activated when a push notification arrives. Push messages for Android and iOS use the Google GCM (Google Cloud Messaging) APIs [130] and Apple's APN (Apple Push Notifications) APIs [8] (in particular the silent push notification feature), respectively. Phone to server communication is protected with TLS.

Most of the Android code is written in Java (Android SDK), while the component that processes the audio samples is written in C (Android NDK). In particular, we use the ARM Ne10 library, based on the ARM NEON engine [20] to optimize vector operations and FFT computations. The iOS application is written in Objective-C and uses Apple's vDSP package of the Accelerate framework [7], in order to leverage the ARM NEON technology for vector operations and FFT computations. On both mobile platforms we parallelize the computation of the similarity score across the available processor cores.

**Time Synchronization.** Sound-Proof requires the recordings from the phone and the computer to be synchronized. For this reason, the two devices run a simple time-synchronization protocol (based on the Network Time Protocol [197]) with the server. The protocol is implemented over HTTP and allows each device to compute the difference between the local clock and the one of the server. Each device runs the time-synchronization protocol with the server while it is recording via its microphone. When recording completes, each device adjusts the timestamp of its sample taking into account the clock difference with the server.

**Run-time Overhead.** We compute the run-time overhead of Sound-Proof when the phone is connected either through WiFi or through the cellular network. We run 1000 login attempts with a Google Nexus 4 for each connection type, and we measure the time from the moment the user submits his username and password to the time the web server logs the user in. On average it takes 4677ms (± 181ms) over WiFi and 4944ms (± 233ms) over Cellular to complete the 2FA verification. Table 5.1 shows the

| **Operations** | Mean (ms) | Std.Dev. |
|---|---|---|
| Recording | 3000 | — |
| Similarity score computation | 642 | 171 |
| Cryptographic operations | 118 | 15 |
| **Networking** | | |
| WiFi | 978 | 135 |
| Cellular | 1243 | 209 |

Table 5.1: Overhead of the Sound-Proof prototype. On average it takes 4677ms (± 181ms) over WiFi and 4944ms (± 233ms) over Cellular to complete the 2FA verification.

average time and the standard deviation of each operation. The recording time is set to 3 seconds. The similarity score is computed over the set of one-third octave bands $B = [50\text{Hz} - 4\text{kHz}]$. (Section 5.6.1 discusses the selection of the band set.) After running the time-synchronization protocol, the resulting clock difference was, on average, 42.47ms (± 30.35ms).

## 5.6 Evaluation

**Data Collection.** We used our prototype to collect a large number of audio pairs. We set up a server that supported Sound-Proof. Two subjects logged in using Google Chrome[2] over 4 weeks. At each login, the phone and the computer recorded audio through their microphones for 3 seconds. We stored the two audio samples for post-processing. Login attempts differed in the following settings.

- *Environment:* an office at our lab with either no ambient noise (labelled as Office) or with the computer playing music (Music); a living-room with the TV on (TV); a lecture hall while a faculty member was giving a lecture (Lecture); a train station (TrainStation); a cafe (Cafe).
- *User activity:* being silent, talking, coughing, or whistling.
- *Phone position:* on a table or a bench next to the user, in the trouser pocket, or in a purse.
- *Phone model:* Apple iPhone 5 or Google Nexus 4.

[2]We used Google Chrome since it is currently the most popular browser [247]. We have also tested Sound-Proof with other browsers and have experienced similar performance (see Section 5.8).

- *Computer model:* Mac Book Pro "Mid 2012" running OS X10.10 Yosemite or Dell E6510 running Windows 7.

At the end of the 4 weeks we had collected between 5 and 15 login attempts per each setting, totalling 2007 login attempts (4014 audio samples).

### 5.6.1 Analysis

We used the collected samples to find the configuration of system parameters (i.e., $\tau_{dB}$, $\ell_{max}$, $B$, and $\tau_C$) that led to the best results in terms of False Rejection Rate (FRR) and the False Acceptance Rate (FAR). A false rejection occurs when a legitimate login is rejected. A false acceptance occurs when a fraudulent login is accepted. A fraudulent login is accepted if the sample submitted by the attacker and the sample recorded by the victim's phone have a similarity score greater than $\tau_C$, and if both samples have an average power greater than $\tau_{dB}$.

To compute the FAR, we used the following strategy. For each phone sample collected by one of the subjects (acting as the victim), we use all the computer samples collected by the other subject as the attacker's samples. We then switch the roles of the two subjects and repeat the procedure. The total number of victim–adversary sample pairs we considered was 2,045,680.

**System Parameters.** We set the average power threshold $\tau_{dB}$ to 40dB which, based on our measurements, is a good threshold to reject silence or very quiet recordings like the sound of a fridge buzzing or the sound of a clock ticking. Out of 2007 login attempts we found 5 attempts to have an average power of either sample below 40dB and we discard them for the rest of the evaluation.

We set $\ell_{max}$ to 150ms because this was the highest clock difference experienced while testing our time-synchronization protocol (see Section 5.5).

An important parameter of Sound-Proof is the set $B$ of one-third octave bands to consider when computing the similarity score described in Section 5.4.1. The goal is to select a spectral region that (*i*) includes most common sounds and (*ii*) is robust to attenuation and directionality of audio signals. We discarded bands below 50Hz to remove very low-frequency noises. We also discarded bands above 8kHz, because these frequencies are attenuated by fabric and they are not suitable for scenarios where the phone is in a pocket or a purse. We tested all sets of one-third octave bands

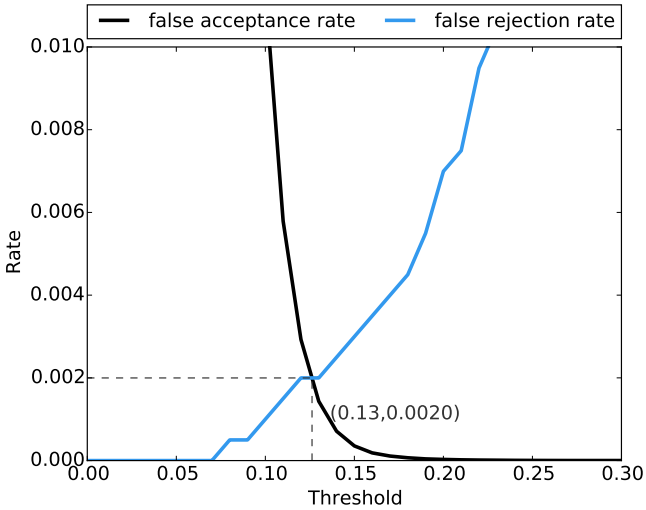



Figure 5.3: False Rejection Rate and False Acceptance Rate as a function of the threshold $\tau_C$ for $B = [50\text{Hz} - 4\text{kHz}]$. The Equal Error Rate is 0.0020 at $\tau_C = 0.13$.

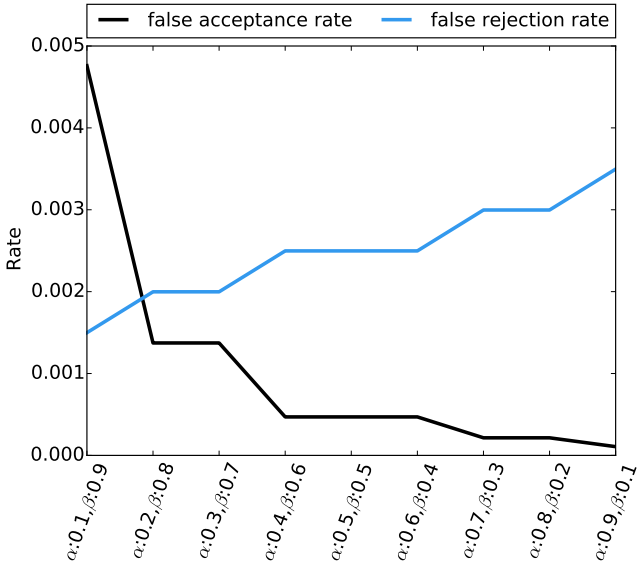
$B = [x - y]$ where $x$ ranged from 50Hz to 100Hz and $y$ ranged from 630Hz to 8kHz.

We found the smallest Equal Error Rate (EER, defined as the crossing point of FRR and FAR) when using $B = [50\text{Hz} - 4\text{kHz}]$. Figure 5.3 shows the FRR and FAR using this set of bands where the EER is 0.0020 at $\tau_C = 0.13$. We experienced worse results with one-third octave bands above 4kHz. This was likely due to the high directionality of the microphones found on commodity devices when recoding sounds at those frequencies [265]. Appendix A shows similar plots for all the band ranges $B$ we tested starting from 50Hz to 100Hz and going up from 630Hz to 4kHz.

We also computed the best set of one-third octave bands to use in case usability and security are weighted differently by the service provider.[3] In particular, we computed the sets of bands that minimized $f = \alpha \cdot FRR + \beta \cdot FAR$, for $\alpha \in [0.1, \ldots, 0.9]$ and $\beta = 1 - \alpha$. Figure 5.4(b) shows the set of bands that provided the best results for each configuration of $\alpha$ and $\beta$. As before, we experienced better results with bands below 4kHz. Figure 5.4(a)

[3] For example, a social network provider may value usability higher than security.

plots the FRR and FAR against the possible values of $\alpha$ and $\beta$. We stress that the set of bands may differ across two different points on the x-axis.




(a) False Rejection Rate and False Acceptance Rate when usability and security have different weights.

| | B | $\tau_c$ |
|---|---|---|
| $\alpha = 0.1,\ \beta = 0.9$ | [80Hz − 2500Hz] | 0.12 |
| $\alpha = 0.2,\ \beta = 0.8$ | [50Hz − 2500Hz] | 0.14 |
| $\alpha = 0.3,\ \beta = 0.7$ | [50Hz − 2500Hz] | 0.14 |
| $\alpha = 0.4,\ \beta = 0.6$ | [50Hz − 800Hz] | 0.19 |
| $\alpha = 0.5,\ \beta = 0.5$ | [50Hz − 800Hz] | 0.19 |
| $\alpha = 0.6,\ \beta = 0.4$ | [50Hz − 800Hz] | 0.19 |
| $\alpha = 0.7,\ \beta = 0.3$ | [50Hz − 1000Hz] | 0.2 |
| $\alpha = 0.8,\ \beta = 0.2$ | [50Hz − 1000Hz] | 0.2 |
| $\alpha = 0.9,\ \beta = 0.1$ | [50Hz − 1250Hz] | 0.21 |

(b) One-third octave bands and similarity score threshold.

Figure 5.4: Minimizing $f = \alpha \cdot FRR + \beta \cdot FAR$, for $\alpha \in [0.1, \ldots, 0.9]$ and $\beta = 1 - \alpha$.

Experiments in the remaining of this section were run with the configuration of the parameters that minimized the EER to 0.0020: $\tau_{dB} =$ 40dB, $\ell_{max} = 150$ms, $B = [50\text{Hz} - 4\text{kHz}]$, and $\tau_C = 0.13$.

### 5.6.2 False Rejection Rate

In the following we evaluate the impact of each setting that we consider (environment, user activity, phone position, phone model, and computer model) on the FRR. Figures 5.5 and 5.6 show a box and whisker plot for each setting. The whiskers mark the 5th and the 95th percentiles of the similarity scores. The boxes show the 25th and 75th percentiles. The line and the solid square within each box mark the median and the average, respectively. A gray line marks the similarity score threshold ($\tau_C = 0.13$) and each red dot in the plots denotes a login attempt where the similarity score was below that threshold (i.e., a false rejection).

**Environment.** Figure 5.5 shows the similarity scores for each environment. Sound-Proof fares equally well indoors and outdoors. We did not experience rejections of legitimate logins for the Music (over 432 logins), the Lecture (over 122 logins), and the TV (over 430 logins) environments. The FRR was 0.003 (1 over 310 logins) for Office, 0.003 (1 over 370 logins) for TrainStation, and 0.006 (2 over 338 logins) for Cafe.

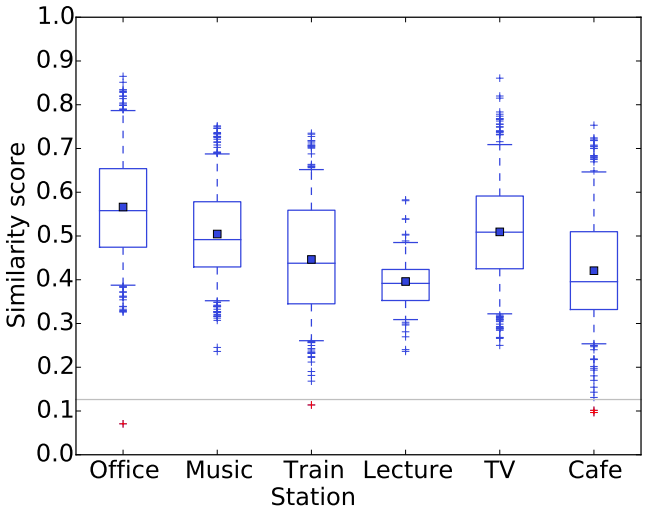
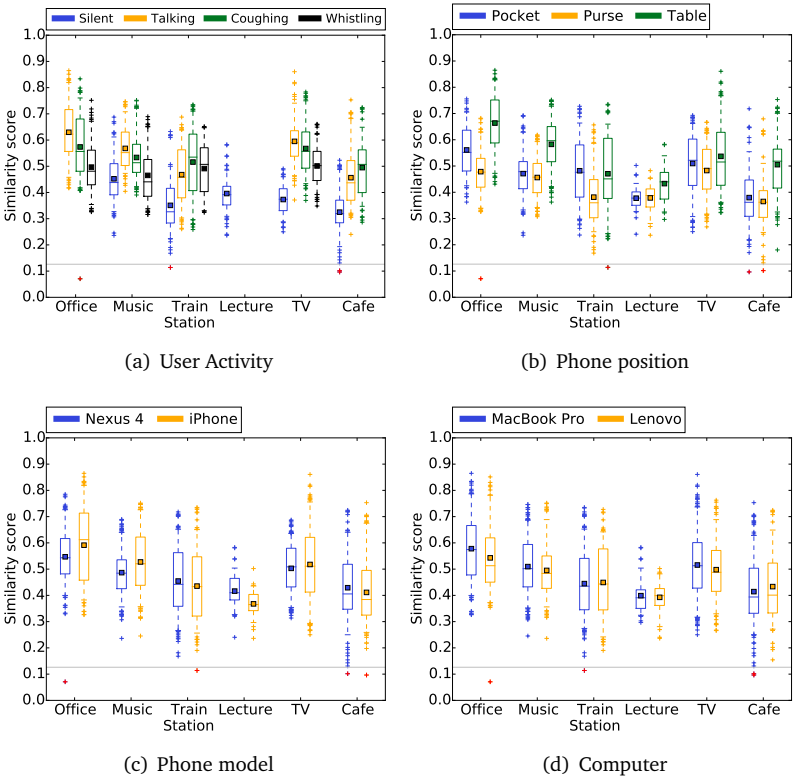



Figure 5.5: Impact of the environment on the False Rejection Rate.

**User Activity.** Figure 5.6(a) shows the similarity scores for different user activities. In general, if the user makes any noise the similarity score improves. We only experienced a few rejections of legitimate logins when the user was silent (TrainStation and Cafe) or when he was coughing (Office). In the Lecture case the user could only be silent. We also avoided whistling in the cafe, because this may be awkward for some users. The FRR was 0.005 (3 over 579 logins) when the user was silent, 0.002 (1 over 529 logins) when the user was coughing, 0 (0 over 541 logins) when the user was speaking, and 0 (0 over 353 logins) when the user was whistling.
**Phone Position.** Figure 5.6(b) shows the similarity scores for different phone positions. Sound-Proof performs slightly better when the phone is on a table or on a bench. Worse performance when the phone is in a pocket or in a purse are likely due to the attenuation caused by the fabric around the microphone. The FRR was 0.001 (1 over 667 logins) with the phone on a table, 0.001 (1 over 675 logins) with the phone in a pocket, and 0.003 (2 over 660 logins) with the phone in a purse.
**Phone Model.** Figure 5.6(c) shows the similarity scores for the two phones. The Nexus 4 and the iPhone 5 performed equally good across all environments. The FRR was 0.002 (2 over 884 logins) with the iPhone 5 and 0.002 (2 over 1118 logins) with the Nexus 4.
**Computer.** Figure 5.6(d) shows the similarity scores for the two computers we used. We could not find significant differences between their performance. The FRR was 0.002 (3 over 1299 logins) with the MacBook Pro and 0.001 (1 over 703 logins) with the Dell.

**Distance Between Phone and Computer.** In some settings (e.g., at home), the user's phone may be away from his computer. For instance, the user could leave the phone in his bedroom while watching TV or working in another room. We evaluated this scenario by placing the computer close to the TV in a living-room, and testing Sound-Proof while the phone was away from the computer. For this set of experiments we used the iPhone 5 and the MacBook Pro. The average noise level by the TV was measured at 50dB. We tested 3 different distances: 4, 8 and 12 meters (running 20 login attempts for each distance). All login attempts were successful (i.e., FRR=0). We also tried to log in while the phone was in another room behind a closed door, but logins were rejected.

**Discussion.** Based on the above results, we argue that the FRR of Sound-Proof is small enough to be practical for real-world usage. To put it in




(a) User Activity

(b) Phone position

(c) Phone model

(d) Computer

Figure 5.6: Impact of user activity, phone position, phone model, and computer model on the False Rejection Rate.

perspective, the FRR of Sound-Proof is likely to be smaller than the FRR due to mistyped passwords (0.04, as reported in [168]).

### 5.6.3 Advanced Attack Scenarios

A successful attack requires the adversary to submit a sample that is very similar to the one recoded by the victim's phone. For example, if the victim is in a cafe, the adversary should submit an audio sample that features typical sounds of that environment. In the following we assume a strong adversary that correctly guesses the victim's environment. We also evaluate the attack success rate in scenarios where the victim and the attacker access the same broadcast audio source from different locations.

| | False Acceptance Rate | | |
|---|---|---|---|
| | SC-SP | SC-DP | DC-DP |
| TV channel 1 | 1 | 0.1 | 0.1 |
| TV channel 2 | 1 | 1 | 0 |
| TV channel 3 | 1 | 0 | - |
| TV channel 4 | 1 | 0 | - |
| Web radio 1 | 1 | 0 | 0.4 |
| Web radio 2 | 0.1 | 0.8 | 0.8 |
| Web TV 1 | 0 | 0 | 0 |
| Web TV 2 | 0 | 0 | 0 |

Table 5.2: False Acceptance Rate when the adversary and the victim devices record the same broadcast media. SC-SP stands for "same city and same Internet/cable provider", SC-DP stands for "same city but different Internet/cable providers", DC-DP stands for "different cities and different Internet/cable providers". A dash in the table means that the TV channel was not available at the victim's location.

**Similar Environment Attack.** In this experiment we assume that the victim and the adversary are located in similar environments. For each environment, we compute the FAR between each phone sample collected by one subject (the victim) and all the computer samples of the other subject (the adversary). We then switch the roles of the two subjects and repeat the procedure. The FAR for the Music and the TV environments were 0.012 (1063 over 91960 attempts) and 0.003 (311 over 90992 attempts), respectively. The FAR for the Lecture environment was 0.001 (8 over 7242 attempts). When both the victim and the attacker were located at a train station the FAR was 0.001 (44 over 67098 attempts). The FAR for the Office environment was 0.025 (1194 over 47250 attempts). When both the victim and the attacker were in a cafe the FAR was 0.001 (32 over 56994 attempts).

The above results show low FAR even when the attacker correctly guesses the victim's environment. This is due to the fact that ambient noise in a given environment is influenced by random events (e.g., background chatter, music, cups clinking, etc.) that cannot be controlled or predicted by the adversary.

**Same Media Attack.** In this experiment we assume that the victim and the adversary access the same audio source from different locations. This happens, for example, if the victim is watching TV and the adversary

correctly guesses the channel to which the victim's TV is tuned. We place the victim's phone and the adversary's computer in different locations, but each of them next to a smart TV that was also capable of streaming web media. Since the devices have access to two identical audio sources, the adversary succeeds if the lag between the two audio signals is less than $\ell_{max}$. We tested 4 cable TV channels, 2 web radios and 2 web TVs. For each scenario, we run the attack 100 times and report the FAR in Table 5.2. When the victim and the attacker were in the same city, we experienced differences based on the media provider. When the TVs reproduced content broadcasted by the same provider, the signals were closely synchronized and the similarity score was above the threshold $\tau_C$. The FAR dropped in the case of web content. When the TVs reproduced content supplied by different providers, the lag between the signals caused the similarity score to drop below $\tau_C$ in most of the cases. The similarity score sensibly dropped when the victim and the attacker were located in different cities.

## 5.7 User Study

The goal of our user study was to evaluate the usability of Sound-Proof and to compare it with the usability of Google 2-Step Verification (2SV), since 2FA based on one-time codes is arguably the most popular. (We only considered the version of Google 2SV that uses an application on the user's phone to generate one-time codes.) We stress that the comparison focuses solely on the usability aspect of the two methods. In particular, we did not make the participants aware of the difference in the security guarantees, i.e., the fact that Google 2SV can better resist co-located attacks.

We ran repeated-measure experiments where each participant was asked to log in to a server using both mechanisms in random order. After using each 2FA mechanism, participants ranked its usability answering the System Usability Scale (SUS) [42]. The SUS is a widely-used scale to assess the usability of IT systems [30]. The SUS score ranges from 0 to 100, where higher scores indicate better usability.

### 5.7.1 Procedure

**Recruitment.** We recruited participants with a snowball sampling method. Most subjects were recruited outside our department and were not working in or studying computer science. The study was advertised as a user study to "evaluate the usability of two-factor authentication mechanisms". We informed participants that we would not collect any personal information and offered a compensation of CHF 20. Among all respondents to our

email, we discarded the ones that were security experts and ended up with 32 participants.

**Experiment.** The experiment took place in our lab where we provided a laptop and a phone to complete the login procedures. Both devices were connected to the Internet through WiFi. We set up a Gmail account with Google 2SV enabled. We also created another website that supported Sound-Proof and mimicked the Gmail UI.

Participants saw a video where we explained the two mechanisms under evaluation. We told participants that they would need to log in using the account credentials and the hardware we provided. We also explained that we would record the keystrokes and the mouse movements (this allowed us to time the login attempts).

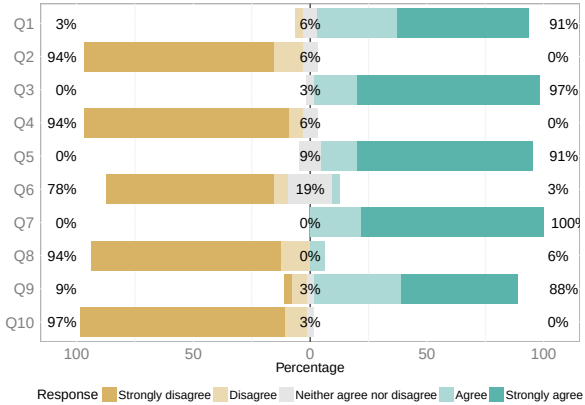
We then asked participants to fill in a pre-test questionnaire designed to collect demographic information. Participants logged in to our server using Sound-Proof and to Gmail using Google 2SV. We randomized the order in which each participant used the two mechanisms. After each login, participants rated the 2FA mechanism answering the SUS.

At the end of the experiment participants filled in a post-test questionnaire that covered aspects of the 2FA mechanisms under evaluation not covered by the SUS.

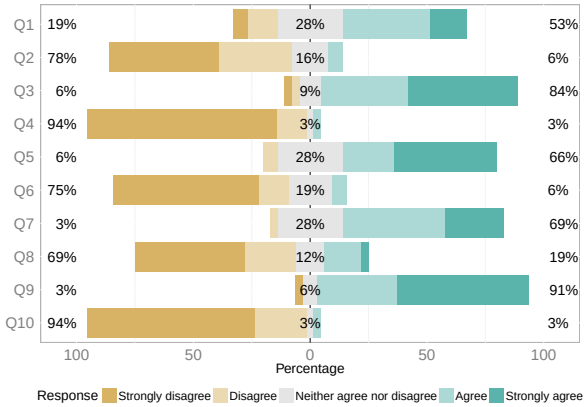
### 5.7.2 Results

**Demographics.** 58% of the participants were between 21 and 30 years old. 25% of the participants were between 31 and 40 years old. The remaining 17% of the participants were above 40 years old. 53% of the participants were female. 69% of the participants had a master or doctoral degree. 50% of the participants used 2FA for online banking and only 13% used Google 2SV to access their email accounts.

**SUS Scores.** The mean SUS score for Sound-Proof was 91.09 (±5.44). The mean SUS score for Google 2SV was 79.45 (±7.56). Figure 5.7(a) and Figure 5.7(b) show participant answers on 5-point Likert-scales for Sound-Proof and for Google 2SV, respectively. To analyze the statistical significance of these results, we used the following null hypothesis: "there will be no difference in perceived usability between Sound-Proof and Google 2SV". A one-way ANOVA test revealed that the difference of the SUS scores was statistically significant ($F(1,31) = 21.698,\ p < .001,\ \eta_p^2 = .412$), thus the null hypothesis can be rejected. We concluded that users perceive




(a) SUS answers for Sound-Proof




(b) SUS answers for Google 2SV

Figure 5.7: Distribution of the answers by the participants of the user study. System Usability Scale (SUS) of Sound-Proof (a) and Google 2-Step Verification (b). Percentages on the left side include participants that answered "Strongly disagree" or "Disagree". Percentages in the middle account for participants that answered "Neither agree, nor disagree". Percentages on the right side include participants that answered "Agree" or "Strongly agree".

Sound-Proof to be more usable than Google 2SV. Appendix B reports the items of the SUS.

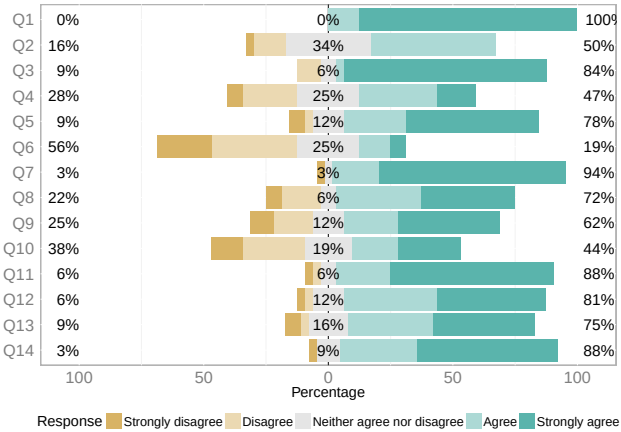



Figure 5.8: Distribution of the answers to the Post-test questionnaire. Percentages on the left side include participants that answered "Strongly disagree" or "Disagree". Percentages in the middle account for participants that answered "Neither agree, nor disagree". Percentages on the right side include participants that answered "Agree" or "Strongly agree".

**Login Time.** We measured the login time from the moment when a participant clicked on the "login" button (right after entering the password), to the moment when that participant was logged in. We neglected the time spent entering username and password because we wanted to focus only on the time required by the 2FA mechanism. Login time for Sound-Proof was 4.7 seconds (±0.2 seconds); this time was required for the phone to receive the computer's sample and compare it with the one recorded locally. With Google 2SV, login time increased to 24.4 seconds (±7.1 seconds); this time was required for the participant to take the phone, start the application and copy the verification code from the phone to the browser.

**Failure Rates.** We did not witness any login failure for either of the two methods. We speculate that this may be due to the priming of the users right before the experiment, when we explained how the two methods work and that Sound-Proof may require users to make some noise in quiet environments.

**Post-test Questionnaire.** The post-test questionnaire was designed to collect information on the perceived quickness of the two mechanisms (Q1–Q2) and participants willingness to adopt any of the schemes (Q3–Q6). We also included items to inquire if participants would feel comfortable using the mechanisms in different environments (Q7–Q14). Figure 5.8 shows participants answers on 5-point Likert-scales. The full text of the items can be found in Appendix C.

All participants found Sound-Proof to be quick (Q1), while only 50% of the participants found Google 2SV to be quick (Q2). If 2FA were mandatory, 84% of the participants said that they would use Sound-Proof (Q3) and 47% said that they would use Google 2SV (Q4). In case 2FA were optional the percentage of participants willing to use the two mechanisms dropped to 78% for Sound-Proof (Q5) and to 19% for Google 2SV (Q6). Similar to [47, 215], our results for Google 2SV suggest that users are likely not to use 2FA if it is optional. With Sound-Proof, the difference in user acceptance between a mandatory and an optional scenario is only 6%.

We asked participants if they would feel comfortable using either mechanism at home, at their workplace, in a cafe, and in a library. 95% of the participants would feel comfortable using Sound-Proof at home (Q7) and 77% of the participants would use it at the workplace (Q8). 68% would use it in a cafe (Q9) and 50% would use it in a library (Q10). Most participants (between 91% and 82%) would feel comfortable using Google 2SV in any of the scenario we considered (Q11–Q14).

The results of the post-test questionnaire suggest that users may be willing to adopt Sound-Proof because it is quicker and causes less burden, compared to Google 2SV. In some public places, however, users may feel more comfortable using Google 2SV. In Section 5.8 we discuss how to integrate the two approaches.

The post-test questionnaire allowed participants to comment on the 2FA mechanisms evaluated. Most participants found Sound-Proof to be user-friendly and appreciated the lack of interaction with the phone. Appendix D lists some of the users' comments.

## 5.8 Discussion

**Software and Hardware Requirements.** Similar to any other 2FA based on software tokens, Sound-Proof requires an application on the user's phone. Sound-Proof, however, does not require additional software on the computer and seamlessly works with any HTML5-compliant browser that implements the WebRTC API. Chrome, Firefox and Opera, already support

WebRTC and a version of Internet Explorer supporting WebRTC will soon be released [192]. Sound-Proof needs the phone to have a data connection. Moreover, both the phone and the computer where the browser is running must be equipped with a microphone. Microphones are ubiquitous in phones, tablets and laptops. If a computer such as a desktop machine does not have an embedded microphone, Sound-Proof requires an external microphone, like the one of a webcam.

**Other Browsers.** Section 5.6 evaluates Sound-Proof using Google Chrome. We have also tested Sound-Proof with Mozilla Firefox and Opera. Each browser may use different algorithms to process the recorded audio (e.g., filtering for noise reduction), before delivering it to the web application. The WebRTC specification does not yet define how the recorded audio should be processed, leaving the specifics of the implementation to the browser vendor. When we ran our tests, Opera behaved like Chrome. Firefox audio processing was slightly different and it affected the performance of our prototype. In particular, the Equal Error Rate computed over the samples collected while using Firefox was 0.012. We speculate that a better Equal Error Rate can be achieved with any browser if the software token performs the same audio processing of the browser being used to log in.

**Privacy.** The noise in the user's environment may leak private information to a prying server. In our design, the audio recorded by the phone is never uploaded to the server. A malicious server can also access the computer's microphone while the user is visiting the server's webpage. This is already the case for a number of websites that require access to the microphone. For example, websites for language learning, Gmail (for video-chats or phone calls), live chat-support services, or any site that uses speech-recognition require access to the microphone and may record the ambient noise any time the user visits the provider's webpage. All browsers we tested ask the user for permission before allowing a website to use `getUserMedia`. Moreover, browsers show an alert when a website triggers recording from the microphone. Providers are likely not to abuse the recording capability, since their reputation would be affected, if users detect unsolicited recording.

**Quiet Environments.** Sound-Proof rejects a login attempt if the power of either sample is below $\tau_{dB}$. In case the environment is too quiet, the website can prompt the user to make any noise (by, e.g., clearing his throat, knocking on the table, etc.).

**Fallback to Code-based 2FA.** Sound-Proof can be combined with 2FA mechanisms based on one-time codes, like Google 2SV. For example, the webpage can employ Sound-Proof as the default 2FA mechanism, but give to the user the option to log in entering a one-time code. This may be useful in cases where the environment is quiet and the user feels uncomfortable making any noise. Login based on one-time codes is also convenient when the phone has no data connectivity (e.g., when roaming).

**Failed Login Attempts and Throttling.** Sound-Proof deems a login attempt as fraudulent if the similarity score between the two samples is below the threshold $\tau_C$ or if the power of either sample is below $\tau_{dB}$. In this case, the server may request the two devices to repeat the recording and comparison phase. After a pre-defined number of failed trials, the server can fall-back to a 2FA mechanism based on one-time codes. The server can also throttle login attempts in order to prevent "brute-force" attacks and to protect the user's phone battery from draining.

**Login Evidence.** Since audio recording and comparison is transparent to the user, he has no means to detect an ongoing attack. To mitigate this, at each login attempt the phone may vibrate, light up, or display a message to notify the user that a login attempt is taking place. The Sound-Proof application may also keep a log of the login attempts. Such techniques can help to make the user aware of fraudulent login attempts. Nevertheless, we stress that the user does not have to attend to the phone during legitimate login attempts.

**Continuous Authentication.** Sound-Proof can also be used as a form of continuous authentication. The server can periodically trigger Sound-Proof, while the user is logged in and interacts with the website. If the recordings of the two devices do not match, the server can forcibly log the user out. Nevertheless, such use can have a more significant impact on the user's privacy, as well as affect the battery life of the user's phone.

**Alternative Devices.** Our 2FA mechanism uses the phone as a software token. Another option is to use a smartwatch and we plan to develop a Sound-Proof application for smartwatches based on Android Wear and Apple Watch. We speculate that smartwatches can further lower the false rejection rate because of the proximity of the computer and the smartwatch during logins.

**Logins from the Phone.** If a user tries to log in from the same device where the Sound-Proof application is running, the browser and the application will capture audio through the same microphone and, therefore, the login attempt will be accepted. This requires the mobile OS to allow access to the microphone by the browser and, at the same time, by the Sound-Proof application. If the mobile OS does not allow concurrent access to the microphone, Sound-Proof can fall back to code-based 2FA.

**Comparative Analysis.** We use the framework of Bonneau et al. [37] to compare Sound-Proof with Google 2-Step Verification (Google 2SV), with PhoneAuth [63], and with the 2FA protocol of [241] that uses WiFi to create a channel between the phone and the computer (referred to as FBD-WF-WF in [241]). The framework of Bonneau et al. considers 25 "benefits" that an authentication scheme should provide, categorized in terms of usability, deployability, and security. Table 5.3 shows the overall comparison. The evaluation of Google 2SV in Table 5.3 matches the one reported by [37], besides the fact that we consider Google 2SV to be non-proprietary.

*Usability:* No scheme is scalable nor it is effortless for the user because they all require a password as the first authentication factor. They are all "Quasi-Nothing-to-Carry" because they leverage the user's phone. Sound-Proof and PhoneAuth are more efficient to use than Google 2SV because they do not require the user to interact with his phone. They are also more efficient to use than FBD-WF-WF, because the latter requires a non-negligible setup time every time the user logs in from a new computer. All mechanisms incur some errors if the user enters the wrong password (Infrequent-Errors). All mechanisms also require similar recovery procedures if the user loses his phone.

*Deployability:* Sound-Proof, PhoneAuth, and FBD-WF-WF score better than Google 2SV in the category "Accessible" because the user is asked nothing but his password. The three schemes are also better than Google 2SV in terms of cost per user, assuming users already have a phone. None of the mechanisms is server-compatible. Sound-Proof and Google 2SV are the only browser-compatible mechanisms as they require no changes to current browsers or computers. Google 2SV is more mature, and all of them are non-proprietary.

*Security:* The security provided by Sound-Proof, PhoneAuth, and FBD-WF-WF is similar to the one provided by Google 2SV. However, we rate Sound-Proof and PhoneAuth as not resilient to targeted impersonation, since a targeted, co-located attacker can launch the attack from the victim's

environment. FBD-WF-WF uses a paired connection between the user's computer and phone, and can better resist such attacks.

| Scheme | Usability | | | | | | | | | | | Deployability | | | | | Security | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 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 |
| Sound-Proof | | S | | Y | Y | S | S | S | Y | Y | | Y | | Y | S | | Y | Y | | Y | Y | Y | Y | Y | |
| Google 2SV | | S | | Y | S | S | S | S | S | S | | Y | Y | Y | S | S | Y | Y | | Y | Y | Y | Y | Y | |
| PhoneAuth | | S | | Y | Y | S | S | S | Y | Y | | | | Y | S | | Y | Y | | Y | Y | Y | Y | Y | |
| FBD-WF-WF | | S | | Y | S | S | S | S | Y | Y | | | | Y | S | S | Y | Y | | Y | Y | Y | Y | Y | |

Table 5.3: Comparison of Sound-Proof against Google 2-Step Verification (Google 2SV), PhoneAuth [63], and FBD-WF-WF [241], using the framework of Bonneau et al. [37]. We use 'Y' to denote that the benefit is provided and 'S' to denote that the benefit is somewhat provided.

## 5.9 Related Work

In chapter 4 we discussed alternative approaches to 2FA. In the following we review related work that leverages audio to verify the proximity of two devices.

Halevi et al., [144] use ambient audio to detect the proximity of two devices to thwart relay attacks in NFC payment systems. They compute the cross-correlation between the audio recorded by the two devices and employ machine-learning techniques to tell whether the two samples were recorded at the same location or not. The authors claim perfect results (0 false acceptance and false rejection rate). They, however, assume the two devices to have the same hardware (the experiment campaign used two Nokia N97 phones). Furthermore, their setup allows a maximum distance of 30 centimeters between the two devices. Our application scenario (web authentication) requires a solution that works (*i*) with heterogeneous devices, (*ii*) indoors and outdoors, and (*iii*) irrespective of the phone's position (e.g., in the user's pocket or purse). As such, we propose a different function to compute the similarity of the two samples, which we empirically found to be more robust, than what proposed in [144], in our settings.

Truong et al., [256] investigate relay attacks in zero-interaction authentication systems and use techniques similar to the ones of [144]. They propose a framework that detects co-location of two devices comparing features from multiple sensors, including GPS, Bluetooth, WiFi and audio. The authors conclude that an audio-only solution is not robust to detect co-location (20% of false rejections) and advocate for the combination of multiple sensors. Furthermore, their technique requires the two devices to sense the environment for 10 seconds. This time budget may not be available for web authentication.

The authors of [234] use ambient audio to derive a pair-wise cryptographic key between two co-located devices. They use an audio fingerprinting scheme similar to the one of [142] and leverage fuzzy commitment schemes to accommodate for the difference of the two recordings. Their scheme may, in principle, be used to verify proximity of two devices in a 2FA mechanism. However, the experiments of [234] reveal that the key derivation is hardly feasible in outdoor scenarios. Our scheme takes advantage of noisy environments and, therefore, can be used in outdoor scenarios like train stations.

## 5.10 Summary and Future Work

Two-factor authentication is an effective mechanism to prevent attackers from accessing users' accounts and data. Deployed solutions have seen little adoption as users find it cumbersome to change their behavior when authenticating to a website.

We proposed Sound-Proof, a two-factor authentication mechanism for web logins that does not require the user to interact with his phone. In our solution the second authentication factor is the vicinity of the user's smartphone to the computer from which he is logging in. In particular two simultaneous recordings of the surrounding ambient audio are performed on the two devices and compared to test for their proximity. Sound-Proof is deployable today and works with major browsers. The user does not have to interact with his smartphone upon login and we have shown how our system works even if the phone is the user's pocket or purse as well as in a wide variety of environments. In comparison to Google 2-Step Verification, the participants in our user study found Sound-Proof to be more usable. More importantly, the majority said that they would use Sound-Proof for online services for which two-factor authentication is optional. We see the possibility to foster large-scale adoption of two-factor authentication for the web with a solution that is both usable and deployable today.

### 5.10.1 Future Work

With Sound-Proof we presented a two-factor authentication solution that is transparent to the user as he logs into a website. We now discuss interesting directions for future research.

**Audio Comparison.** Our audio comparison algorithm is based on cross-correlation. We perform some optimizations to make it more suitable to our needs such as filtering out lower and higher frequency bands. While this approach has shown to give good results in the experimental evaluation that we performed in multiple environments, we believe that there is room for improvement. Most audio comparison frameworks have seen research in order to perform a fast and accurate lookup of a short audio sample in a large samples dataset (e.g., to find a short and noisy recording of a song in a music catalog). Our use-case is different and different comparison techniques can be further researched to improve the accuracy without hindering usability.

**Feature Extraction.** In Sound-Proof, the audio sample recorded by the browser is sent to the phone to perform the audio comparison. While this works well in scenarios where the user's smartphone has data connection, it becomes problematic to achieve the same functionality when that is not the case. One option to consider is to extract audio features from the recording on the browser so that they fit, for example, in an SMS message that can be sent to the phone. This solution requires further research into which features can be extracted from the audio sample while preserving the accuracy and security of our solution.

**Security Guarantees.** In this work we presented an empirical evaluation of our proposed solution. We collected a large number of samples in various environments and showed the robusteness of Sound-Proof balancing security and usability. Future work can focus on understanding the physical properties of the audio samples that can be recorded with current platforms, evaluate them in terms of their entropy, and fully gauge the adversary's probability to successfully produce an audio sample that would match the user's one.

**User Studies.** We acknowledge the limitations of our small-scale user study, which focused on the usability aspects of Sound-Proof. A possible direction for future research is to perform further user studies in the field of two-factor authentication. In particular it would be interesting to understand how users actually interact with the second authentication factor and if, or how, they perceive the security benefits of two-factor authentication. Future user studies would enable researchers to propose solutions that better suit the needs of end users and possibly enable faster and more widespread adoption.

# Chapter 6
# Practical and Secure Location Verification Tokens for Payments

## 6.1 Introduction

Fraudulent transactions at points of sale made with stolen or duplicated payment cards are a major problem. In 2010 alone, these transactions constituted one third of the 1.26 billion EUR total fraud in the Single Euro Payments Area [97]. To improve the security of existing payment systems, companies and researchers have suggested usage of mobile devices as a two-factor authentication mechanism [109, 210]. As most users already have smartphones, deployment of such two-factor authentication is practical. Many online service providers already employ two-factor authentication using smartphones. Examples of this approach are online banking applications [32] and Google 2-Step Verification [129]. In a typical implementation, the user reads a one-time code off the smartphone screen and enters it on the service's web page during login. Login operations to services like online banking are typically performed when the user has time to interact with his smartphone to complete the authentication process. In addition, web services are easily modifiable. Thus, in most cases, an extra authentication step can be added to the login procedure of a web service at little cost. This approach cannot be, however, integrated in point of sale transactions, because interactions at a shop counter with a smartphone are inconvenient and add undesirable transaction delay. Additionally, the payment terminal infrastructure is hard to modify.

Recent proposals leverage location data from the user's phone as the second authentication factor for payments [109, 210]. During a transaction, either the card issuer [109] or the user [210] can verify that the location of the user's smartphone matches the location of the point of sale terminal used for the transaction. Previous work, however, overlooks important usability and security aspects, as it requires changes in both the point of sale infrastructure and the user experience. Furthermore, it assumes a trustworthy mobile OS, even though compromise of mobile operating systems has become commonplace [102, 288].

To secure mobile services despite mobile OS compromise, researchers have proposed using system-wide Trusted Execution Environments (TEEs), such as ARM TrustZone [16], which provide isolated execution of applica-

tions and secure storage of credentials [179, 231]. Integrating system-wide TEEs with any two-factor authentication protocol, however, requires the verifying party (e.g., the card issuer) to correctly bind a user's identity to the TEE running on his mobile device through an enrollment scheme. How to establish this binding in the presence of a compromised OS is an open problem [186].

In this chapter we propose a smartphone-based two-factor authentication solution for payments at points of sale that uses location data to identify fraudulent transactions. In contrast to previous work, our solution does not require changes to established user interaction models and is compatible with the existing point of sale infrastructure. We leverage system-wide TEE architectures to provide a secure system, despite mobile OS compromise. As part of our solution, we design two secure enrollment schemes for smartphones to bootstrap two-factor authentication for payments, which may also be used in other application scenarios. To summarize, We make the following contributions.

- We propose a smartphone-based two-factor authentication solution for payments at points of sale, that uses the phone's location as the second authentication factor. Our solution makes use of smartphone TEEs to resist mobile OS compromise.
- As part of our solution, we construct two secure enrollment schemes that allow a card issuer to bind the identity of a user with the TEE running on his device. The first enrollment scheme leverages the unique identity of the user's SIM card and resists adversaries that can remotely compromise the victim's mobile OS. The second enrollment scheme uses specially crafted SMS messages that are processed within the device baseband OS. This scheme provides protection against more powerful adversaries that can additionally perform hardware attacks on devices to which they have physical access.
- Through prototype implementation using an open-source baseband OS, Android devices, and an ARM TrustZone development board, we show that our solution can be easily deployed. It requires small changes to existing smartphones, and no changes to the point of sale infrastructure and the user experience. Our experiments show that, during a transaction, location verification takes less than 4 seconds on average, which is a tolerable delay for most payment scenarios.
- We survey known approaches for two-factor authentication using mobile phones and argue why they cannot be deployed in the con-

sidered application scenario. We also analyze commonly suggested enrollment schemes and show why they do not withstand strong attackers that we consider in this work.

## 6.2 Problem Statement

Our goal is to design a smartphone-based two-factor authentication mechanism that prevents fraudulent transactions at points of sale. In the following we detail the requirements for any deployable and secure solution to this problem.

We first aim to design mechanisms that must not change the user interaction model and the current point of sale infrastructure. Previous work shows that introducing changes to established user interaction models makes adoption of new security mechanisms impractical [37]. Similarly, having to update the deployed points of sale makes adoption of additional security mechanisms hard [140].

Second, a solution must remain secure despite a targeted adversary that may compromise the mobile OS on the victim's device. Current systems [32, 129] and related research proposals [63, 185], which use smartphones for two-factor authentication, assume the mobile OS to be trustworthy. This assumption is too strong, as the complexity of smartphone platforms has increased, mobile operating system vulnerabilities have become commonplace [102, 288].

Third, any two-factor authentication mechanism that replaces dedicated tokens with smartphones, must have an enrollment scheme, where the verifying party binds the identity of a user to his device. A dedicated security token is a user-specific device, which the service provider binds to the user identity before the token is shipped to the user. As smartphones replace such tokens, the service provider can only bind the user identity to his device after the user has already purchased the smartphone. In addition to initial enrollment, a practical solution must also support device migration. In applications like payments at points of sale, it is realistic to assume a one-time service registration performed, for example, when the user visits a branch of his bank in person. Requiring a similar operation every time the user starts using a new smartphone becomes both expensive for the bank and inconvenient for the user.

## 6.3 Background on TrustZone-enabled Smartphones

Most mobile devices support system-wide TEEs, like ARM TrustZone [16]. In a TrustZone-enabled device, the application processor supports two

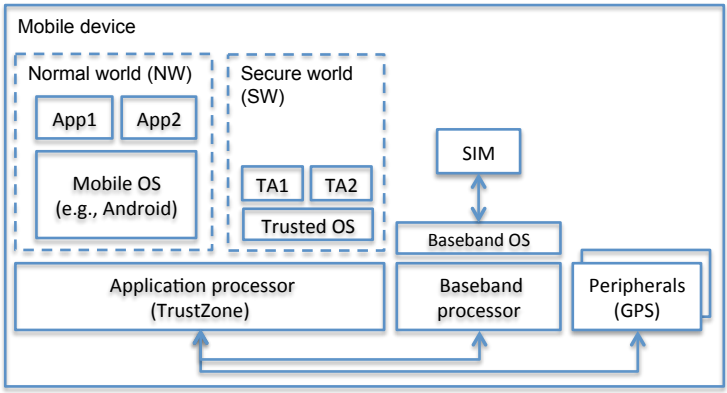



Figure 6.1: Architecture overview of a TrustZone-enabled device.

execution states, namely, *secure world* and *normal world* (or non-secure world). The processor switches between these states in a time-slicing manner, so that only one state is active at a time. The normal world runs the mobile OS and regular applications on top of it. The secure world runs *trusted applications* (TAs), which are executed on top of a small layer of software called the *trusted OS*. The overview of the architecture of a TrustZone-enabled mobile device is shown in Figure 6.1.

In general the security extensions that are implemented in an ARMv7 system-on-chip make it seem as if there were two virtual CPUs running on a single physical core. Figure 6.2 shows the details of the security extensions implemented in the CPU core. Processors that support security extensions must provide: two separate virtual Memory Management Units (MMUs) two separate Virtual Memory Address spaces for both worlds; an additional processor mode (the *monitor* mode); a new instruction (`SMC` — secure monitor call) to switch to monitor mode; banked and common coprocessor configuration registers; and finally, the possibility to tag cache lines as secure and non-secure.

Applications in the secure world are isolated from the mobile OS in the normal world. In contrast to dedicated security elements like smart cards and TPMs, system-wide TEEs like TrustZone allow secure access to various device hardware resources from the TEE, and to configuration of secure mobile device event handling. Access to device peripherals and the baseband processor environment is typically possible by both the trusted OS in the secure world and the mobile OS in the normal world. In addition, certain peripherals can be reserved for secure world access only. Access to

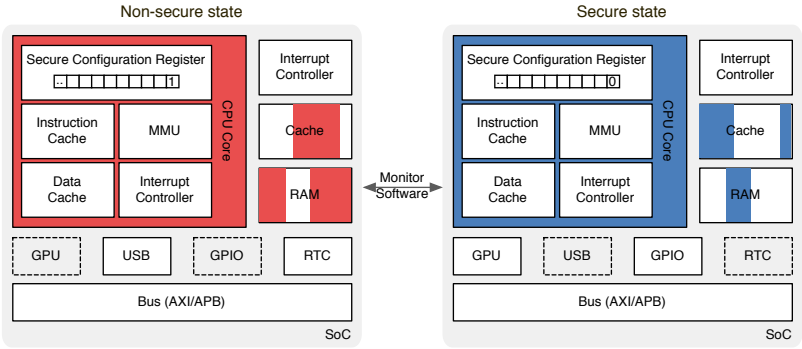



Figure 6.2: Sample configuration for a TrustZone-enabled SoC with virtual cores in the Non-secure and Secure state.

memory areas can be configured in a similar manner, and in a mobile device a small amount of memory is reserved for the secure world. Access control to hardware resources is implemented through specific control hardware and signals on the system communication bus.

This separation is enabled in a TrustZone-aware device by augmenting some system components. In particular, as seen in Figure 6.3, the system supports external memory isolation through the TZASC (TrustZone Access Space Controller), internal memory isolation through the TZMA (TrustZone Memory Adapter), an AXI-to-APB Bridge (Advance eXtensible Interface to Advanced Peripheral Bus) to connect low-bandwidth peripherals, and the TZPC (TrustZone Protection Controller) to configure the TZMA adapter.

Hardware interrupts can also be configured for secure world processing if need be. For more details on TrustZone, we advise the interested reader to look at the ARM documentation [16].

A standard trusted OS only allows execution of code that has been signed by a trusted authority, such as the device manufacturer. Typically, the device manufacturer ships each device with a device-specific key-pair (we refer to it as the *device key*). The public part of the device key is certified by the manufacturer, and the issued *device certificate* contains an immutable device identifier, such as the IMEI number (International Mobile Equipment Identity). The corresponding private key is only accessible by software that runs in the secure world [167].

To limit the size of the TCB, a typical trusted application handles only security-critical processing, such as user credential processing or data encryption. A *companion application*, running in the normal world, handles

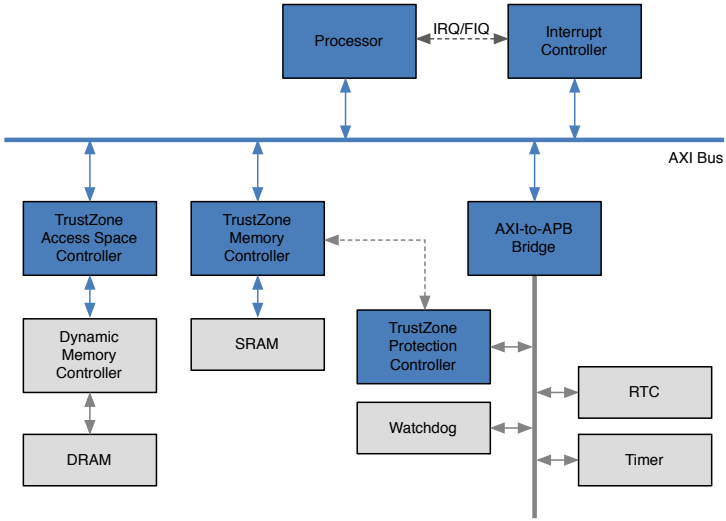



Figure 6.3: TrustZone-aware components (highlighted in blue) in an example ARM system design.

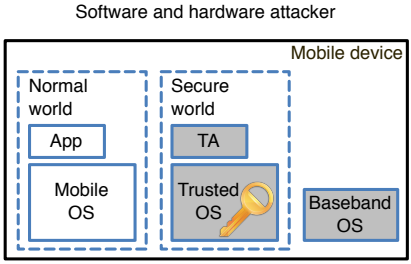
the communication, the UI rendering and other complex tasks. The rationale is that inclusion of complex libraries in the trusted OS, like network stacks or video drivers, considerably increases the size of the device TCB, and with that the attack surface of the secure world.

Device manufacturers have shipped their mobile devices with system-wide TEEs like ARM TrustZone for almost a decade. The usage of these environments, thus far, has been primarily limited to a few manufacturer-specific use cases, like implementation of subsidy locks and secure boot [167]. Deployment of third-party applications in system-wide TEEs has been limited, because the installation of new trusted applications is subject to the approval of the device manufacturer. Nevertheless, recent research shows that system-wide TEEs can be safely opened up for third-party trusted application development [166], and on-going TEE API standardization activities [121] are likely to make trusted application deployment more accessible to third-parties.

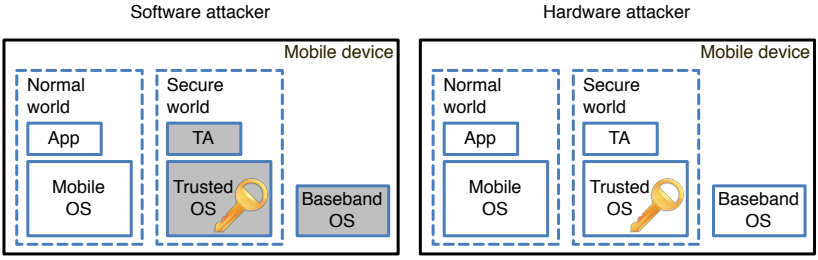
## 6.4 Adversarial Model

We consider a targeted adversary who possesses the victim's payment card (or a clone) and knows its PIN code (if any). His goal is to perform a

fraudulent transaction at a point of sale. The adversary does not have physical access to the victim's smartphone. He does, however, have access to other similar devices. We regard the device hardware, the trusted OS and the baseband OS as the TCB on the victim's device, and distinguish between two types of adversary.




(a) Victim's device




(b) Adversary's device

Figure 6.4: Adversarial model. Grey boxes show trusted components for each type of adversary. The victim's device (a) can be only targeted with remote attacks and the TCB is the same in both attacker models. Trusted components on the adversary's device (b) depend on the considered attacker model: a software attacker cannot tamper with the TCB on his device; a hardware attacker has complete access to any component of his device.

A *software* attacker can remotely compromise the mobile OS on the victim's smartphone, but cannot compromise its TrustZone secure world nor the baseband OS execution. Likewise, he cannot compromise the TrustZone secure world nor the baseband OS on any other device. Software attacks against the mobile OS are a real threat [102, 288], while software attacks

against the TCB (i.e., the baseband OS and the TrustZone secure world) are significantly harder, due to its limited size and attack surface.

A *hardware* attacker can additionally perform hardware attacks against devices that he owns or has physical access to. On such devices, he may compromise the baseband OS execution, the TrustZone secure world and, ultimately, extract the TrustZone-protected secrets, such as the device key. This adversarial model is justified by the fact that neither a TrustZone-enabled processor nor the baseband processor provide tamper resistance properties, commonly found in smart cards or hardware security modules. Figure 6.4 illustrates trusted software components (gray boxes) in these different scenarios.

Neither of the attackers controls the cellular network communication. Furthermore, they cannot launch GPS spoofing attacks on the victim's device. Finally, we do not address denial-of-service attacks.

## 6.5 Our Solution

We use the location of the user's phone as the second authentication factor during a transaction at a point of sale. Our solution leverages system-wide TEEs available on mobile devices to provide card issuers with trustworthy location information despite a potentially compromised mobile OS on the user's smartphone. We focus on ARM TrustZone since it is currently the most widely deployed system-wide TEE on mobile devices. The schemes we propose can, nevertheless, be used with other system-wide TEEs as well.

Figure 6.5 shows an overview of our scheme. Prior to payments at a point of sale, we assume that the user has already installed two applications provided by the card issuer on his device: a companion application running in the normal world and a trusted application running in the secure world. Additionally, the user has also completed an enrollment scheme (see below), and the card issuer has established a binding between the user and the TEE on his device. During payment, the user inserts or swipes his payment card in a point of sale terminal and optionally enters its PIN code (step 1). The terminal sends the transaction information to the card issuer (step 2). The card issuer contacts the TEE on the user's smartphone (step 3), which replies with a location statement (step 4). The card issuer then checks whether the location statement was sent by the correct device and compares it against the location of the terminal. Finally, the card issuer sends the transaction decision (authorize or deny) to the terminal (step 5).

We leverage location data due to two main reasons. First, we want a solution that does not change the user interaction model and the hardware infrastructure (see Section 6.2). We therefore resort to the sensing

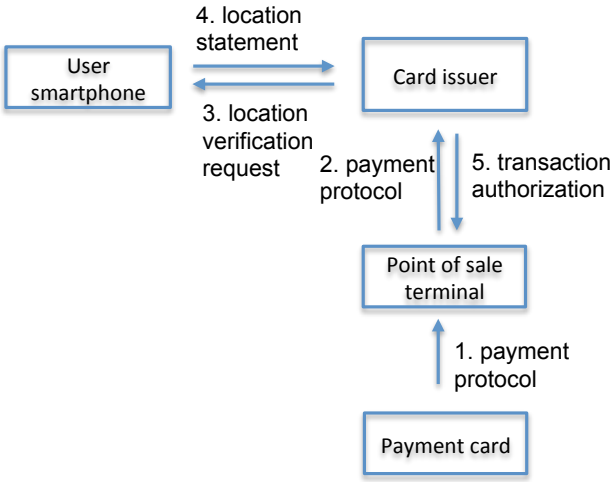



Figure 6.5: Overview of location-based two-factor authentication for payments at point of sale. During a payment transaction, the card issuer queries the user's smartphone for its location over an Internet connection.

capabilities of modern smartphones. Second, among available smartphone sensors, GPS units are almost ubiquitous, and previous work has shown that GPS coordinates are a practical and useful means that card issuers can leverage to identify fraudulent transactions [109, 210]. Our solution could, in principle, use any sensor available on the device.

### 6.5.1 User Enrollment

Before the card issuer can verify the location of the user's smartphone, it needs to bind the user identity to the TEE running on his mobile device. To achieve this binding, we present two enrollment schemes. The *signed-IMSI enrollment* scheme is easier to deploy but can only withstand software attackers; the *baseband-assisted enrollment* scheme is also secure against hardware attackers. However, the latter requires minor software changes to the baseband OS. Both schemes leverage the implicit binding between the user and his SIM card. They require a one-time registration in which the user provides his phone number to the card issuer in a reliable manner, for example, by visiting his bank's branch in person. The goal of both enrollment schemes is to establish a shared *service key*, between the card issuer and the trusted application running in the TEE on the user's device.

**Signed-IMSI enrollment.** The card issuer uses the SIM identifier (i.e., the IMSI) and the mobile network infrastructure to verify that the enrolling device is indeed the one where the user's SIM card is installed. Figure 6.6 illustrates the steps of the enrollment scheme.

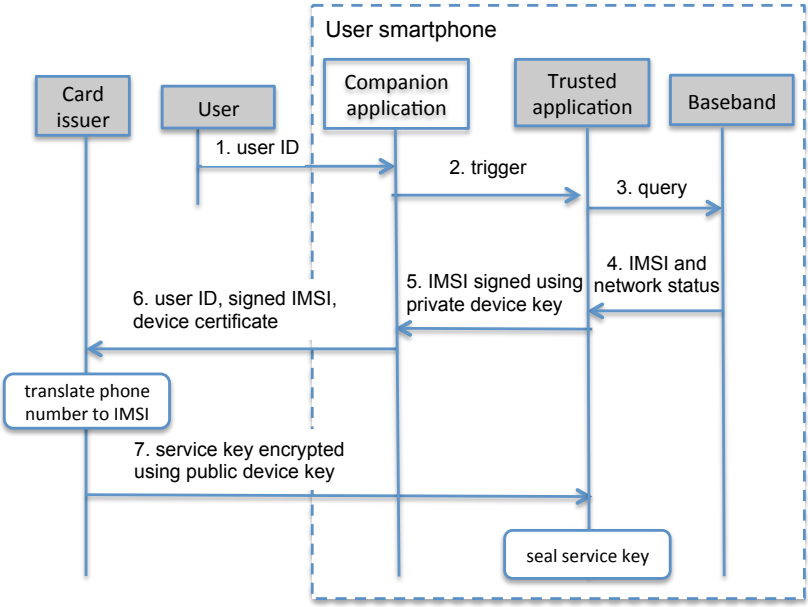



Figure 6.6: Signed-IMSI enrollment scheme. Gray boxes are trusted entities. The trusted application fetches the IMSI of the installed SIM card from the baseband processor. It signs the IMSI with the private device key and forwards it to the card issuer (through the companion application). The card issuer can link the IMSI to a previously registered phone number.

The user starts the companion application and provides his user ID, (e.g., the bank customer number, step 1). This application triggers the execution of the trusted application (step 2) that queries the baseband OS for the IMSI of the SIM card (steps 3-4).[1] The trusted application also verifies from the baseband OS that the device is connected to the mobile network, to discard the possibility of a fake SIM card reporting a false

[1]The IMSI is needed for cellular protocols and is available to the baseband OS through standardized interfaces.

IMSI.[2] The trusted application signs the IMSI using its device private key (step 5) and passes the signature to the companion application. At this point the companion application sends the signed IMSI, the user ID, and the device certificate to the card issuer over the Internet (step 6). The card issuer uses the received user ID to retrieve the user's phone number and queries the mobile infrastructure for the corresponding IMSI (see Section 6.7 for details). The card issuer checks that the IMSI received from the user's phone matches the one reported by the mobile infrastructure. If the two IMSIs match, the card issuer proceeds to verify (*i*) the validity of the device public key using the device certificate and the public key of the manufacturer, and (*ii*) the validity of the signature over the IMSI. If all checks are successful, the card issuer picks a fresh service key and encrypts it under the device public key; the ciphertext is sent to the user's smartphone (step 7). The companion application passes the encrypted service key to the trusted application that decrypts it using the private part of the device key and encrypts it using a symmetric storage key available only in the secure world (*sealing*). The sealed service key can be stored by the companion application in the normal world.

**Baseband-assisted Enrollment.** In this scheme the card issuer sends an SMS message carrying an *enrollment key* to the phone number provided by the user during registration. We augment the baseband OS to use this key and compute an authentication tag on the device's IMEI.[3] The steps of this scheme are shown in Figure 6.7.

The user starts the companion application and provides his user ID (step 1), which is forwarded to the card issuer over the Internet (step 2). The card issuer sends an enrollment SMS message to the corresponding user's phone number, containing a fresh enrollment key (step 3). The baseband OS on the user's device intercepts the SMS message and extracts the enrollment key. The baseband OS uses the enrollment key to authenticate the device's IMEI,[4] provides the authentication tag to the companion application (step 4), and deletes the enrollment key. The companion application forwards the authenticated IMEI and the device certificate to the card issuer (step 5). The card issuer checks (*i*) the validity of the device certificate, (*ii*) the validity of the authentication tag, and (*iii*) that the IMEI authenticated with the enrollment key matches the one in the received certificate. If all checks

[2] A false SIM card lacks the correct keying material to connect to the cellular network

[3] The IMEI is bound to the device key by the device certificate.

[4] The IMEI is read from a read-only memory on the device, written by the device manufacturer. The baseband OS has access to the IMEI to handle cellular communication.

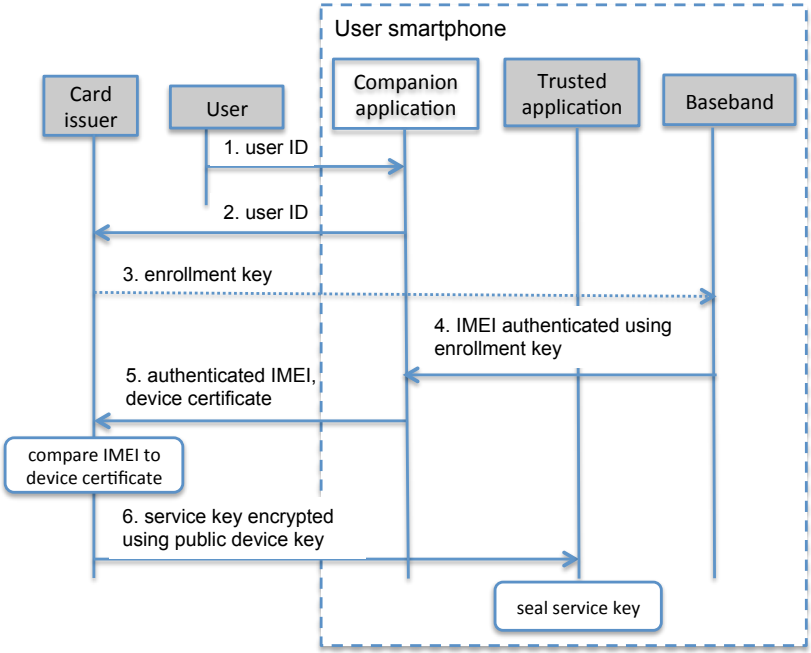



Figure 6.7: Baseband-assisted enrollment scheme. Gray boxes are trusted entities. The card issuer sends an SMS message with an enrollment key (dotted line); the baseband OS uses that key to authenticate the device's IMEI and deletes the key. The card issuer can check the authenticated IMEI against the one received in the device certificate.

are successful, the card issuer picks a fresh service key and encrypts it under the device public key extracted from the device certificate; the ciphertext is sent to the user's smartphone (step 6). The companion application passes the encrypted service key to the trusted application that seals it.

### 6.5.2 Location Verification

After successful enrollment, the user's device shares a service key with the card issuer that can be used to create an authentic channel between the two parties. During a transaction payment, therefore, the card issuer can query the device for an authenticated location statement. As detailed in Figure 6.8, the location verification is a simple challenge-response protocol.

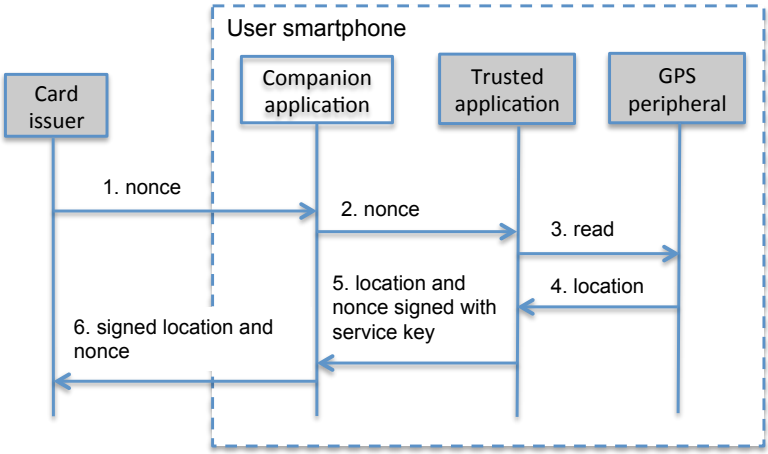



Figure 6.8: Location verification is a simple challenge-response protocol using the service key established during enrollment. Gray boxes are trusted entities. The card issuer sends a nonce; the trusted application reads the GPS coordinates and authenticates them together with the nonce.

The card issuer picks a fresh nonce (for replay protection) and sends it to the trusted application through the companion application (steps 1-2).[5] The trusted application reads the location coordinates from the device GPS unit (steps 3-4), unseals the service key and uses it to authenticate the nonce and the current coordinates. The authenticated message is sent back to the card issuer through the companion application (steps 5-6). The card issuer verifies the authenticity of the location statement using the service key. At this point, the card issuer matches the location of the user's smartphone against the one of the terminal used for the transaction, to decide whether to authorize or deny the transaction.

We note that neither the companion application nor the trusted application need to be continuously running in the background. The execution of the trusted application is triggered by the companion application which,

[5]Alternatively, the card issuer could verify the freshness of location statements checking the timestamp provided by the GPS unit. As one of our goals is not to change the payment transaction user experience, the location verification must be initiated by the card issuer. Thus, a location verification requires the exchange of two messages between the card issuer and the user smartphone in either cases (i.e., usage of the GPS timestamp does not allow a more efficient implementation).

in turn, is started by the mobile OS when receiving a request for that application (e.g., through a push notification).

### 6.5.3 Device Migration

Both enrollment schemes support device migration by re-running the enrollment operation. When the user switches to a new device and moves his SIM card to it, he can start the enrollment process from the companion application of the new device. As a result of either the signed-IMSI or the baseband-assisted enrollment, the card issuer invalidates the previously used service key, re-associates the user identity to the device key of the new device, and sends a fresh service key to the TEE on that device. Device migration does not require out-of-band communication with the card issuer as long as the user keeps his phone number (even if he gets a new SIM card associated with his old phone number).

## 6.6 Security Analysis

Recall that our adversary holds the victim's payment card (or a clone) and his goal is to make fraudulent transactions at points of sale. The adversary does not have physical access to the victim's smartphone but he does have remote control over that smartphone's mobile OS.

With the deployment of our system, the adversary must convince the card issuer that the enrolled user's smartphone is close to the terminal where the fraudulent transaction is taking place. To do so, the adversary must either succeed in an impersonation attack during enrollment or tamper with the location verification protocol.[6]

In an impersonation attack, the adversary must halt the enrollment scheme on the victim's device (he can do so since he controls that device's mobile OS), and use the victim's ID to run the enrollment scheme on a device that he owns. In particular, the adversary has two possible *strategies*: he must induce the card issuer to encrypt the service key either

(a) under the public key of the adversary's device (the adversary will thus be able to make fraudulent transactions if his phone is in proximity of the terminal where the transaction takes place), or

(b) under a public key for which the adversary knows the private key (so that the adversary will be able to generate arbitrary location statements when the card issuer requests them).

[6] We acknowledge that the adversary may still succeed in his goal if the fraudulent transaction takes place close to where the victim's device is located.

In the following we provide an informal analysis of the enrollment schemes, with respect to the two strategies above. Finally, we argue that the location verification mechanism is secure after successful user enrollment.

### 6.6.1 Signed-IMSI enrollment

This enrollment scheme is secure against a software attacker as defined in Section 6.4. This attacker can control the mobile OS on any device (including the victim's) but does not have sufficient capabilities to control any baseband OS or the TrustZone secure world execution environment.

Strategy (a) requires the adversary to start an enrollment scheme on his device using the victim's ID. Since the card issuer knows the victim's phone number and can retrieve the corresponding IMSI, the adversary must force the trusted application on his device to send the IMSI of the victim's SIM card. To do so, the adversary may use a custom SIM card where he can manipulate the IMSI. However, such SIM card misses the key that the victim's SIM card uses for authentication with the network operator and, thus, cannot connect to the cellular network. When the trusted application on the adversary's device queries the baseband OS for cellular network status, it detects that the phone is not connected and will abort the enrollment scheme.

Strategy (b) requires the adversary to hold a private key corresponding to a valid (i.e., certified by the device manufacturer) public key. This is not possible since a software adversary cannot compromise the ARM TrustZone architecture of any device and leak the secrets stored therein.

### 6.6.2 Baseband-assisted Enrollment

This enrollment scheme is secure against a hardware attacker, as defined in Section 6.4, that controls the mobile OS on any device (including the victim's), as well as the baseband OS and the TrustZone secure world execution environment on devices to which he has physical access.

Strategies (a) and (b) require the adversary to either intercept the enrollment SMS message and extract the enrollment key, or provide a crafted IMEI to the baseband OS on the victim's device. Since the adversary does not control the GSM network, SMS messages cannot be intercepted. Furthermore, the enrollment key is deleted by the baseband OS, so that the normal world cannot read it. Finally, the IMEI is stored on read-only memory during device manufacturing [167], thus the adversary cannot feed an arbitrary IMEI to the baseband OS on the victim's device.

### 6.6.3 Location Verification

After a successful enrollment, the trusted application running in the secure world on the victim's device shares a service key with the card issuer. At this time, the adversary can only try to force the victim's device to report a location statement with GPS coordinates matching the location where the fraudulent transaction takes place. Since none of the considered adversaries (i.e., software and hardware) control the secure world on the victim's device, the adversary can only try to change the coordinates provided by the GPS unit on that device. We note that GPS units on modern smartphones only allow to reset the GPS sensor. That is, the adversary cannot feed the trusted application on the user's phone with arbitrary coordinates. The trusted application can, nevertheless, detect a reset and restart the GPS sensor.

## 6.7 Implementation

We implement three prototypes to evaluate the feasibility of the proposed two-factor authentication solution and enrollment schemes. First, we modify an open-source baseband OS to show that the changes required to existing baseband operating systems are small. We test our baseband modifications on an older mobile phone, because the baseband environment on modern smartphones is not modifiable by third-party developers.

Second, we implement the trusted application on a TrustZone development board to show that its deployment on TrustZone-enabled devices is straightforward, and that the time overhead to generate location statements is negligible compared to network delays. We use a TrustZone development board because installation of trusted applications on current smartphones requires approval by the device manufacturer.

Third, we implement a client-server prototype using an Android smartphone to evaluate the end-to-end performance of the location verification mechanism at the time of a payment transaction.

### 6.7.1 Baseband Implementation

To accommodate the baseband-assisted enrollment scheme of Section 6.5.1, we augment osmocomBB [206], which is the only available open-source baseband OS. It is implemented for Motorola mobile phones like the C123 or C118, introduced in 2005. The GSM layer 1 (the physical layer) executes directly on the mobile phone, while layers 2 and 3 (respectively the data-link layer and the third layer, subdivided in the Radio Resource management, the Mobility Management, and the Connection Management) run as the

`mobile` application on a host PC connected with the device through a USB-to-serial cable.

We leverage the widely used SMS Protocol Data Unit mode, standardized in [1], to format the enrollment SMS message sent by the card issuer to the user's phone number. In the standard, a User Data Header structure can contain so called Information Element Identifier elements, that are reserved for future use. We encode the enrollment key in the Information Element Data field of one such identifier. We add to the baseband OS the logic to identify and handle enrollment SMS messages. Once the key is found in the SMS message header, the baseband OS extracts it and computes an authentication tag over the device's IMEI. We use the HMAC-SHA1 implementation provided by the mbed TLS [21] library as the authentication algorithm.

We test the prototype baseband OS in a Motorola C118 connected through a USB-to-serial cable to a host PC running Ubuntu 12.10. The original `mobile` application provided by osmocomBB consists of 19,482 lines of C code; we add a total of 523 lines of code, where `polar_sha1.c` accounts for 451 lines of code. Our changes increase the code size by 2.7%. In terms of binary size, a compiled version of the original `mobile` application is 2029 kB; our modified version accounts for 2077 kB in total (i.e., 2.3% larger). The layer1 firmware (`layer1.compalram.bin`) that is installed on the mobile phone accounts for 63 kB and remains unmodified.

### 6.7.2 Trusted Application Implementation

We implement a trusted application that provides location statements, on a TrustZone-enabled development board. We use it to evaluate the implementation complexity and the time required to produce location statements. The board is an ARM Motherboard Express uATX [19] coupled with an ARM CoreTile Express A9 [18]. The board features a Cortex-A9 processor which is clocked at 400 MHz. The development board contains no GPS unit or baseband processor. In the normal world of the system, we run Android version 4.1.1 with Linux kernel version 2.6.38.7, properly patched to support the ARM board as well as Android. In the secure world, we run Open Virtualization SierraTEE [243], release "02 June 2013". SierraTEE is an open-source framework that provides a basic secure world kernel, compliant with the GlobalPlatform TEE specifications [121].

The implementation of the trusted application accounts for less than 150 lines of code. Thus, incorporating our trusted application into an existing trusted OS, that already provides the necessary cryptographic functions

and system calls, would hardly change the existing memory and storage requirements.

**Location Verification.** The application that generates location statements runs on top of Open Virtualization, in the secure world, while the companion application runs in the normal world, on top of Android. When the card issuer initiates a location verification protocol with the user's smartphone, that device switches from normal world to secure world and executes the trusted application that generates the location statement.

We set up an experiment where the companion application, running in the normal world, invokes the trusted application and provides it with a 128-bit nonce. As the development board has no GPS unit, we emulate it by creating a system call in the secure kernel that just returns longitude, latitude and accuracy values. The trusted application runs HMAC-SHA256 over the data fetched from the system call and the provided nonce. The location statement is returned to the companion application in the normal world. A shared memory buffer is used for exchanging data between the two worlds.

We measure the total time required for the companion application to receive a location statement from the trusted application. This time includes (i) the performance delay introduced by the context switching and required data copying between the normal world and the secure world, and (ii) the time it takes for the trusted application to generate a location statement. The above experiment is repeated 1000 times. Average completion time is 3.0 milliseconds, with a standard deviation of 0.04 milliseconds. The time spent in context switching between the normal world and the secure world is below one millisecond.

**Enrollment.** Since our board is not equipped with a baseband processor, we do not implement the enrollment schemes of Section 6.5.1. Nevertheless, we now explain how they can be realized.

During the signed-IMSI enrollment scheme, the trusted application must query the baseband OS for the IMSI of the installed SIM card, and for the cellular network status. In a mobile device, communication between the baseband OS and the mobile OS (e.g., Android) is implemented through a manufacturer-supplied binary (e.g., a driver). A stripped-down version of this binary may be as well installed in the secure world by the device manufacturer. Given that subsidy lock is one of the most used services in TrustZone-enabled devices, it is reasonable to assume that the

secure world is able to communicate with the SIM card in a modern smartphone [167]. For reference, the full binary in the Samsung Galaxy S3 phone (i.e., `libril.so`) is 49 kB. The complete API offers roughly 200 function calls (extracted by looking at the *strings* of the binary) to the baseband OS. In contrast, the stripped-down version to support enrollment only requires the function calls `GET_SIM_STATUS` and `GET_IMSI`.

In the baseband-assisted enrollment scheme, the trusted application is only invoked at the end of the process to decrypt and seal the service key sent by the card issuer. Hence, there is no requirement for direct communication between the secure world and the baseband OS.

### 6.7.3 Client-Server Implementation

To evaluate the performance of the location verification protocol, the client prototype provides the functionalities of both the companion application running in normal world, and the trusted application running in secure world. This implementation does not account for the needed context switching between the normal world and the secure world. As mentioned before, this time is below one millisecond, and thus negligible compared to networking delays of a full end-to-end implementation.

We develop against the API level 16 of the Android SDK (version 4.1, "JellyBean" [123]). Cryptographic operations are based on the Bouncy Castle crypto library [173]. We use 2048-bit RSA keys as device keys. Authentication of location statements leverages HMAC-SHA256 with an 128-bit service key. Communication between the server and the client uses the push notification feature of Google Cloud Messaging (GCM) [130]; the reverse channel is a standard HTTP connection.

The client provides functionalities for the signed-IMSI enrollment scheme (cf. Section 6.5.1) and the location verification mechanism (cf. Section 6.5.2). During enrollment, the application queries the baseband OS through the Android Java API provided by the `TelephonyManager` service, for the IMSI of the SIM card and the network connection status. During location verification, the application reads the GPS location (latitude and longitude, accuracy and satellite fix time) using the `LocationManager` system service.

The server-side processing is implemented in python, using the CherryPy Web Framework [59] and SQLite [246]. This web service is accessed through a RESTful web API that provides enrollment and location verification operations. During the signed-IMSI enrollment scheme, the server translates a phone number to the corresponding IMSI using an HLR-lookup query with an external service provider. An HLR-lookup query is carried

out by network operators using the Signalling System #7 (SS7) protocols. In particular, the Home Location Register (HLR) of a network operator holds information about its users such as their phone numbers and to which network a device is currently connected. Among other information, the HLR holds the IMSI of the SIM card connected to the network. Several HLR lookup services, such as [62], are available to third-parties.

While our client prototype implementation is tailored towards a device using Android OS, similar functionalities are easily done on other smartphone platforms.

## 6.8 Experimental Evaluation

The previous section shows that context switching between the two TrustZone execution states (i.e., normal and secure world) and cryptographic operations to produce location statements, require only a few milliseconds. Network delays, therefore, account for the majority of the time required to verify the location of the user's device by the card issuer. In this section we analyze the time to complete location verification and present the experimental evaluation of our client-server prototype.

We focus on the location verification protocol as the enrollment procedure is a one-time operation and its performance is less critical. The client prototype is installed on a Samsung Galaxy S3 smartphone with the latest software updates (as of the time of writing), after a factory reset. The server is running on a standard laptop and shares a service key with the client. We provide results for both *static tests* run with the phone in a fixed location (office environment) and a *field study* run in a scenario close to actual deployment. Table 6.1 provides an overview of our results which we elaborate below.

| | **Static tests** | | | **Field study (3G)** | |
|---|---|---|---|---|---|
| | WiFi (n=101) | 3G (n=101) | Edge (n=101) | Orange (n=46) | Sunrise (n=34) |
| average (sec) | 0.60 | 1.82 | 2.20 | 2.54 | 3.68 |
| std dev (sec) | 0.08 | 0.05 | 0.30 | 0.78 | 1.45 |

Table 6.1: Completion time for location verification during payment transactions. *n* denotes the number of samples in each scenario.

### 6.8.1 Static Tests

During static tests, the client device is in a fixed position, on a desk in our office environment. We run tests for Edge (GSM only mode), 3G (WCDMA only), and WiFi (mobile data turned off) connections. For each connection setting, we measure the completion time, i.e., how long it takes from the moment the server issues a request, until the moment it receives the location statement and verifies its authenticity. The experiment is repeated 100 times (the server issues one request per second), and Figure 6.9(a) shows the completion time for each location verification. Results show longer completion times during the first runs, for Edge and 3G connections. This behavior is presumably caused by the time it takes to "activate" the radio on the phone. To validate our hypothesis, we set the interval of two consecutive server requests to 30 seconds, allowing the radio of the phone to "deactivate" after each request. Results are shown in Figure 6.9(b). Confirming our hypothesis, completion time in this scenario has greater variance, especially when using Edge or 3G networks.

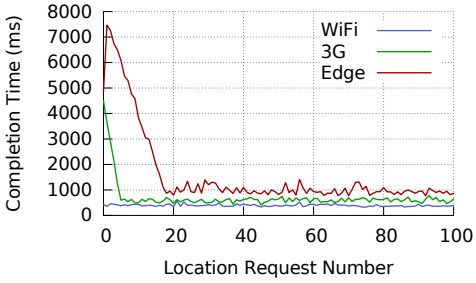
Finally, Figure 6.10 shows the average and the standard deviation for the measurements of Figure 6.9(b). The completion time for our solution is, on average, 2.2 seconds with an Edge connection, 1.82 seconds with a 3G connection, and 0.6 seconds with a WiFi connection, respectively.

### 6.8.2 Field Study

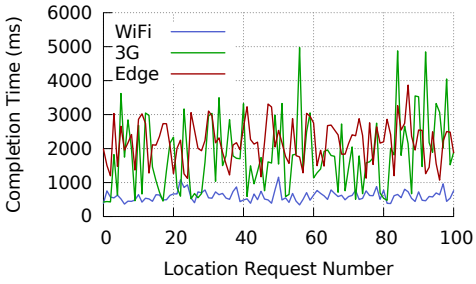
To test our solution in a setting close to the actual payment scenario, we use two client devices and carry out a two-day field study with two subjects in Zürich. Each subject carries a client device and a triggering device. The two clients have SIM cards issued by different operators (*Orange* and *Sunrise*).

The triggering device initiates a server request. The server runs on a standard laptop and listens for incoming triggers. We use a separate trigger device to make sure that, at the time of a location verification, the radio on the client device is not active. In an actual deployment, if the radio happens to be active when a location verification request is received, completion time is expected to be smaller than the ones reported below.

For two days, each subject uses the triggering device to initiate a location verification each time he is close to a payment terminal in, for example, shops, cafes, museums, parking lots, etc. Completion time, for the device with the Orange SIM card, is 2.54 seconds on average (standard deviation 0.78 seconds). The smartphone with the Sunrise SIM card shows slightly worse performance as the completion time raises to 3.68 seconds on average (standard deviation 1.45 seconds).




(a)




(b)

Figure 6.9: Completion time for 100 location verifications. In Figure (a) the server initiates one request per second. In Figure (b) the server waits 30 seconds before issuing each request.

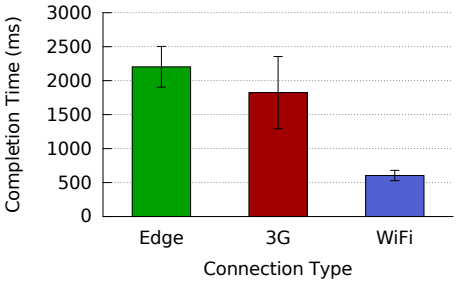



Figure 6.10: Average time and standard deviation required to complete a location verification, for different connection types.

Table 6.2 shows the accuracy of the location information and the elapsed time between the server request and the actual GPS fix on the client. The reported accuracy is within an acceptable range to distinguish shops next to each other (17.40 meters on average), and the location fix is available with a very short delay (less than 257 milliseconds on average).

| **GPS accuracy (m)** | | | **GPS fix delay (ms)** | | |
|---|---|---|---|---|---|
| average | max | min | average | max | min |
| 17.40 | 48.0 | 4.0 | 256.83 | 3430.00 | 0.00 |

Table 6.2: Location accuracy results during the field study.

## 6.9 Discussion

This section further discusses the proposed mechanisms in terms of integration with current payment systems, deployment considerations, and privacy issues. Finally, we discuss the applicability of our solution to other scenarios.

### 6.9.1 Integration with Payment Systems

Our protocols can increase the security of any payment card transaction (either "chip and PIN" or "swipe and sign") where the card issuer is contacted by the terminal to authorize or deny the payment. We now detail the integration of our solution with deployed payment systems and, in particular, with the EMV payment standards [87].

An EMV transaction involves the card, the terminal, the card issuer and an *acquirer*. The acquirer is a banking institution that processes credit and debit card payments for merchants. Third-party payment processors may be also involved, or the issuer and the acquirer may be the same banking institution. EMV specifications for transactions with online authorization dictate two *cryptograms* (authenticated messages) exchanged between the card and its issuer. The cryptogram sent by the card is denoted as the Application Request Cryptogram (ARQC) and accounts for a number of fields that are supplied either by the card or by the terminal. Mandatory fields for ARQC include the transaction amount, the transaction date, and a random nonce generated by the terminal. EMV also defines optional fields that individual payment systems (e.g., Mastercard M/Chip or Visa VSDC) may require. The card issuer replies with an Application Reply Cryptogram (ARPC) that notifies the card whether the transaction has been approved.

Figure 6.11 illustrates the messages defined in the EMV standards; dashed arrows show the additional messages required to implement our solution.

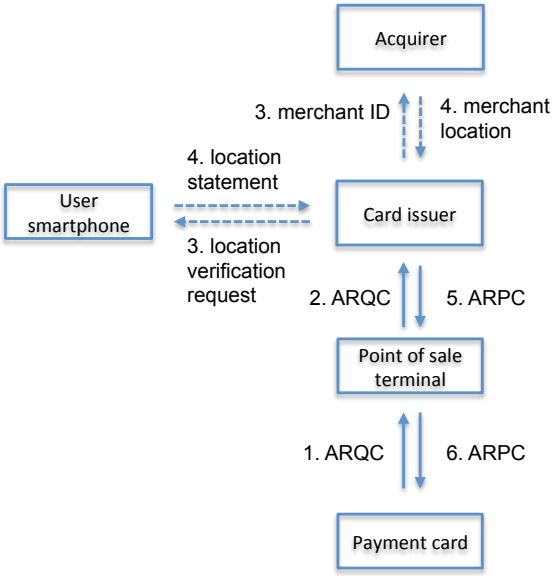



Figure 6.11: Integration of the location verification protocol within the EMV payment standards. Dashed lines indicate the additional messages required for our solution.

Our solution requires the card issuer to know the physical location of the terminal used for a transaction. In the EMV standards an *acquirer ID* globally identifies a banking institution, while a *merchant ID* identifies a merchant within a banking institution. Either the card or the terminal can request that these elements are included in the ARQC message. Once the card issuer learns the acquirer and merchant identifiers of a transaction, it can contact the acquirer with the purported merchant ID in order to retrieve the merchant location. Popular payment systems are already providing merchant information, such as location, through standardized APIs [187, 267].

In current EMV payment systems, the acquirer ID and the merchant ID fields are defined by the standard but are not mandatory for ARQC messages. Through card software updates it is possible to include these fields in ARQC messages of every transaction. The EMV standards allow

for remote update of payment card software through *issuer scripts* that may be included in the ARPC cryptogram.

A noteworthy benefit of our solution is that it enables gradual and secure deployment for selected users. A card issuer can enable location verification for a subset of its customers, for example, by updating the payment cards of customers that decide to opt-in. Once location verification is enabled for these users, an adversary that has obtained a payment card of any such user cannot circumvent the system. In comparison, solutions that require gradual terminal updates do not provide similar protection, as the adversary can always bypass the added security mechanisms by utilizing not yet updated terminals.

### 6.9.2 Deployment Considerations

Even though fraud reduction is a clear incentive for banks or payment card issuers to adopt our solution, it is the user adoption that will be the driving factor. The proposed schemes do not change the user experience at points of sale, but they do consume internet bandwidth and battery on the user's device. Since payment card issuers are currently covering the costs of frauds, it is an open question whether users would pay the additional costs in terms of bandwidth and battery on their mobile devices, without any apparent benefit. To boost the user adoption, the card issuer may offer lower fees to users who opt-in, just like car insurance companies do for customers who install anti-theft mechanisms on their cars.

Our solution assumes the user's smartphone to have Internet connectivity at the time of a transaction. This may not hold for two reasons. First, high roaming charges induce many users to turn Internet access off while traveling abroad. International regulatory bodies have started to forbid excessive roaming charges. For example, the EU has recently decided to completely remove roaming charges within its member countries by 2015 [251]. A cost-effective alternative to avoid current roaming charges is to use SMS messages for communication between the user's device and the card issuer. SMS-based communication, can however experience high delays (SMS message delivery is based on a best-effort basis and can take up to 12 seconds in normal load conditions [217]). Second, Internet connectivity might not be available in remote areas or underground locations (although many underground shopping centers or transport stations have cellular coverage).

Card issuers can handle lack of connectivity based on transaction value, merchant location or user specific policies. For example, high-value transactions in areas where Internet connectivity is expected to be available may

only be authorized after a successful location verification with the user's smartphone. A possible solution to handle temporary lack of connectivity for low-value transactions could be to keep, on the device, an authenticated log of timestamped locations and report the one that is closest to the transaction time, once Internet connectivity is again available. While this solution does not allow for real-time fraud prevention, it allows card issuers to perform offline fraud detection.

### 6.9.3 Privacy Considerations

The card issuer can ask the user's device for location statements and track the user over time. However, if the protocol is triggered at times of genuine transactions, our solution does not leak extra information, since the card-terminal transaction already reveals the user location to the card issuer. The card issuer may also abuse the system and query the user's device for a location statement when the card is not involved in a transaction. We argue that the system abuse can be prevented through precise terms of agreement; card issuers that break those terms will damage their reputation and lose customers. Another solution is to let the card issuer send the location of the point of sale terminal to the device and let the device compare it against its current location, as done in [210].

With respect to third-parties (i.e., law-enforcement authorities), location statements issued by the user's device can be denied. Since the (symmetric) service key used to authenticate location statements is shared with the card issuer, no third-party can identify who produced a location statement (either the user's device or the card issuer). Finally, we remark that an adversary in control of the mobile OS on the victim's device can query the GPS unit at will and track the user, independently of the solution presented in this chapter.

### 6.9.4 Other Application Scenarios

Our protocols can be applied to other application scenarios beside payments at points of sale. In particular, they can be used in any scenario where the verifying party (e.g., a service provider) knows the user's phone number and the location of the infrastructure used to perform transactions. Two prominent examples are public transportation ticketing and building access.

In the public transportation scenario the user holds a transport ticket (e.g., an NFC card) that is used at dedicated machines to access the transportation network (e.g., at the entrance of metro stations). Our solution can provide assurance to the public transportation authority that a lost or stolen transport ticket is not used by any party but the rightful owner.

Similarly, building access control systems require a user to carry an access token with a short-range wireless interface. Entry is granted if the token is presented to a dedicated reader and the valid PIN is entered by the user. Location statements can increase the security of such access control systems. Upon presenting the access token to the reader, the user phone is queried for its location; if the purported location matches the one of the reader, the building access authority can grant access.

As part of our field study (described in Section 6.8), we tested completion time and accuracy of our location verification protocol in public transportation and building entrance scenarios. Results are summarized in Table 6.4 and Table 6.3.

Completion time at public transport stations takes 3.03 seconds on average, using a 3G connection, for the device using Orange, and 3.39 seconds on average for the smartphone using Sunrise. We argue that three seconds to grant access to the transportation network may be an undesirable delay. Nevertheless, our solution could be used for offline ticket abuse monitoring. The public transportation authority could disable a ticket after witnessing a number of consecutive fraudulent uses. In this scenario, the measured location accuracy (see Table 6.3) is around 14 meters on average. In most public transportation applications this accuracy is sufficient to distinguish entrances to the transportation network from one another.

| **Scenario** | **GPS accuracy (m)** | | | **GPS fix delay (ms)** | | |
|---|---|---|---|---|---|---|
| | avg | max | min | avg | max | min |
| Building access | 14.04 | 48.0 | 4.0 | 139.31 | 3087.00 | 0.00 |
| Public transport | 15.47 | 48.0 | 6.0 | 210.52 | 4035 | 0.00 |

Table 6.3: Location accuracy for public transport and building access tests.

We also test the time it takes to run our protocol when entering buildings in two campuses of ETH Zurich, one in the city center and the other in the city suburbs. Completion time takes 2.31 and 4.40 seconds on average, depending on the network operator used as shown in Table 6.4. The location accuracy in this scenario is around 15 meters, which is enough to differentiate building doors from each other in most cases.

## 6.10 Alternative Approaches

In this section we discuss alternative ways in which smartphones could provide a second authentication factor for payments at points of sale, and con-

| | Building access | | Public transport | |
|---|---|---|---|---|
| | Orange (n=59) | Sunrise (n=40) | Orange (n=43) | Sunrise (n=63) |
| average (sec) | 2.31 | 4.40 | 3.03 | 3.39 |
| std dev (sec) | 0.57 | 1.74 | 0.66 | 1.33 |

Table 6.4: Completion time for location verification for public transport and building access tests. *n* denotes the number of samples in each scenario.

clude that location verification provides a practical means for card issuers to identify fraudulent transactions. We also analyze commonly suggested enrollment schemes and show how they fail to provide secure user-to-device binding, given our realistic attacker model. Finally, we describe alternative TEEs available on current smartphones and their shortcomings, compared to system-wide TEEs such as ARM TrustZone.

### 6.10.1 Two-factor Authentication Approaches

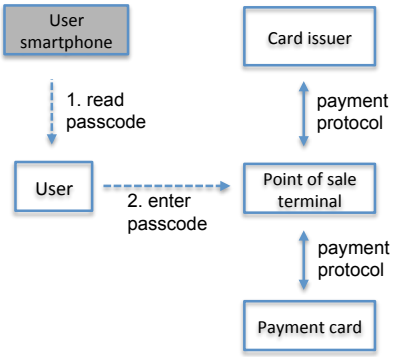



Figure 6.12: *Smartphone as an authentication token replacement:* the user reads a passcode off his smartphone and enters it into the payment terminal. Solid arrows represent standard transaction messages, while dashed arrows show additional messages for two-factor authentication.

In a typical transaction at a point of sale, the user enters or swipes his payment card into a terminal and optionally types its PIN code. The card runs a protocol with the terminal that contacts the card issuer for online transaction verification. We present common two-factor authentication approaches: *(1) Authentication token replacement (Figure 6.12)*. The

smartphone acts as a dedicated authentication token and displays one-time passcodes that the user must type into the terminal. Google 2-Step Verification [129] is a prominent example of this approach in the context of web login authentication. A similar approach can be applied to payments at points of sale. *(2) User confirmation device (Figure 6.13)*. The card issuer contacts the user's device which presents a confirmation dialog to the user. The confirmation result is sent back to the card issuer. Authentication solutions like this one have already been deployed for online banking [261]. *(3) Distance-verification device* (Figure 6.14). The user places his smartphone next to the payment terminal, which starts a distance-verification protocol over a short-range wireless connection, such as NFC [132].

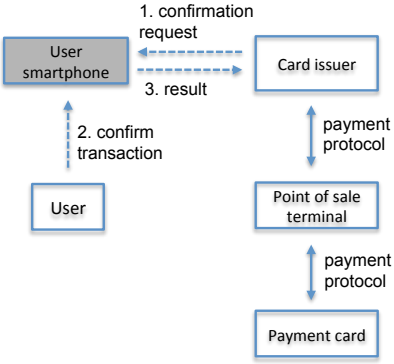



Figure 6.13: *Smartphone as a user confirmation device:* the user confirms the transaction using his smartphone; the devices delivers the user's decision to card issuer. Solid arrows represent standard transaction messages, while dashed arrows show additional messages for two-factor authentication.

The given approaches require additional user interaction at the time of the transaction. Changes to established user interaction models hinder the introduction of new security mechanisms [37]. Additionally, the majority of point of sale terminals do not have the required software components for passcode entry (Figure 6.12) or hardware interfaces for distance-verification (Figure 6.14). The replacement of deployed terminals would be gradual and optional, which allows the adversary to target the terminals that have not been updated yet.

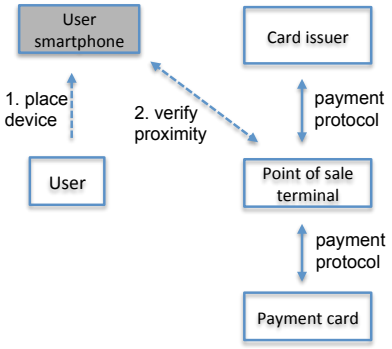



Figure 6.14: *Smartphone as a distance verification device:* the user positions the phone next to the terminal; the terminal verifies the proximity of the phone over short-range wireless connection (e.g, NFC). Solid arrows represent standard transaction messages, while dashed arrows show additional messages for two-factor authentication.

### 6.10.2 Common Enrollment Solutions

We now explain why commonly suggested enrollment schemes are not secure or feasible to deploy, assuming an adversary that controls the mobile OS on the victim's device.

**Device Identifier Enrollment.** A simple way to bind a user identity to the device key of his TEE, is to leverage the device's IMEI, typically included in the device certificate. The IMEI is available on the device's package or displayed on-screen. During enrollment, the user provides the IMEI of his device to the card issuer using a reliable out-of-band channel, for example, visiting a branch of the card issuer in person. The card issuer then verifies the device certificate with respect to the IMEI provided by the user.

Communicating the IMEI to the card issuer in a trustworthy way is more complicated than it seems. Device sales packages are not always available. Also, if the mobile OS is compromised, the adversary controls the IMEI shown on the device screen. Additionally, IMEI-based enrollment does not provide flexible device migration: every time the user changes devices, he must provide the IMEI of the new device to the card issuer, using an out-of-band channel.

**Password Enrollment.** The user-to-device binding can also be implemented by asking the user to enter a password or a similar initialization

secret, known by the card issuer, in his device. A trusted application can authenticate itself to the card issuer, using the certified device key and the user-provided password.

The user should type in the password only when a trusted application can securely receive it. The compromised mobile OS can otherwise intercept and forward the password to the adversary, who can then launch an impersonation attack. A reliable communication interface from the user to a trusted application is called *trusted path* [283, 284]. The device hardware and software resources used for user interaction (e.g., the display buffer or the touchscreen input events) can be temporarily reserved for system-wide TEEs such as ARM TrustZone. A *security indicator*, such as a colored bar on the top of the screen [238] or a dedicated LED [23] can be used to inform the user about the type of application he is communicating with (trusted or untrusted). However, current smartphones do not support this division of user interface resources, nor do they provide dedicated security indicators. Furthermore, several academic studies, and a few decades of practical experience, have shown that users tend to ignore security indicators [83, 156, 232].

Previous work assumes that password-based enrollment is secure if the enrollment is done early in the device lifecycle, before the adversary has the opportunity to compromise the mobile OS [63, 164]. This assumption is hard to justify since not every service enrollment happens at the beginning of a device life time.

**SMS Enrollment.** If the user provides his phone number during registration, the card issuer can send an SMS message that will be received by the device in which the user's SIM card is installed. Similarly to our solution, the SMS message could carry an initialization secret to bootstrap security services. The problem with this approach is that SMS messages provide a trustworthy channel to the the baseband OS of the device where the SIM card is installed, but not to the secure world on that device. In current mobile device architectures, the baseband OS is accessible by both the mobile OS in the normal word and by the trusted applications running in the secure world. Therefore, when the baseband OS receives an SMS message, it notifies the mobile OS, which can read any initialization secret and leak it to the attacker. Assuming a mobile device architecture in which the baseband OS interacts only with the secure world is not feasible, as the mobile OS needs to interact with the baseband for, e.g., phone calls. To overcome this limitations, the baseband-assisted enrollment scheme uses an enhanced baseband OS to achieve secure enrollment.

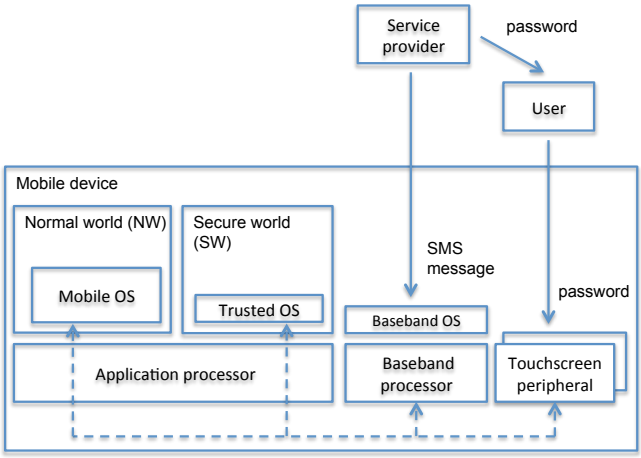



Figure 6.15: Commonly suggested user enrollment schemes. Solid arrows illustrate trustworthy communication channels. Dashed arrows illustrate communication channels in which one of the end points can be either the normal world or the secure world.

**Enrollment Using Initialization Secrets.** Figure 6.15 illustrates the user enrollment problem when using initialization secrets. Any channel to the secure world can be intercepted by the mobile OS in the normal world, be it the baseband OS, the user-interface, or any other interface such as NFC or Bluetooth. For example, the baseband OS cannot pass the initialization secret received over SMS messages to the application processor because it does not know which processor state (normal world or secure world) is active.

### 6.10.3 Alternative Trusted Execution Environments

In addition to ARM TrustZone, SIM cards are TEEs widely available on smartphones. A SIM card can store secrets and execute small pieces of code (often referred to as *SIM applications*) in isolation from the mobile OS. Therefore, one could argue that the location verification mechanism can be implemented as a SIM application within the SIM card processing environment. This approach has two drawbacks. First, provisioning of SIM applications to a SIM card requires the service provider to negotiate with the network operator that issued that SIM card. Providers who target global applications must negotiate with a large number of network

operators separately. Second, in current architectures, SIM cards cannot directly access the device peripherals, such as the GPS unit. When the SIM card needs location information, the application processor must read the coordinates from the GPS unit and provide them to the SIM card. The SIM application has no means of knowing whether these coordinates have been tampered with. Some recent device configurations support a dedicated connection between the SIM card and the device NFC unit [139]. Similar dedicated lines could be added for secure access to the GPS unit. However, targeted hardware changes to allow for specific security services based on SIM applications are costly and hard to justify for widespread adoption.

Some mobile devices are equipped with a slot for SD cards. The latter may be used as a TEE to run code in isolation from the mobile OS or to securely store credentials [114, 235]. However, just like SIM cards, SD cards do not have direct access to the device peripherals, such as the GPS unit, and remote provisioning of applications by third-parties to SD cards is not currently available.

## 6.11 Related Work

In chapter 4 we discussed alternative approaches to 2FA also in the context of payments. In the following we review related work that focuses on secure enrollment and trusted execution environments for smartphones.

*Secure Enrollment.* Secure enrollment for trusted platforms [211] and TEEs on mobile devices [186] is a challenging research problem. Both deployed systems [32, 129] and academic work [116, 179, 210] have overlooked it or assume a non-compromised OS at enrollment time [63, 164].

In [262] the authors evaluate how TrustZone-enabled devices can substitute hardware tokens for two-factor authentication. They assume a trusted path to the user for which secure implementation is problematic. In [263] the authors systematize hardware-based solutions that increase the security guarantees available on mobile platforms. In their work the authors do not address the problem of secure enrollment. Even in recent work that proposes the use of mobile TEEs for security-critical applications [48, 76, 249], the problem of secure enrollment is not addressed.

*Trusted Sensors.* Saroiu and Wolman [231] put forward the problem of trustworthy sensor readings on mobile phones. They propose to leverage virtualization to handle sensors readings and provide signed measurements to applications that run in guest VMs. A similar approach is considered in [115] where the authors consider participatory sensing scenarios and study how to protect user privacy and how to allow for local data processing

while providing (TPM) signed sensor readings. Trustworthy participatory sensing applications are also considered in [78], where the authors design and implement a trusted sensing platform. The latter is a board equipped with sensors and a TPM that communicates with the mobile device over Bluetooth and provides signed sensor measurements.

Liu et al. [179] build on top of Credo [220] and TrustZone to propose software abstractions that expose trusted sensor services to mobile applications. In particular, the authors of [179] focus on scenarios where trustworthy reading must be processed locally by applications (e.g., blurring people's faces from a photo taken by the phone's camera) before being sent. Therefore the framework in [179] aims at protecting the integrity of a sensor reading as well as the local code that must process it. The authors provide a sample application scenario where sensor readings account for GPS traces, while data processing applies noise to the aggregated outcome in order to achieve differential privacy [80].

Plug-n-Trust [245] focuses on mHealth scenarios and provides a system to protect integrity and confidentiality of data collected by body-area network of sensors. Their approach leverages a smart card that plugs into a phone's microSD slot and creates a trusted computing environment for collecting and processing data provided by surrounding sensors. Security rests upon tamper resistance of both the smart card and the sensor nodes where encryption/authentication keys are stored.

YouProve [116] addresses scenarios where sensor readings must be modified by local applications for, e.g., privacy issues. The authors of [116] propose a framework to verify that a modified data item preserves the meaning of the original sensor reading. YouProve uses TaintDroid [89] to track the measurement from the moment it is produced by the sensor, until the moment it is uploaded to a service provider. Each modification is tracked, and a summary of all changes is signed using the secret key of the phone's TPM.

To the best of our knowledge, all previous work on trusted sensors focuses on designing or developing a trusted computing environment on a mobile device in order to cryptographically sign sensor readings. However, these work do not address secure deployment aspects, including secure enrollment and device migration. To the best of our knowledge, we are the first to propose a complete and deployable solution that allows a number of application scenarios to leverage on-smartphone trusted sensor measurements.

## 6.12 Summary and Future Work

As we have seen in the context of web applications, two-factor authentication plays an important role in securing users' accounts and data. In this chapter we have further shown how this extends to other scenarios. In particular we focused our attention to providing a solution that can thwart fraudulent transactions at points of sale.

We proposed a practical solution that adds the required security to recently proposed location-based two-factor authentication mechanisms. We have identified the necessary requirements for a deployable solution, which includes no changes to the user experience and to the infrastructure that is already in place. We have proposed a new mechanism to secure the readings of the smartphone GPS sensor in order to strengthen the security of our solution. Furthermore, we have proposed two novel enrollment schemes that enable secure enrollment and convenient device migration despite a compromised mobile OS. With these solutions we can protect customers transactions at points of sale even in the presence of a motivated adversary that performs targeted attacks against a particular user. Through prototype implementations on current hardware we have shown that our solution is indeed deployable. We further tested the proposed location-based mechanism in a field study and showed that the location verification operations causes an acceptable delay during a payment transaction. Finally, we presented other use-cases where our solution can be used to enhance their security such as entrance to buildings and ticketing.

### 6.12.1 Future Work

In this chapter we presented a solution to strengthen the security of payments at points of sale. We now look at future research directions.

**TrustZone Implementation.** As part of our work we presented prototype implementations that allowed us to showcase the feasibility and deployability of our proposed solutions. Unfortunately, at the time of this writing, it was not possible for us to fully implement the proposed security mechanisms on a standalone device. We believe that an open platform that allows deployment of applications in the secure world of TrustZone-enabled smartphones will enable a better understanding of all the intricacies of the work presented in this chapter. Such research would also provide a strong proof that our work indeed satisfies not only the security but also the timing and usability requirements of our solution.

**Terminals Location.** The assumption we made in this chapter is that the location of payment terminals is known in advance by the service provider that makes the location comparison. In cases where the terminals location is unknown, our proposed solution cannot be deployed. We believe that one could carry out a study on the accuracy of the location information of payment terminals. Furthermore, it is possible that within an organization (e.g., a supermarket chain) payment terminals are moved between different locations due to renovations or relocations. It would be interesting to understand how often this happens and how many false negatives would be generated by such a move if the database is not promptly updated. One could further propose solutions to crowd-source the data in cases where the terminals are moved or the location data is missing. Such a solution would have to take into account an attack window while the data is crowd-sourced. Furthermore, it would be interesting to understand how fast the terminal locations can be successfully crowd-sourced depending on the payments distribution among multiple terminals.

# Part II

# Smartphone Attacks and Countermeasures

# Chapter 7
# Introduction

In the previous part of this thesis, we have shown how smartphones can be used to enhance the security of many daily operations. In Sound-Proof we showcased a solution for web authentication that is resilient to a remote attacker who cannot compromise his victim's mobile OS. In the scenario of payments at points of sale, our solution can withstand a stronger attacker, who is able to compromise the victim's mobile OS. Although mobile platforms implement many security mechanisms they are still vulnerable to attacks [102, 180, 278, 289, 290].

In this part of the thesis we look at the open challenges that the security community faces when focusing on attacks against modern smartphones. Due to their ubiquity and the abundance of information stored on them, smartphones have become the target of various attacks. In particular, through the usage of infected applications, attackers try to exfiltrate users' information, such as location, contacts and application interactions. Malicious applications are typically shipped to customers as a repackaged version of a benign application. The attacker adds the malicious payload and makes the application available as a direct download or through third-party marketplaces.

The infection rates of malicious applications vary considerably depending on the source reporting them. Researchers found low infection rates (as low as 0.0009%) in an empirical study based on domain-resolution traces [174], and slightly higher ones in a direct study leveraging reports from Android devices (approximately 0.27%) [257]. In contrast, the Lookout report from 2014 [180] shows an increase in malware infection rates year over year of 75%, going from 4% to 7%. This report also informs the readers of new malware types ranging from ransomware (malware that locks the user out of his smartphone unless a payment is issued to the attacker, such as ScareMeNot and Koler) to bitcoin mining on infected devices (such as CoinKrypt).

Irrespective of which report more accurately represents the real malware infection rate of smartphones, it is clear that the threat posed by malicious applications is real. We focus our attention on two types of attacks that can exfiltrate users' data. We first analyze the threat posed by application phishing attacks. In these attacks a malicious mobile application masquerades as a legitimate one in order to steal user credentials. Next, we look at how two malicious applications installed on the user's

device can communicate covertly to evade the permission-based security architecture of Android.

In Chapter 9, we provide a summary of application phishing attacks and possible countermeasures. Our analysis shows that personalized security indicators may help users to detect phishing attacks but rely on the user's alertness. Previous studies in the context of website phishing have shown that users tend to ignore personalized security indicators and fall victim to attacks despite indicators deployment. Consequently, the research community has deemed personalized security indicators an ineffective phishing detection mechanism. We revisit the question of personalized security indicator effectiveness and evaluate them in the previously unexplored, and increasingly important context of mobile applications. We designed and developed a mobile banking application that deployed personalized security indicators. We then conducted a user study with 221 participants and found that the deployment of personalized security indicators decreased the phishing attack success rate from 100% to 50%. Personalized security indicators can, therefore, help phishing detection in mobile applications and their reputation as an anti-phishing mechanism should be reconsidered.

While an attentive user can detect application phishing attacks and prevent his credentials from being stolen, we then look at application collusion attacks on mobile platforms. In these attacks, two applications that appear harmless when analyzed individually can collude and escalate their privileges. In particular, users are implicitly led to believe that by approving the installation of each application independently, they can limit the damage that an application can cause.

In Chapter 10 we implement and analyze a number of covert and overt communication channels that enable applications to collude and indirectly escalate their permissions. Furthermore, we present and implement a covert channel between an installed application and a web page loaded in the system browser. We measure the throughput of all these channels as well as their bit-error rate and required synchronization for successful data transmission. The measured throughput of covert channels ranges from 3.70 bps to 3.27 kbps on a Nexus One phone and from 0.47 bps to 4.22 kbps on a Samsung Galaxy S phone; such throughputs are sufficient to efficiently exchange users' sensitive information (e.g., GPS coordinates or contacts). We test two popular research tools that track information flow or detect communication channels on mobile platforms, and confirm that even if these tools detect some channels, they still do not detect all of them and fail to fully prevent application collusion. Attacks using covert

communication channels remain therefore, a real threat to smartphone security and an open problem for the research community.

# Chapter 8
# Related Work

In this chapter we review related work in the area of smartphone attacks and countermeasures. We report both academic and industry work in the area of smartphone malware analysis and technologies to detect and prevent attacks.

## 8.1 Smartphone Malware Attacks

Malicious mobile applications typically exploit platform vulnerabilities (e.g., for privilege escalation) or use system APIs that provide access to sensitive user data. A malicious application can, for example, leak the user location or send SMS messages to premium numbers without user consent. Recently, more sophisticated attacks have emerged where malicious applications cooperated in botnets or mined cryptocurrencies [180]

As of today there have been two major ways through which malware spread on mobile platforms. The majority of malware is shipped as a repackaged legitimate application. The authors of [289] found that in a set of 1260 malware samples, 86% of them are repackaged applications. In these attacks malware authors inject malicious payloads in legitimate applications. The majority of repackaged applications that contain malicious payloads are found on alternative marketplaces (75%). Users download these applications unaware of the embedded malware and the payloads are executed as the user starts the application.

Recently, malware authors have leveraged another infection mechanism. Instead of repackaging applications directly, they have infected the programs used to compile and package the applications. This technique has been used in late 2015 to generate malicious applications for the iOS platform [278]. Developers using an infected version of XCode (i.e., the iOS IDE used to develop, compile and package applications to submit to the AppStore) have packaged malicious software in 39 applications that were then approved for distribution on the AppStore.

The research community has focused their attention on understanding the infection rates of malware on smartphones. On the one hand multiple malware families have been identified. In [289] the authors classify malware samples in 48 distinct families all targeting the Android platorm. The samples have also been made available for further analysis [290]. In [104], the authors analyze 46 different malware samples for Android, iOS and

Symbian platforms. Given the amount of malware samples collected one would think that their infection rates would be high.

On the other hand, recent research has found little traces of malware in the wild. In [174] the authors analyze the traces collected over a three-months period by a US mobile network provider. In particular, the traces are scanned in search of DNS domains lookups associated to malware samples. Only 0.0009% of the population if found to be infected with known malware. Furthermore, the authors note that a malware campaign that lasted for months was stopped even before the malware sample was identified. The authors of [257] also try to quantify the malware infection rates. Their approach differs from the previous work in that they collect signatures of applications installed on approximately 55000 mobile devices. The authors report malware infection rates on Android devices by comparing recorded traces on the phones against two malware datasets. The infection rates are slightly higher than the previous work, depending on which malware dataset is used, namely 0.28% and 0.29%. The authors further propose a solution to identify potentially compromised devices by looking at the applications installed as an indicator of how likely they are to be or become infected.

### 8.1.1 Sophisticated Attacks

While the research community has found little evidence of malware being a widespread problem on mobile platforms, we now present advanced techniques through which malware can exfiltrate users' private data.

**Application Phishing.** Phishing attacks have been first studied in the web context, where a user is redirected to a malicious webpage typically following a hyperlink sent over e-mail or other means [73, 106, 112]. In an application phishing attack, a malicious application presents the user a user interface associated with another application. The user is led to believe that the malicious application is the original one and enters his credentials that are then sent to the attackers.

Application phishing attacks have been reported in the wild [75, 100, 108, 183, 236] leading to not only credential theft but also economical losses. In the context of smartphones, phishing attacks have been first hinted at in [61, 104, 282] and more recently extensively studied, for the Android platform, in [34]. In this latest work, the authors present a systematic evaluation of application phishing attacks and use static analysis techniques to detect applications that use APIs that enable certain classes

of application phishing attacks. We will extend the analysis on application phishing attacks and countermeasures as well as present a first user-study on personalized security indicators in the context of mobile platforms in Chapter 9.

**Application Collusion.** Application collusion attacks are the most recent instantiation of decades-old attacks presented, at first, to exfiltrate data in multi-tier architectures [170]. In these attacks two colluding processes on the same platform can exchange information through the file system. More recent work focused on timing channels, where two processes can exchange bits of information by timing particular operations on the same machine. Cache-based [150] and memory-bus based attacks [151, 277] have been presented for the x86 architecture. More recently, channels that exploit virtual memory deduplication [279], and finally, covert channels that exploit the heat dissipation of processor cores [188] have also been shown to work.

In the context of smartphones, Soundcomber [233] is the first work that looks at colluding applications. In Soundcomber, the authors use the smartphone microphone to harvest sensitive information, such as credit card numbers, by detecting voice and tone patterns. The recording application, then transmits the information to an application that has access to the network, so that the credit card number can be transmitted over the internet. The channels presented use globally-available settings (vibration, volume, screen lock, etc.) or file locks. In Chapter 10 we will focus on application collusion attacks and present a variety of overt and covert channels that can be created on mobile platforms. Recently in [169], the authors have further extended our list of covert communication channels.

**Side Channels.** Application collusion attacks require two applications to synchronize and exchange information. Side channels, in contrast, can be leveraged by malicious applications to extract information out of other applications. In particular, in the context of smartphones, researchers have explored how side channels can be used to infer private information. In [49, 208, 281] the authors propose to use accelerometers, gyroscopes and orientation sensors, available on the majority of smartphones, to infer which characters a user is typing on a keyboard and learn a user's passwords or PIN codes.

In Memento [159], the authors describe a new side channel attack on smartphones. By monitoring memory and CPU usage, a malicious

application can infer which webpage is being loaded in another process (e.g., a web browser) as well as finer-grained information.

**Root Exploits.** There have been a number of root exploits that target mobile platforms. Root exploits allow a malicious application to escalate its privileges and execute at the highest possible privilege (i.e., root). Root exploits typically exploit techniques that have been known for decades for x86 platforms. Examples include buffer overflows [3], format strings vulnerabilities [225] or integer overflows [36]. Many of these techniques can be used to exploit also ARM-based systems, such as the majority of modern smartphones, and therefore a large number of attacks have been shown to be successful both on Android [122, 280] and iOS [57, 92].

## 8.2 Smartphone Malware Countermeasures

We have shown how a number of malware samples have been discovered in the wild and analyzed by both industry and researchers alike. We also presented a number of research work focusing on sophisticated attack techniques that attackers can exploit on current platforms. We now survey work focusing on mobile malware countermeasures.

**Permission-Based Security.** A significant amount of work has been performed, in the past few years, on the Android platform and specifically on the permission-based model [33, 46, 56, 67, 202, 205, 264].

Barrera et al. present an empirical methodology for the analysis and visualization of its permission-based model, which can help in refining the permission system [33]. In particular, the authors find that developers only use a set of available permissions and that some permissions are overly broad. In [101], the authors build a tool to detect overprivileged applications. In their analysis, they find that approximately 33% of analyzed applications have more permissions than needed. Focusing on the same subject, the authors of [60] found that popular applications ask for more permissions than other applications.

The Kirin tool [90] uses predefined security rule templates to match dangerous combinations of permissions requested by applications. Saint [205] allows run-time control over communication among applications according to their permissions. In [46], the authors discuss possible unchecked information flows due to applications that use Broadcast Intents without proper permissions checking.

Moving away from developers and focusing on users' understanding of permissions, in [103], the authors find that users have a poor understanding of permissions and that their display in the Android system yields to poor judgement when installing applications. Now that Android supports permissions revocation after installation new studies should be performed to understand how users behave. In [60], the authors also look at the effectiveness of user-consent permissions systems for Facebook, Chrome and Android applications. They found that users do not have enough context to judge the privacy and security of an application when installing it. Finally, in [209], the authors analyze the description of applications presented to user using natural language processing tools. Their goal is to infer if the permissions requested by applications are motivated through the description of the application behavior.

**Information Leakage.** Information leakage attacks target both users' private data and business data stored on smartphones. In [45], the authors provide a framework that enables the creation of two separate domains on the Android platform. Their framework protects against inter-domain communication at different levels of the stack (e.g., kernel, middleware). In [43, 44], the authors present a security framework that can be implemented on Android to tame confused deputy attacks and application collusion attacks. They introduce a runtime IPC monitor to identify malicious calls that lead to confused deputy attacks. Furthermore, they build a Mandatory Access Control (MAC) framework to detect and prevent covert channels. We will analyze their work in more detail in Chapter 10 testing it against our covert channels implementation.

In order to preven information leakage from private data sources (e.g., the address book or the GPS coordinates of a user), the authors of TaintDroid [89] implement an information flow tracking mechanism. They identify a number of data sources and sinks and track data transmission across the system from a source to a sink. When the system detects that private data is leaving the system the user can be alerted or the operation can be blocked to prevent the data leak. We will test TaintDroid against our implemented overt and covert channels in Chapter 10. With Edgeminer [50], the authors improve the detection rate of information-flow solutions by implementing a solution that tracks data through implicit control flow transitions. Finally, in PiOS [82] the authors provide a static analysis tool to detect data leakage in iOS applications, and found that half of the analyzed applications leaked the unique device identifier. This allows developers to link users' data and usage patterns across multiple applications.

**Control Flow.** In [66], the authors augment the work presented in [82] to enforce application control flow integrity (CFI) and prevent control flow attacks. The solution works against return oriented programming and attacks that circumvent address-space layout randomization (ASLR) techniques. Their prototype implementation works for iOS applications but is also applicable to other ARM-based devices. In comparison to the extended version of the Google Native Client [237], PiOS does not require access to the source-code of the application.

In Xifer [68], the authors propose a new ASLR solution that works both for x86 and ARM systems. This solution is effective against code reuse attacks and works by randomizing all code blocks across the available address space and across multiple runs. In particular, the proposed solution remains compatible to code signing, which is used on both iOS and Android platforms.

## 8.3 Summary

In the last years researchers and industry have analyzed malicious software on mobile platforms. Despite the fact that low infection rates of malware have been found by different studies, the research community has proven how sophisticated attacks are possible on current platforms. We also presented a large number of research proposals aimed at preventing or detecting mobile malware.

In the following chapters we will focus in detail on application phishing and application collusion attacks. Research as well as industry proposed countermeasures do not fully protect users against such attacks. We will evaluate available countermeasures and study in detail personalized security indicators to counter application phishing attacks. We will further analyze in detail overt and covert channels that can be implemented on today's platforms.

# Chapter 9
# Personalized Security Indicators against Phishing on Smartphone Applications

## 9.1 Introduction

Application phishing attacks in mobile platforms occur when malicious applications mimic the user interface (UI) of legitimate applications to steal user credentials. Phishing applications have been reported in the wild [100, 236, 291] with successful phishing attacks targeting thousands of users and procuring high revenues for the attackers [108]. Mobile phishing applications do not exploit system vulnerabilities [104]. They instead they use standard system features and APIs, and leverage the user's incapacity to distinguish the legitimate application from a phishing one.

Online services use *personalized security indicators* to aid the user in distinguishing the legitimate website from a phishing one [31, 254]. The personalized security indicator (or "indicator" from now on) is an image that the user chooses when he enrolls for the online service. After enrollment, the website displays the indicator every time the user logs in. The indicator allows the user to authenticate the website and the user should enter his credentials only if the website displays the correct indicator.

Mobile applications can also use indicators to mitigate application phishing attacks [34, 282]. The user chooses the indicator when he installs the application and must check that the application shows the correct indicator at each login. The indicator is stored by the application and the mobile OS prevents access from other applications.

In this chapter, we start by categorizing application phishing attacks in mobile platforms and possible countermeasures. We show that all known countermeasures incur a tradeoff between security, usability and deployability. The benefits of security indicators are that they can counter all the phishing attack types, and they can be easily deployed by service providers since they do not require changes to the mobile platform or to the marketplace infrastructure.

Personalized indicators, however, rely on the user to detect phishing by checking the presence of the correct indicator. Previous work in the context of websites has shown that users tend to ignore personalized indicators [171, 232] when entering their login credentials. Consequently, the research

community has deemed personalized indicators as an ineffective phishing detection mechanism [39, 149].

We revisit the question of personalized indicator effectiveness and evaluate them in the previously unexplored context of smartphone applications. These are becoming an increasingly ubiquitous method for accessing many security-critical services, such as e-banking, and, they therefore constitute an important attack vector. Over one week, 221 study participants used a banking application we developed on their own smartphones to complete various e-banking tasks. On the last day of the study, we launched a phishing attack. All study participants that did not use security indicators fell for the attack. In contrast, the attacks were successful for only 50% of the participants that used indicators.

While further studies are still needed to gain more confidence in the effectiveness of personalized security indicators, this first study on smartphones shows that indicators can be more effective than previously believed when deployed in a suitable context. We conclude that the research community should reconsider the reputation of indicators as an anti-phishing mechanism in new deployment models such as smartphone applications.

Finally, we look at the problem of setting up personalized security indicators. We assume that the mobile device can be infected by a malicious application at the time of indicator setup. This assumption is not far fetched. Users might have to select a personalized security indicator for a sensitive application (e.g., a mobile banking application) after a long time that they have been using their smartphones. We will present a solution that can be used to set up personalized security indicators securely even when the device has already been compromised.

To summarize, we make the following contributions:

- We analyze mobile application phishing attacks and possible countermeasures. We conclude that none of the countermeasures prevents all attacks and the problem of phishing remains largely unsolved.
- We report the results from a first user study that evaluates personalized indicators on smartphone applications. In our study, the deployment of indicators prevented half of the phishing attacks.
- We outline directions for further studies that are needed to better assess the effectiveness of indicators as an anti-phishing mechanism under various deployment models.
- We propose a solution to the problem of secure setup of personalized security indicators in the context of mobile applications.

## 9.2 Phishing Attacks and Countermeasures

In this section we categorize application phishing attacks on smartphones. All attacks are effective on Android and one of them also works for iOS. We discuss possible countermeasures and analyze them with respect to security, usability and deployment. Table 9.1 summarizes our analysis.

### 9.2.1 Phishing Attacks

**Similarity attack** The phishing application has a name, icon, and UI that are similar or identical to the legitimate application. The adversary must induce the user to install the phishing application in place of the legitimate one. Successful similarity attacks have been reported for Android [75, 100, 108, 236] and iOS [183].

**Forwarding attack** Another phishing technique is to exploit the application forwarding functionality of Android [104]. A malicious application prompts the user to share an event (e.g., a highscore in a game) on a social network and shows a button to start the social network application. When the user taps the button, the malicious application does not launch the social network application, but rather displays a phishing screen. The phishing screen asks the user to enter the credentials to access his account on the social network. Application forwarding is a common feature of Android and forwarding attacks may therefore be difficult for the user to detect.

**Background attack** The phishing application waits in the background and uses the Android ActivityManager, or a side-channel [176], to monitor other running applications. When the user starts the legitimate application, the phishing application activates itself in the foreground and displays a phishing screen [34, 104].

**Notification attack** The attacker shows a fake notification and asks the user to enter his credentials [282]. The notification window can be customized by the adversary to mimic the appearance of the legitimate application.

**Floating attack** The attacker leverages the Android feature that allows one application to draw an Activity on top of the application in the foreground. This feature is used by applications to always keep a window in

the foreground, for example, to display floating sticky notes. A phishing application that has the `SYSTEM_ALERT_WINDOW` permission can draw a transparent input field on top of the password input field of the legitimate application. The UI of the legitimate application remains visible to the user who has no means to detect the overlaid input field. When the user taps on the password field to enter his password, the focus is transferred to the phishing application which receives the password entered by the user.

### 9.2.2 Phishing Countermeasures

None of the attacks we discuss exploit OS vulnerabilities, but rather use standard Android features and APIs. Therefore, security mechanisms on the device (e.g., sandboxing or permission-based access control) or security screening run by the marketplace operator cannot prevent such attacks.

Similar to website phishing, thwarting application phishing attacks requires tailored security mechanisms. We describe possible countermeasures and categorize them in terms of security, usability and ease of deployment.

**Signature-based detection** Signature-based malware detection techniques that look for patterns of system calls and permissions can be implemented by the marketplace operator (e.g., the Google Bouncer system [124]). Recently, the authors of [34] developed a static analysis tool to detect the use of APIs that enable background attacks. The drawback of signature-based detection is that many phishing attacks (e.g., forwarding and similarity attacks) do not require specific API calls. This approach applies only to a subset of possible attacks.

**Name similarity** Marketplace operators can attempt to detect similarity attacks by searching for applications with similar names or icons. Since many legitimate applications have similar names or icons (e.g., banking applications for the same bank in different countries), this approach would produce a significant number of false positives. Detecting phishing applications in the marketplace does not rely on the user's alertness or change the user experience. Checking for phishing applications installed from the web or from third-party marketplaces (sideloading) could leverage the Google App Verification service [134].

| | Marketplace Phishing Detection | | | On-device Phishing Prevention | | | |
|---|---|---|---|---|---|---|---|
| | Signature-based detection | Name similarity | Visual similarity | Limited multi-tasking | Application name | Visual similarity | Personal indicator |
| **attacks** | | | | | | | |
| similarity attack | – | ✓ | – | – | – | ✓ | ✓ |
| forwarding attack | – | – | ✓ | – | ✓ | ✓ | ✓ |
| background attack | ✓ | – | ✓ | ✓ | ✓ | ✓ | ✓ |
| notification attack | ✓ | – | – | ✓ | ✓ | – | ✓ |
| floating attack | – | – | – | ✓ | ✓ | – | ✓ |
| **security** | | | | | | | |
| false positives/negatives | ✓ | ✓ | ✓ | – | – | ✓ | – |
| reliance on user alertness | – | – | – | – | ✓ | – | ✓ |
| **usability** | | | | | | | |
| user effort at installation | – | – | – | – | – | – | ✓ |
| user effort at runtime | – | – | – | – | ✓ | – | ✓ |
| restrictions on device functionality | – | – | – | ✓ | ✓[1] | – | – |
| significant performance overhead | – | – | – | – | – | ✓ | – |
| **deployment** | | | | | | | |
| changes to application provider (e.g., bank) | – | – | – | – | – | – | – |
| changes to marketplace | ✓ | ✓[2] | ✓[2] | – | – | – | – |
| changes to mobile OS | – | – | – | ✓ | ✓ | ✓ | – |
| changes to application | – | – | – | – | – | – | ✓ |

[1] restriction to full-screen applications with constant user interaction (Android Immersive mode)

[2] to check for phishing applications installed via sideloading

Table 9.1: Comparison of mechanisms to prevent application phishing attacks in mobile platforms.

**Visual similarity** The marketplace operator can attempt to mitigate background or forwarding attacks by searching for applications with similar UIs and, in particular, similar login screens. UI extraction and exploration are challenging problems and none of the known techniques provides full coverage [26]. Another option is to perform visual similarity comparisons directly on the device. In [184] the authors propose periodically taking screenshots and comparing them to the login screens of installed applications. While this solution does not incur the problem of UI extraction, it incurs a significant runtime overhead.

In general, if detection is based on matching UIs, phishing applications that use a slightly modified version of the legitimate application UI may go unnoticed. Finding an effective tradeoff (a similarity threshold) is a challenging task and is likely to include both false positives and negatives [184].

**Limited multi-tasking** Another approach to counter background or floating attacks is to limit multi-tasking on the device. The legitimate application can trigger a restricted mode of operation where no third-party applications can activate to the foreground. Multi-tasking can be re-enabled once the user explicitly terminates the application. Activation to the foreground can always be allowed for system services, to receive phone calls or SMS messages. This approach does not rely on the user's alertness but it requires changes to the OS and hinders the user experience. For example, a user cannot receive social network notifications while he is interacting with an application that disables multi-tasking.

**Application name** The mobile OS can show a status bar with the name of the application in the foreground [34, 238]. Phishing detection with this approach is effective only if the user is alert and the phishing application has a name and icon that are noticeably different from the ones of the legitimate application. This technique cannot address name similarity attacks. Furthermore, the status bar reduces the screen estate for applications that run in full-screen mode. An approach where the status bar appears only when the user interacts with the application is only practical for applications with low interaction, such as video players (Android *Lean Back* mode). For applications that require constant interaction, such as games (Android *Immersive* mode), forcing a visible status bar would hinder the user experience.

**Personalized indicator** When the application is installed, the user chooses an image from his photo gallery. When the application asks the user for

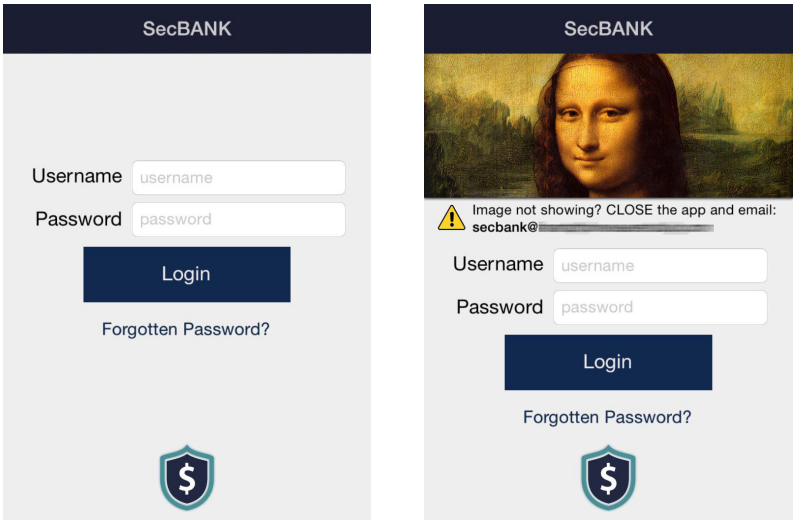
his credentials, it displays the image chosen by the user at installation time. An alert user can detect a phishing attack if the application asking for his credentials does not show the correct image. The mobile OS prevents other applications from reading the indicator of a particular application (through application-specific storage). This countermeasure can also mitigate floating attacks. In particular, the legitimate application can check if it is running in the foreground and remove the image when it detects that the application has lost focus (e.g., overriding the `onWindowFocusChanged()` method). Personal indicators can be easily deployed as they do not require changes to the OS or to the marketplace. However, they demand extra user effort at install time (because the user must choose the indicator) and used (because the user must check that the application displays the correct indicator).

**Summary** Our analysis is summarized in Table 9.1. All the countermeasures we discuss incur trade-offs in effectiveness, usability, and deployment. Personalized indicators can address all the attack types we consider and are easy to deploy as they do not require changes on the device or at the marketplace. However, personalized indicators rely on the user to detect phishing attacks.

Previous work [171, 232] has shown that most users ignore the absence of indicators when logging into an online banking service from a desktop web browser. These results contributed to the reputation of personalized indicators as a weak mechanism to detect phishing. Since smartphone applications are an increasingly important access method for many security-critical services such as e-banking; the user interface of mobile applications is different from those of standard websites; and personalized indicators have not been evaluated in the context of smartphone applications, we decided to assess their effectiveness as a detection mechanism for mobile application phishing attacks. Furthermore, while for a long time the research community considered browser warnings, based on old implementations, ineffective [73, 83, 248], a recent study on newer implementations showed the opposite [2]. We argue that it is important to re-evaluate the effectiveness of security mechanisms when their implementations or deployment models have changed significantly.

## 9.3 User Study

The goal of our user study was to evaluate the effectiveness of personalized indicators as a phishing-detection mechanism for mobile applications. We




(a) (b)

Figure 9.1: (a) SecBank application for the control group. The application did not use personalized indicators. (b) SecBank application for the three experimental groups. The application displayed the personalized indicator chosen by the user on the login screen (i.e., the Mona Lisa).

focused on a mobile banking scenario and implemented an application that allowed users to carry out e-banking tasks for a fictional bank called SecBank. As no bank currently uses indicators when the user performs a login operation, an evaluation in the context of a real deployment was not possible. The application had two components: a benign component that included the logic to log in and to carry out the banking tasks, and a malicious component that was in charge of carrying out the phishing attack. We used only one application to minimize the burden on the participants enrolling in our user study. That is, participants were asked to install only one application, rather than the banking application and another innocent-looking application that would perform the phishing attack.

In order to avoid participants focusing on the security aspects of the study, we advertised it as a user study to assess the usability of a mobile application (the full text of the advertisement e-mail can be found in Appendix E.1). We asked participants to install the SecBank application on their phones and provided them with login credentials (username and pass-

word) to access their accounts at SecBank. We assigned each participant to either a control group that used a SecBank application without personalized indicators (Figure 9.1(a)), or one of three experimental groups that used it with personalized indicators (Figure 9.1(b)). The experimental groups differed by the type of phishing attack. The user study lasted one week. During the first three days, we asked participants to carry out one e-banking task per day, in order to familiarize participants with the application. On the seventh day, we asked participants to perform a fourth e-banking task and, at this time, the malicious component of the application performed a phishing attack. We recorded whether participants entered their credentials while under attack.

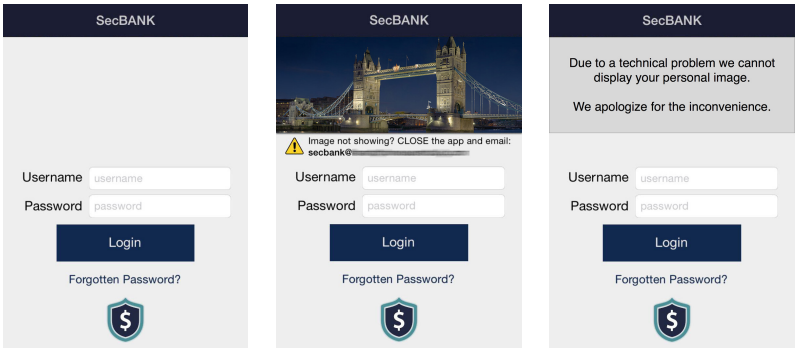
**Ethical guidelines** We informed the participants that the application would record their input and have access to the photo gallery on their phones. We further explained that the application would send no personal information to our servers. We collected the participants' email addresses to send them instructions on how to complete the e-banking tasks. The email addresses were deleted once the study was finished. At the end of the study, we briefed participants about the true purpose and the methodology of the study. We notified the ethical board of our institution which reviewed and approved our protocol before we started the user study.

### 9.3.1 Procedure

**Recruitment and group assignment** We recruited participants through an email sent to all people with an account at our institute (students, faculty and university staff). The study was advertised as a user study to "test the usability of a mobile banking application" without details of the real purpose of our design. We offered a compensation of CHF 20.- to all participants who completed the pre-test questionnaire.

We received 465 emails from potential participants to whom we replied with a link to an online pre-test questionnaire designed to collect email addresses and demographic information. 301 participants filled in the pre-test questionnaire. We assigned them to the following four groups (one control group and three experimental groups) in a round-robin fashion:

- **Control Group (A):** The application used by this group did not use personalized indicators. On the last day of the user study, the malicious component of the application showed a clone of the SecBank login screen, leaving the user with no visual clues to identify the phishing attack.




(a) (b) (c)

Figure 9.2: (a) Missing-Image attack: The application does not show the indicator chosen by the user. (b) Random-Image attack: The application shows a random image from the local photo gallery (e.g., the Tower Bridge of London). (c) Maintenance attack: The application shows a message explaining that the indicator cannot be displayed due to technical reasons.

- **Missing-Image Group (B):** The application used by this group supported personalized indicators. On the last day of the user study, the malicious component of the application performed a phishing attack and showed the SecBank login screen without the indicator (Figure 9.2(a)).

- **Random-Image Group (C):** The application used by this group supported personalized indicators. On the last day of the user study, the malicious component of the application performed a phishing attack and showed the SecBank login screen with a photo randomly chosen from the local photo gallery. The photo displayed was different from the one chosen by the user as the personalized indicator (Figure 9.2(b)).

- **Maintenance Group (D):** The application used by this group supported personalized indicators. On the last day of the user study, the malicious component of the application performed a phishing attack and showed the SecBank login screen with an "under maintenance" notification in place of the indicator chosen by the user (Figure 9.2(c)).

We sent an email to all participants who completed the pre-test questionnaire with a link to a webpage from which they could install the SecBank application (the full text of this e-mail can be found in Appendix E.2). Participants in the Control Group (A) were directed to a webpage where we only explained how to install the application. Participants in experimental groups B, C, and D were directed to a webpage that also explained the concept of personalized indicators. The webpage advised that participants should not enter their login credentials if the application was not showing the correct indicator (the text of the webpage can be found in Appendix F). The instructions were similar to the ones used in banking websites that deploy indicators [31, 254]. 276 participants visited the webpages and installed the SecBank application on their devices. After installation, the SecBank application for groups B, C, and D showed explanatory overlays to guide participants in choosing a personalized indicator from their photo gallery.

**Tasks** The study lasted one week. Participants were asked to perform four e-banking tasks on days 1, 2, 3, and 7. We sent instructions via email and asked participants to complete the task within 24 hours (The e-mail text of the first task can be found in Appendix E.3. The following tasks had similar e-mail structure and text.). The tasks were the following:

- **Task 1** (Day 1): "Transfer $200 to Anna Smith."
- **Task 2** (Day 2): Download the bank statement from the "Account Overview" tab.
- **Task 3** (Day 3): Activate the credit card from the "Cards" tab.
- **Task 4** (Day 7): Transfer $100 to "George White".

The goal of tasks 1–3 was to help participants to become familiar with the SecBank application. We sent the instructions to perform the last task four days after (including a weekend) the completion of task 3. During this last task, the malicious component of the application performed a phishing attack on all participants. Participants in the Control Group (A) saw a login screen that matched that of their SecBank application. Participants in the Missing-Image Group (B) saw a login screen similar to the one of SecBank, but without any personalized indicator (Figure 9.2(a)). Participants in the Random-Image Group (C) saw a login screen similar to SecBank, but with a random image from their photo gallery (e.g., the Tower Bridge as shown

in Figure 9.2(b)). Finally, participants in the Maintenance Group (D) saw a message explaining that for technical problems the indicator could not be displayed (Figure 9.2(c)).

| Gender | |
|---|---|
| Male | 150 (68%) |
| Female | 71 (32%) |
| Age | |
| Up to 20 | 43 (20%) |
| 21 – 30 | 164 (74%) |
| 31 – 40 | 9 (4%) |
| 41 – 50 | 3 (1%) |
| 51 – 60 | 0 (0%) |
| Over 60 | 2 (1%) |
| Use smartphone to read emails | |
| Yes | 214 (97%) |
| No | 7 (3%) |
| Use smartphone for social networks | |
| Yes | 218 (99%) |
| No | 3 (1%) |
| Use smartphone for e-banking | |
| Yes | 97 (44%) |
| No | 124 (56%) |

Table 9.2: Demographic information of the 221 participants that completed all tasks.

### 9.3.2 Results

Out of 276 participants that installed the application, 221 completed all tasks. We provide their demographics and other information collected during the pre-test questionnaire in Table 9.2. The majority of the participants were male (68%) and thirty years old or younger (94%). Most participants used their smartphone to read emails (97%) and to access social networks (99%). Slightly less than half of the participants (44%) used their smartphones for mobile banking.

The 221 participants that completed all tasks were distributed as follows: 56 in the Control Group (A), 55 in the Missing-Image Group (B), 56 in the Random-Image Group (C), and 54 in the Maintenance Group (D).[1]

[1]We note that, by chance, the participants that dropped out of the study were almost evenly distributed among the four groups.

| | Attack not successful | Attack successful |
|---|---|---|
| Control Group (A) | 0 (0%) | 56 (100%) |
| Missing-Image Group (B) | 30 (55%) | 25 (45%) |
| Random-Image Group (C) | 23 (41%) | 33 (59%) |
| Maintenance Group (D) | 29 (54%) | 25 (46%) |
| Experimental groups combined | 82 (50%) | 83 (50%) |

Table 9.3: Success rate of the phishing attack.

**Indicator effectiveness** Table 9.3 shows the success rates for the phishing attack during Task 4. All of the 56 participants in the Control Group (A) fell for the attack (i.e., all participants entered their login credentials to the phishing application). 83 out of 165 (50%) attacks in the experimental groups B, C, and D were successful.

To analyze the statistical significance of these results we used the following null hypothesis: "there will be no difference in the attack success rate between users that use personalized indicators and users that do not use personalized indicators". A Chi-square test showed that the difference was statistically significant ($\chi^2(3, N = 221) = 46.96, p < 0.0001$) and thus the null hypothesis can be rejected. We conclude that, in our user study, the deployment of security indicators decreased the attack success rate and improved phishing detection.

**Difference between attacks** A closer look at the performance of participants in groups B, C, and D reveals that: 30 out of 55 participants in the Missing-Image Group (B), 23 out of 56 participants in the Random-Image Group (C), and 29 out of 54 participants in the Maintenance Group (D) retained from logging in.

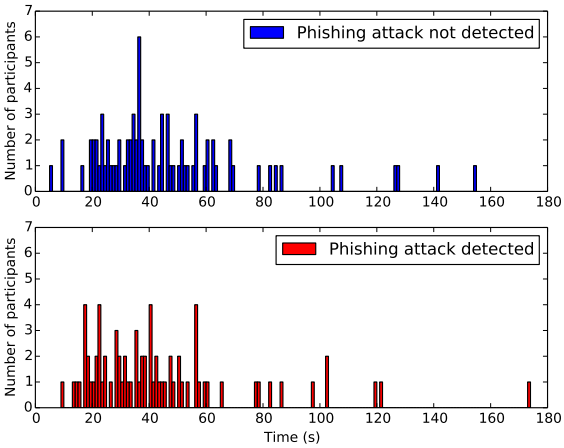
To analyze the success rates of the different attack types we used the following null hypothesis: "the three attack types we tested are equally successful". A Chi-squared test showed no statistically significant difference in the attack success rates, and thus we fail to reject the null hypothesis ($\chi^2(2, N = 165) = 2.53, p = 0.282$). We conclude that in our test setup the three attacks were equally successful and, therefore, indicators worked as well in all three scenarios.

| | Attack not successful | Attack successful |
|---|---|---|
| Gender | | |
| Male | 59 (52%) | 54 (48%) |
| Female | 23 (44%) | 28 (56%) |
| Age | | |
| Up to 20 | 15 (43%) | 20 (57%) |
| 21 – 30 | 57 (48%) | 61 (52%) |
| 31 – 40 | 6 (86%) | 1 (14%) |
| 41 – 50 | 2 (67%) | 1 (33%) |
| 51 – 60 | 0 (0%) | 0 (0%) |
| Over 60 | 2 (100%) | 0 (0%) |
| Use smartphone for e-banking | | |
| Yes | 41 (54%) | 35 (46%) |
| No | 41 (46%) | 48 (54%) |
| Smartphone display size (diagonal) | | |
| up to 4in | 28 (58%) | 20 (42%) |
| from 4in to 4.5in | 44 (45%) | 54 (55%) |
| from 4.6in to 5in | 10 (53%) | 9 (47%) |

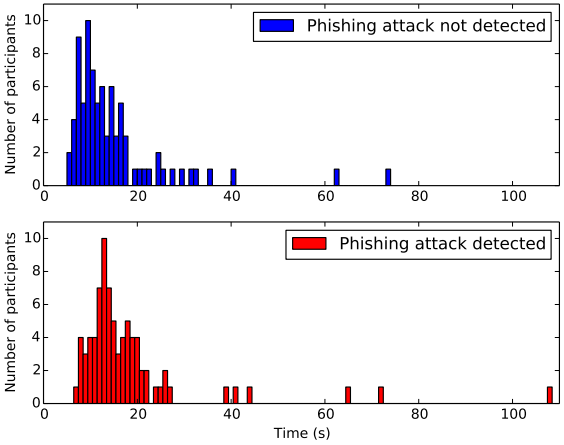
Table 9.4: Success rate of the phishing attack in relation to gender, age, familiarity with mobile banking, and smartphone display size.

**Other factors** We performed post hoc analysis of our dataset to understand if there were any relationship between the attack success rate and gender ($\chi^2(1, N = 221) = 0.99, p = 0.319$), age group ($\chi^2(4, N = 221) = 8.36, p = 0.079$), smartphone display size ($\chi^2(2, N = 221) = 5.40, p = 0.369$) or familiarity with mobile banking ($\chi^2(1, N = 221) = 1.98, p = 0.160$). We did not find any statistical significance for any of the factors we considered. Table 9.4 provides the results break-down.

Finally, we report the mean time spent by participants setting up the personalized indicator or logging in. The mean time spent setting up the indicator for participants that did not fall victim to the attack was 43s (±28s); the mean time for participants that fell for the attack was 46s (±28s). The mean time spent on the login screen for participants that did not fall victim to the attack was 18s (±14s); the mean time for participants that fell for the attack was 14s (±10s). The distribution of the times spent while setting up the indicator and while logging in are shown in Figure 9.3(a) and Figure 9.3(b), respectively.




(a) Personal indicator setup




(b) Login screen

Figure 9.3: Distribution of the time spent by participants setting up the personalized indicator (a) and logging into the SecBank application (b).

### 9.3.3 Post-test questionnaire

Participants that completed all tasks received a post-test questionnaire with the following items:

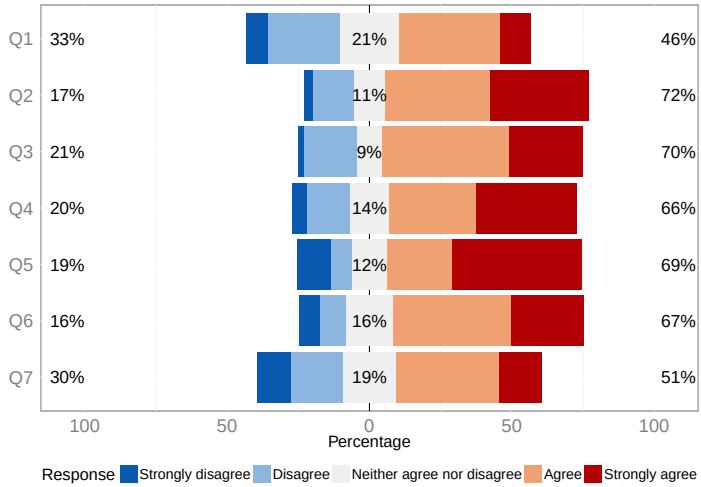



Figure 9.4: Answers to the post-test questionnaire. Items Q1–Q4 were answered by all participants. Items Q5–Q7 were answered only by participants in the experimental groups (B, C, and D). Percentages on the left side include participants that answered "Strongly disagree" or "Disagree". Percentages in the middle account for participants that answered "Nor agree, nor disagree". Percentages on the right side include participants that answered "Agree" or "Strongly agree".

Q1 *I am concerned about malware on my smartphone*
Q2 *I am concerned about the secrecy of my password for mobile banking*
Q3 *I am concerned about the secrecy of my password for my email account*
Q4 *I am more concerned about the secrecy of my mobile banking password compared to my email password*
Q5 *During the study, I have made sure that my personal image was showing before entering my username and password*
Q6 *The login mechanism using personal images was intuitive and user-friendly*
Q7 *I would use personal images in other applications*

All participants answered items Q1–Q4. Figure 9.4 shows their answers on 5-point Likert-scales. 33% of the participants were not concerned, 21% had neutral feelings, and 46% were concerned about malware on their smartphones (Q1). Most participants reported that they were concerned

about password secrecy (72% agree or strongly agree for their e-banking passwords (Q2) and 70% agree ir strongly agree for their email passwords (Q3)). Participants were more concerned (66% agreed or strongly agreed) about the secrecy of their e-banking password than that of their email passwords (Q4).

Participants in groups B, C, and D additionally answered items Q5–Q7. 69% of the participants reported that they checked the presence of the indicator at each login (Q5). 67% of the participants found personalized indicators user-friendly (Q6), and 51% of them said they would use personalized indicators in other applications (Q7). Additionally, we asked participants if they were familiar with security indicators prior to our study and only 19% replied that they were.

We also asked participants in the experimental groups if they had noticed anything unusual when logging in to complete Task 4. 23% of the participants did not notice anything unusual, while 23% did not remember. 54% of the participants noticed something was wrong with the SecBank application while they were logging in. To those participants that noticed something wrong with the application, we also asked if they logged in and why (answer to this question was not mandatory.) 36% of them reported that they logged in and the reasons they provided mainly fell in two categories. Some users reported that they logged in because they were role-playing and did not have anything to lose. We list some of their answers below:

> "Because it is a test and I had nothing to lose or hide."

> "This should be a test and I thought that it was safe."

> "I knew this was a test product so there could not possibly be malware for it already. I just thought maybe you guys had some difficulties going on."

Some users from the Maintenance Group (group D) reported that they logged in because they thought there was a temporary bug in the application. This behaviour suggests that users expect bugs in IT systems and, therefore, they are susceptible to attacks that leverage the unreliable view that people have of computer products. We list some of participants' answers below:

> "I thought it was a problem of the app that the image *was* there but just did not load. As it happens sometimes in Safari or other browsers."

"I thought it was a temporary bug."

"I thought that it was a system error."

## 9.4 Discussion

In our study, the deployment of security indicators prevented half of the attacks. Our user study shows a significant improvement in the attack detection rate (50%), compared to previous studies in the context of website phishing attacks (4% in [232] and 27% in [171]). The purpose of our study was not to reproduce these previous studies but rather to evaluate security indicators in the context of smartphone applications as realistically as possible. Below we discuss how our results should be interpreted in comparison to previous related studies and outline directions for further studies that are needed to gain better confidence on the effectiveness of personalized indicators.

**Mobile application context** Mobile user interfaces are considerably simpler than the ones of websites designed for PC platforms. As the user's focus is limited to a few visual elements [224], personalized indicators may be more salient in mobile application UIs. Also the usage patterns of mobile applications may be different from those of websites, which may improve the detection of incorrect or missing UI elements. These reasons may explain why the attack detection rate in our study was higher than the one found in previous studies that focused on web phishing on PC platforms [171, 232].

**Role-playing** When designing our user study we kept the experiment as close to a real world deployment as possible. We asked participants to install the application on their phones and avoided using web platforms for human intelligence tasks like Amazon Mechanical Turk (for example, used in [171]). However, we could not leverage a real bank deployment and its user base (as in [232]) because, to the best of our knowledge, no bank is currently using personalized indicators in its mobile banking application.

Previous work has shown that role-playing negatively impacts the effect of security mechanisms [232]. The responses to the post-test questionnaire give reasons to believe that, due to role-playing, some participants may have logged in despite detecting the attack. It is likely that role-playing increased the attack success rates in our study. A user study run in cooperation with a banking institution willing to deploy personalized indicators would yield more accurate results.

**Duration and security priming** In our study, the phishing attack happened seven days after the study participants had been primed about security and our study did not evaluate participants' behavior at a later point in time. This study setup is similar to [171] where 5 days passed between priming and the attack. In contrast, Schechter et al. [232] recruited customers of a real bank that had been primed at the time they had opened their bank account (possibly long before they took part in the user study and the attack was tested). It is likely that compared to a real-world deployment, the recent security priming decreased the attack success rates. Long-term studies are needed to evaluate the effect of time between the security priming and the attack.

**Population sample** Participants were recruited within our institution across students, faculty and staff. Most participants were male (68%) and below 30 years old (94%). While our institution attracts people from around the world, the large majority of the participants were Swiss nationals. Further studies are required to assess whether our results generalize to different populations (e.g., with different age intervals, nationalities, etc.).

We did not ask participants whether they knew other participants and whether they had discussed the study. While participants may influence each other's behavior, we could not identify any particular relationship or cliques among participants.

**Application deployment** In our study, we distributed the victim application (e-banking app) and the phishing component in a single application, rather than using one victim application and a second application to launch the attack. Since the phishing attack was not launched from a separate application, our study did not evaluate whether participants could detect the attack by UI lag when the phishing application gains control of the device screen.

The motivation behind this study design choice was two-fold. First, we minimized the participant burden during enrollment. Participants were asked to install only the SecBank application, rather than the banking application and a second innocent-looking application that would launch the attack. If participants had had to install a second application (e.g., a weather forecast application) they may have become suspicious about its purpose. Second, previous work has shown that users tend to disregard slight animation effects when the phishing application gains control of the device screen [34]. Due to this design choice, if a study participant decided

to examine the list of running background apps before entering his login credentials, our attack component would not have been visible on this list, and thus such defensive measures are not applicable to our study.

**Recruitment and task perception** A common challenge in designing security-related user studies is to avoid drawing participants' attention to the security aspects under evaluation. If participants are focused on security, and hence more attentive to possible threats, the study results would say little about real-world users to whom security is typically not the primary goal [84, 232]. As our goal was to assess the effectiveness of a security mechanism that has not yet been deployed in the context of smartphone applications, we could not avoid minimal security priming of the participants.

We advertised our study as one on "the usability of a mobile banking application". Similarly, the emails sent to complete the tasks were solely focused on task completion. We cannot verify if some participants discovered the true goal of our study before we revealed it. However, the comments that participants entered in the post-test questionnaire suggest that many participants focused on the usability of the application. We report some comments we received:

> "The tasks were easy to perform, but it remained unclear for me what you were exactly testing."

> "App easy to navigate and user-friendly."

> "The user interface was not so intuitive due to the lack of spaces between buttons and the equality of all interface options/buttons."

**Attack implementation** In the phishing attacks where the UI showed no indicator (group B) or where it showed a maintenance message (group D), we removed the text that asked users to email the bank in case of a missing indicator. We kept that text in the attack that showed a random image (group C). The UI elements shown by the phishing application might have influenced the reaction of the participants and their willingness to enter their credentials. We did not test how changes to the text or to other UI elements affect phishing detection. A potential direction for future studies is to understand how users react to small changes to the UI of an application.

**Indicator placement and size** The SecBank application showed the personalized indicator right above the username and password fields, taking up roughly one third of the screen. The size and the placement of the personalized indicator within the UI may have an impact on the attack detection rate. In the context of websites designed for PC platforms, Lee et al. [171] show that the size of the indicator does not change the effectiveness of personalized indicators as an phishing-detection mechanism. An interesting direction for future work would be to look at alternative types of indicators (e.g., interactive ones) and compare them to the ones used in this work.

## 9.5 Deployment

**Application and infrastructure changes** From the point of view of a service provider, personalized indicators can be easily deployed because they require no changes to the marketplace or to the mobile OS. Introducing personalized indicators only requires a software update of the client application (application updates are frequent throughout an application lifecycle) and no changes to the server-side infrastructure of the application provider (i.e., the bank). The mobile application may guide the user through the indicator setup. Other solutions, as those presented earlier on, require either changes to the mobile OS or to the marketplace infrastructure. A service provider (e.g., a bank) can therefore adopt this security mechanism independently of other service providers or of the mobile platform provider.

**Indicator choice and reuse** Personalized indicators may be used for phishing detection by security-critical applications. If indicators are adopted by multiple applications, users might tend to reuse the same indicator across different applications. This behaviour may provide an attack vector where the attacker develops an application that requires personalized indicators, and hopes that the victim user chooses the same indicator that he had chosen for his banking application. The problem of reusing personalized indicators across applications is comparable to the problem of reusing passwords across online services. We note that the deployment of personalized indicators would most likely be limited to few security-critical services, while users often have to manage passwords for a large number of services and websites.

Similar to password reuse scenarios, users might choose different personalized indicators for “categories” of applications. That is, a particular picture for security critical applications (e.g., banking, email) and another

picture for less critical applications (e.g., social networks). Furthermore, when users are asked to pick a personalized indicator, they might choose among the pictures that are at the top of the list (e.g., the ones that were most recently added to the photo gallery). Therefore, the probability that a picture is selected as the personalized indicator may not be uniform across all pictures in the photo gallery.

Since in our study we did not collect information on the indicators chosen by the participants, further studies are required to explore users' behavior and patterns in choosing personalized indicators.

## 9.6 Secure Setup of Application Indicators

So far we have presented evidence that personalized indicators can help users to detect application phishing attacks. In this section we focus on the setup of personalized indicators and propose a secure protocol to bootstrap indicators in mobile platforms.

Setup of indicators in previous research proposals [72, 260, 282] and deployed systems [31, 254] relies on the "trust on first use" (TOFU) assumption. The indicator setup happens the first time the user registers for an online service [31, 72, 254, 260] or starts the application [282], assuming that the adversary is not present at this time. Otherwise, if the indicator is phished during its setup, malicious software can later on masquerade as the legitimate application.

In the rest of this section we propose a protocol to setup indicators that does not rely on the TOFU assumption and, therefore, can withstand phishing attacks when the user chooses the indicator. As a use case, we consider a mobile banking scenario similar to the one of Section 9.3, and consider an application that supports indicators to improve phishing detection. We start by detailing the system and adversarial model we consider. Later on, we describe the indicator setup protocol and present an implementation for the Android platform. Finally, we show the results of both a performance and a small-scale usability study.

### 9.6.1 System and Adversary Model

Figure 9.5 illustrates the system model we consider. (Gray components are part of our solution and are detailed below.) Mobile applications run on top of an OS that, together with the underlying hardware, constitute the device TCB. The TCB guarantees that a malicious application cannot tamper with the execution of another application (isolated execution) or read its data (application-specific storage). The application active in foreground controls

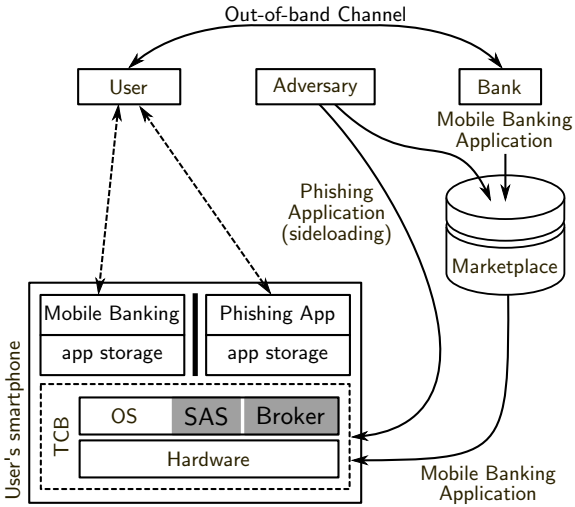



Figure 9.5: System model overview. Each application running on the smartphone is sandboxed and cannot access the memory or storage of other applications. Applications are installed from the marketplace or via sideloading. The bank has an out-of-band channel (e.g., regular mail) to its customers. Our protocol adds two components to the platform TCB (shown in gray). The Broker can measure installed applications and has a UI to receive user inputs. A Secure Attention Sequence (SAS) is used to start the Broker.

the screen and receives user inputs. The TCB enforces that applications running in the background do not intercept user inputs intended for the foreground application and cannot capture its output (user interaction isolation).

Since we consider a mobile banking scenario, we assume the existence an out-of-band-channel between the bank (i.e., the application provider) and the user. An example of this channel is the (regular) mail system routinely used by banks as a secure and authentic channel to transfer PINs and other credentials to their customers. As currently used to reset forgotten passwords, emails can also be used as an alternative out-of-band channel.

The goal of the adversary is to phish the indicator that the user chooses for the banking application. We assume that the user has previously installed, either from the marketplace or via sideloading, a malicious appli-

cation on his smartphone. Leveraging the malicious application on the user's phone, the adversary can launch any of the attacks presented in Section 9.2.1. However, the adversary cannot compromise the device TCB or the out-of-band channel between the bank and the user.

### 9.6.2 Secure Indicator Setup Protocol

Setting up an indicator in the presence of malicious software requires the user to identify the banking application for which he is choosing the indicator. A similarity attack to phish the indicator at setup time may be hard for the user to detect. Similarly, the marketplace operator may fail to identify phishing applications and block their distribution (see Section 9.2). We argue that no party but the bank can attest the validity of the banking application for which the user is about to choose the indicator. For this reason, our protocol relies on a trusted component of the mobile platform that, together with the bank, attests the validity of the banking application installed on the device.

In particular, the trusted component and the bank establish an authentic channel to attest the application. If attestation is successful, the trusted component provides the application with a PIN known by the user. The user can identify the legitimate application, if it shows the correct PIN.

Figure 9.5 shows in gray the components that we add to the device TCB. A system component that we call the Broker can measure applications (e.g., compute a hash of the application binary and its configuration files) installed on the device and has a UI to receive user inputs. The mobile OS is also enhanced with a Secure Attention Sequence (SAS), which is a common approach to start a particular software component of a TCB [119, 175, 190][2]. We implement the SAS operation as two repeated presses of the home button and we use it to start the Broker. When the Broker is running, the mobile OS ensures that no background application can activate to the foreground.

The bank and the Broker establish an authentic channel using the out-of-band channel between the bank and the user. The bank sends a measurement of the legitimate banking application over the authentic channel, so that the Broker can compare it with the measurement of the banking application installed on the device. If the two measurements match, the Broker transfers to the banking application a PIN known by the user. The banking application shows the PIN to the user who can, therefore, identify the legitimate banking application.

[2] A popular SAS is the "ctrl-alt-del" sequence in Windows systems which generates a non-maskable interrupt that starts the user-logon process.

Figure 9.6 illustrates the steps to securely setup a personalized indicator. Here we explain them in detail.

1. The bank uses the out-of-band channel (e.g., regular mail) to send a PIN to the user.

2. The user installs the application, downloading it from the marketplace. When the installation completes, the user performs the SAS operation to start the Broker. While the Broker is running, the mobile OS prevents third-party applications from activating to the foreground.

3. The user inputs the PIN to the Broker.

4. The Broker and the bank use the PIN to run a Password Authenticated Key Exchange (PAKE) protocol [155] and establish a shared key.

5. The bank sends the measurement of the legitimate banking application to the Broker. The measurement can be the hash of the application installation package. The message is authenticated using the key established during the previous step.

6. The Broker verifies the authenticity of the received message, measures the banking application on the device, and checks its measurement against the reference value received from the bank.

7. If the two measurements are identical, the Broker securely transfers the PIN to the banking application (e.g., writes the PIN to the application-specific storage). Otherwise the Broker aborts the protocol and notifies the user.

8. The Broker starts the banking application and the mobile OS restores the functionality that allows background applications to activate to the foreground. The banking application displays the PIN which serves as a confirmation to the user that the application in foreground is the legitimate banking application.

9. The user identifies the application in foreground as the legitimate banking application if it displays the same PIN that the user has received from the bank. At this point, the user can choose a personalized indicator for the banking application.

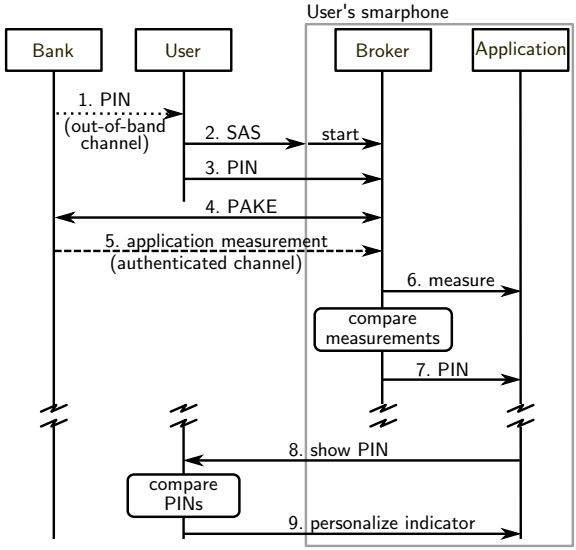



Figure 9.6: The personalized indicator setup protocol. The user performs an SAS operation to start the Broker and enters the PIN. The Broker verifies the installed banking application with the help of the bank. If the installed application is the legitimate one, the Broker starts it and the user can choose a custom indicator.

**Security Analysis.** Our protocol relies on the user attention when setting up the indicator. At the beginning of the setup protocol, the user must input the PIN only after performing the SAS operation. At the end of the protocol, the user must choose an indicator only if the application in foreground displays the same PIN that the user has received from the bank.

The SAS operation and the device TCB guarantee that no information is leaked when the user inputs the PIN to the Broker. In particular, no malicious application can masquerade as the Broker to phish the PIN. The PIN is used as a one-time password to derive a key through the PAKE protocol. The derived key is only used to authenticate one message (from the bank to the Broker). The security properties of the PAKE protocol guarantee that, given a transcript of the protocol, the adversary can neither learn the PIN, nor brute-force the set of possible PINs [182].

The application-specific storage functionality, provided by the mobile OS, guarantees that the PIN received by the banking application can not be read by other applications.

A phishing application on the device will not receive the PIN from the Broker because its measurement differs from the one of the legitimate banking application. The adversary cannot impersonate the bank to the Broker without knowledge of either the PIN or the key derived through the PAKE protocol.

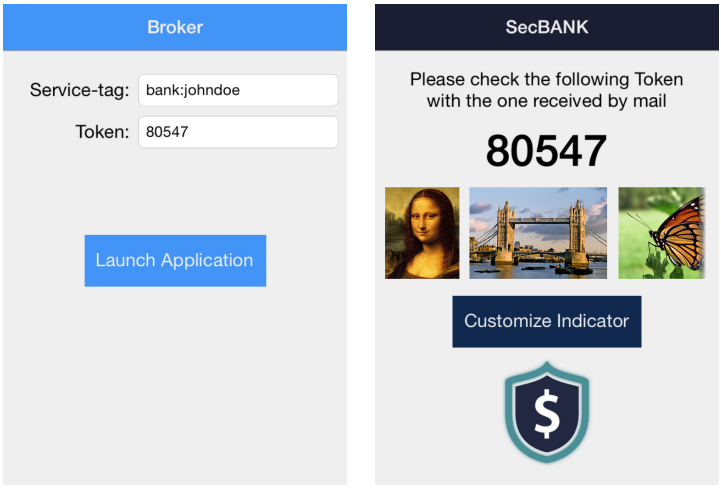
### 9.6.3 Implementation

We build a prototype of our setup protocol for the Android platform. We use a Samsung Galaxy S3 and develop against the CyanogenMod Android OS (version 4.3 "JellyBean"). Cryptographic operations are based on the Bouncy Castle library [173]. Message authentication uses HMAC-SHA256 with a 128-bit key. The bank's server is implemented in Python, using the CherryPy Web Framework and SQLite. Client-server communication works over a standard HTTP channel using messages in the JSON standard format. Our implementation adds two Java files, to implement the Broker as a privileged application, and modifies four system Java files. A total of 652 lines of code were added to the system TCB.

We add an extra tag (`<secureapk>`) to the Android Manifest file of the banking application. The tag contains two fields indicating a URL and an application handle. The Broker uses the URL to contact the application provider (i.e., the bank) and the handle to identify the application to attest. When the banking application is installed, the `PackageParser` of the Android OS reads the extra information in the `<secureapk>` tag and stores it for later use.

We assign the SAS operation to a double-tap on the home button. The SAS operation unconditionally starts the Broker that asks the user to enter the PIN received from the bank (see Figure 9.7(a)). In our implementation, the bank sends to the user a 5-digits PIN (e.g., "80547") and a *service tag*. The service tag contains an application handle used by the Broker to search through the registered handles and to identify the application to attest. The service tag also contains a user ID, sent to the bank by the Broker, to retrieve the correct PIN from its database. An example of a service tag is "bank:johndoe", where "bank" is the application handle and "johndoe" is the user ID. After the user has input the service tag and the PIN, the Broker uses the handle to identify the application to attest. At this time the Broker also fetches the URL stored by the `PackageParser`, to contact the bank's server.

We use SPEKE [155] as an instantiation of the key establishment protocol. SPEKE uses 4 messages, including a key confirmation step. The first SPEKE message sent by the Broker also contains the user ID that allows the




(a) (b)

Figure 9.7: (a) Screenshot of the Broker prototype. The user inputs the service tag and the PIN received by the bank. (b) Banking application started by the Broker. The user must verify that the PIN displayed matches the one received by the bank. The application shows the images in the local photogallery and the user can select one as the indicator.

bank's server to select the correct PIN from its database. The server uses the key established via the PAKE protocol to compute a MAC over the hash of the legitimate application. Both the hash and the authentication tag are sent to the Broker. The Broker verifies the authentication tag, hashes the installed application, and compares the hash computed locally with the received one. If the two hashes match, the Broker writes the PIN to the application's folder so that it can only be read by that application. Otherwise, the Broker aborts and notifies the user.

Figure 9.7(b) shows the banking application started by the Broker. The user is asked to compare the displayed PIN with the one received via mail and choose a personalized indicator.

**Evaluation.** We evaluate the setup protocol using a a sample banking application of 264KB. Client-server communication overhead totals to 705 bytes, of which 641 bytes account for the SPEKE protocol. (Communication overhead is independent of the size of the application.) We test the protocol

| TCB increment | 652 LoC |
|---|---|
| Communication overhead | 705 bytes |
| Execution time (WiFi) | 421ms (±21ms) |
| Execution time (3G) | 2042ms (±234ms) |
| Execution time (Edge) | 2957ms (±303ms) |

Table 9.5: Evaluation summary for the personalized indicator setup prototype.

over a WiFi connection (hosting the server in a remote location, i.e., not on a local network), as well as over 3G and EDGE cellular data connections. Table 9.5 summarizes the evaluation in terms of added lines of code, communication overhead, and execution time. We note that hashing the banking application takes 25ms on average, with a standard deviation of 2ms. The time to hash an application increases linearly with the size of the application and remains lower than the network delay for applications up to 100MB.

### 9.6.4 Usability

We run a small-scale user study to evaluate the usability of our setup protocol. In particular, our goal was to understand how easy it is for users to setup indicators by following written instructions, like the ones a bank customer would receive from his bank. We also wanted to verify how attentive users are when comparing the PIN shown by the application with the one received from the bank.

We invited participants to our lab and asked them to carry out the setup protocol as if they were customers of a bank that uses indicators in its application. We provided participants with a phone and a letter from the bank with instructions on how to setup the indicator. A sample letter can be found in Appendix H We assigned participants to three different groups. One group carried out the set up protocol in benign settings. For the remaining two groups, we simulated variants of a background attack at the time when the banking application displays the PIN (step 8 of the setup protocol). We recorded whether participants pressed the “Customize Indicator” button and chose a personalized indicator while under attack.

**Procedure.** We advertised the study as a user study “to evaluate the usability of a setup protocol for an e-banking application”. Participants were asked to come to our lab and setup the application on a smartphone

that we provided. We informed participants that we would not collect any personal information and offered a compensation of CHF 20.-.

We selected 30 participants and assigned them to one of three groups in a round-robin fashion. (None of these participants were involved in the user study of Section 9.3.) The difference among the groups was the PIN shown by the banking application right before participants were asked to choose a personalized indicator. Group A participants were shown the same PIN that appeared on the letter from the bank. Group B participants were shown a random PIN. Group C participants where shown no PIN.

Participants of group A reflected the user experience of the setup protocol under no attack. Participants of groups B and C reflected the user experience of the setup protocol under a background attack.

*Experiment.* The experiment started with a pre-test questionnaire to collect demographic information. After the questionnaire, participants were given a smartphone and a letter from a fictitious bank with instructions to setup the indicator. The letter included a 5-digit PIN and a service tag to be used during the setup protocol. The procedure was the one described in Section 9.6.2 and was carried out on a Galaxy S3. We stress that we did not interact with the participants for the duration of the setup protocol. Participants were also asked to fill in a post-test questionnaire to evaluate the procedure they had just completed.

**Results.** 37% participants were males and 63% were females. Most of them had completed a university degree (73%), and were aged between 24 and 35 (80%).

All participants in group A managed to complete the task successfully. All participants of group B aborted the setup procedure because they detected a mismatch between the PIN displayed by the banking application and the one in the letter from the bank. 40% percent of the participants in group C failed to detect the missing PIN and completed the setup process, thereby leaking the indicator to the phishing application.

The post-test questionnaire revealed that 91% of the participants rated the instruction sheet easy to understand (Q1) and 94% of them rated the task easy to complete (Q2). 85% of the participants believed that they had completed the task successfully (Q3) and 67% of them would use the mechanism in a mobile banking application (Q4). Figure 9.8 shows the distribution of the answers. Appendix G provides the full text of items Q1–Q4.

Our study suggests that the setup procedure was simple enough to be completed by average users. Regarding attack detection, the “missing

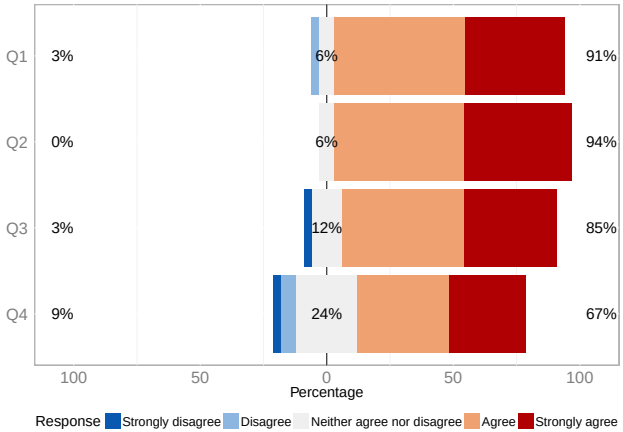



Figure 9.8: Answers to the post-test questionnaire for the user study on the indicator setup protocol. Percentages on the left side include participants that answered “Strongly disagree” or “Disagree”. Percentages in the middle account for participants that answered “Nor agree, nor disagree”. Percentages on the right side include participants that answered “Agree” or “Strongly agree”.

PIN” attack was the only one that went undetected. Participants may have judged the absence of the PIN as a temporary bug and decided to continue with the setup protocol. Similarly to what we have reported in Section 9.4, this result shows that users are acquainted with software bugs and are likely to tolerate a temporary bug even during a security-sensitive operation. Nevertheless, the user study reveals that our solution is usable and effective.

## 9.7 Related Work

In chapter 8 we discussed smartphone malware attacks and countermeasures in general. In the following we review related work that looks at phishing attacks, in particular.

A systematic evaluation of application phishing attacks was recently provided in [34]. The authors use static analysis to detect applications that use APIs that enable certain classes of application phishing attacks. They also introduce an on-device solution that allows users to identify applications with which they are interacting. In the proposed solution, the

OS displays a status bar that shows the application and developer names together with an image chosen by the user. The image, therefore, is used by the user to tell the authentic status bar managed by the OS from a fake status bar that a phishing application can show if it gains control of the entire screen. Compared to personalized indicators, the proposed solution incurs more deployment costs, since it requires changes to the OS and the marketplace. The authors of [34] also use Amazon Mechanical Turk to run a user study with 304 participants, and assess the effectiveness of phishing attacks in mobile platforms. The user study corroborates our findings on personalized indicators, although the authors placed the image in the navigation bar rather than in the application itself. Furthermore, the user study in [34] was a one-off test that did not last for a week and, compared to ours, let participants interact with an emulated android device through a web-browser rather then letting participants use their phones in their own typical setting.

Anti-phishing techniques for the web have been deployed in practice [73, 149]. Proposals include automated comparison of website URLs [189], visual comparison of website contents [58, 287], use of a separate and trusted authentication device [212], personalized indicators [72, 171, 232], multi-stage authentication [148], and attention key sequences to trigger security checks on websites [276]. While some of these mechanisms are specific to the web environment, others could be adapted also for mobile application phishing detection. Website phishing in the context of mobile web browsers has been studied in [198, 229].

Finally, we focus on previous research that studied the effectiveness of security indicators. These work have mostly focused on phishing on the web. Studies in this context have shown that users tend to ignore security indicators such as personalized images [171, 232] or toolbars [157, 275]. Browser warnings have been evaluated positively in a recent work [2], even if previous studies suggest otherwise [73, 83, 248]. In the context of mobile browsers, a recent work [5] further showed how the situation is worse than for desktop browsers.

## 9.8 Summary and Future Work

Despite the security mechanisms used in modern smartphones, it is still possible for an attacker to install a malicious application on his victims' phones. In this chapter we looked at application phishing attacks in detail. Such attacks are an emerging threat for mobile application platforms and the first successful attacks have already caused significant financial losses.

We first analyzed possible ways in which a phishing application can fool the victim and steal his credentials. When listing possible countermeasures against phishing attacks we noticed that no single one could protect against all types of attacks. We focused our attention on personalized indicators, which are easy to deploy and are a well-known countermeasure to address the problem of phishing. Previous studies in the context of websites have shown that indicators fail to prevent the majority of attacks. Consequently, the research community has deemed personalized indicators an ineffective anti-phishing mechanism. In this chapter we reported our findings from the first user study on smartphones that evaluates the effectiveness of personalized security indicators for mobile applications. Our results show a clear improvement over the previous studies, which gives us reason to believe that indicators, when applied to an appropriate context, may be more effective than their current reputation suggests. We conclude that the research community should revisit the question of personalized indicator effectiveness and further studies are needed to fully understand their benefits in new deployment contexts such as in mobile applications. Finally, we proposed a secure setup protocol for personalized security indicators.

### 9.8.1 Future Work

We presented a first user study on how security indicators in the new context of mobile applications can help detect application phishing attacks. We now discuss interesting directions for future research.

**Bank Deployment.** Our user study is based on a mobile banking application we developed. We could not replicate previous studies that used clients of a real bank that uses security indicators. One direction for future work is to collaborate with a banking institution to study the effectiveness of personalized security indicators in a real deployment. This would allow the study to last longer and target a varied population of users.

**Personalized Security Indicators.** In our user study we allowed participants to choose an indicator from their photo gallery. The indicator was then placed prominently on the login screen. Future user studies could look at different indicator size and position. Furthermore one could further evaluate the effectiveness of different indicator types such as interactive indicators. Interactive indicators require the user to not only assess their presence but also interact with them in some manner, such as for example tapping on them or swiping across them.

**User Perception.** An interesting research direction is to study how much users understand changes in user interfaces. This is of particular interest in case that a solution that compares applications UIs is used to detect phishing attacks. Such a solution is fooled by an attacker that changes slightly the appearance of the malicious application. The question that needs to be answered is how much can one change the UI of an application before users would not enter their credentials anymore.

# Chapter 10
# Analysis of the Communication between Colluding Applications on Modern Smartphones

## 10.1 Introduction

Modern smartphone operating systems allow users to install third-party applications directly from on-line *application markets*. Given the large number of applications and different independent developers, every application cannot be trusted to behave according to its declared purpose. Certain types of malicious behaviors can be detected by inspection and testing while others cannot; malicious applications therefore find their way into the application markets [6, 202, 203, 205].

To limit the potential impact of malicious applications, mobile phone operating systems (e.g., Android OS [126], Windows Phone [193]) implement a permission-based security model (also called a *capability model*) that restricts the operations that each application can perform. This model explicitly gives permissions to applications in order for them to execute their required operations. Recent work by Schlegel et al. [233] introduces smartphone malware that makes use of application collusion over a limited number of communication channels to overcome the security mechanisms put in place by the implemented permission-based model. By establishing communication over a covert or overt channel, applications are allowed to execute operations which the system, based on their declared permissions, should prohibit.

It is important to stress that application collusion attacks on permission-based models are not a result of a software vulnerability and are not related to a particular implementation. Instead, they are a consequence of the basic assumption on which the permission-based model relies: applications can be independently restricted in accessing resources and then safely composed on a single platform. Collusion attacks show that this assumption is incorrect and can be exploited to circumvent the permission-based model. Furthermore, in current systems, users are not made aware of possible implications of application collusion attacks—quite the contrary—users are implicitly lead to believe that by approving the installation of individual

applications independently, they can limit the damage that an application can cause.

Although the existence of overt and covert channels and thus the feasibility of application collusion on any platform might not be surprising, the implications of collusion are very damaging on mobile platforms: these platforms are designed for personal use, generally store personal information and facilitate the installation of multiple third-party applications. Furthermore, existing security products (e.g., Lookout Privacy Advisor [253]) analyze and report application permissions independently for each individual application. Since they do not consider application collusion, these products do not correctly reflect the collective privacy implications of the applications that the users install.

In this work, we demonstrate the existence of a number of overt and covert channels by implementing them on an Android platform. We then evaluate the severity of the threats posed by application collusion attacks by measuring the throughput and stability of each channel. Our results show that different covert channels, which are generally hard to detect or prevent, can reach throughputs ranging from 3.70 bps up to roughly 3.27 kbps on the Nexus One and from 0.47 bps up to 4.22 kbps on the Samsung Galaxy S, thus posing a serious threat to privacy on modern smartphones.

While it has been shown that overt channels on mobile smartphones may be detected and restricted by using taint analysis [89], policy enforcement [43, 44], better sandboxing and by reducing access to some APIs, we show that these approaches fail to detect most covert channels. This is consistent with research carried out in the 1970's where it has been shown that covert channels in computer systems are hard to prevent [71, 178]. To evaluate the effectiveness of contemporary tools, we tested both TaintDroid [89] and XManDroid [44] and confirmed that they do not detect all channels and therefore fail to fully prevent application collusion. This shows that application collusion attacks remain a real threat on modern smartphone platforms. Finally, we propose ways of preventing or limiting some of these channels.

In summary, the contributions of this chapter are the following. (1) We demonstrate the practicality of application collusion attacks by implementing several communication channels on the Android platform. (2) We measure and report the throughput of implemented communication channels highlighting the extent of the threat posed by such attacks. (3) We confirm that two recently proposed architecture modifications and tools that deal with overt and covert channels discovery, TaintDroid [89] and XManDroid [44], still fail at detecting several of the implemented channels.

(4) We propose countermeasures that, if not eliminate, limit the capacity of selected communication channels between the applications.

## 10.2 Problem Statement

The goal of this work is to understand the threat posed by colluding applications on modern smartphones. In particular we investigate the feasibility and the practicality of multiple communication channels in terms of throughput, bit-error rate and required synchronization. Figure 10.1(a) illustrates an example channel between two applications. On the left, the *ContactsManager* application has access to private data on the device but not to the network (later in the work referred to as the *source* application), on the right, the *Weather* application having access to the network but no direct access to the private data (later in the work referred to as the *sink* application). The two applications can create a stealthy communication channel to share data. Figure 10.1(b) illustrates an interesting covert channel that can be created between an application and the Browser which does not require any extra application to be installed on the device. We will describe this channel in more detail in Section 10.3.3.

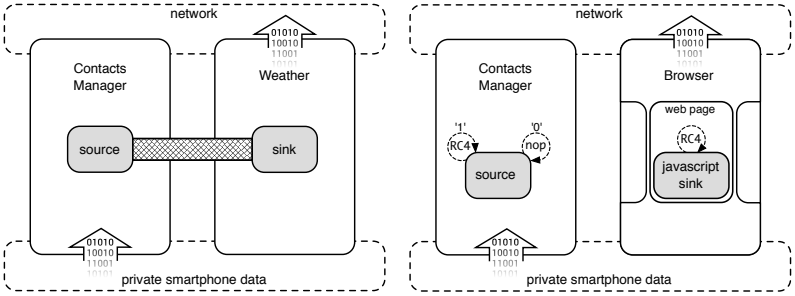



Figure 10.1: On the left, Figure (a) shows a generic example of collusion between the *ContactsManager* and the *Weather* applications through a stealthy (overt) communication channel. On the right, Figure (b) shows a (covert) timing channel between an application and the browser working through the use of dummy RC4 operations.

| Overt Channel | Throughput (kbps) | | | |
|---|---|---|---|---|
| | Nexus One | | Samsung Galaxy S | |
| UNIX Socket Communication | 340.45 | (± 154.02) | 34.78 | (± 11.39) |
| Internal Storage | 292.03 | (± 50.06) | 32.60 | (± 8.47) |
| Shared Preferences | 75.81 | (± 6.83) | 31.00 | (± 2.75) |
| Broadcast Intents | 40.58 | (± 8.41) | 26.74 | (± 4.88) |
| External Storage † | 11.55 | (± 1.10) | 6.12 | (± 3.95) |
| **System Log** ‡ | 2.94 | (± 0.03) | 2.14 | (± 0.11) |

† Requires extra `WRITE_EXTERNAL_STORAGE` permission.
‡ Requires extra `READ_LOGS` permission.

Table 10.1: List of our implemented *overt* channels in the Android OS with corresponding throughputs (with the 95% confidence intervals in parenthesis). The displayed values are averaged over 10 runs for both the Nexus One and the Samsung Galaxy S. The "System Log" is a new channel that we engineered and for which we did not find references in the open literature.

### 10.2.1 Channels Classification

We classify communication channels based on their implementation on current smartphone architectures as follows:

- **Application.** This is the level of the API that an operating system provides to the developers (e.g., Android's Java API, Windows Phone 7 C # / Silverlight APIs, iOS's Objective-C API). Access and usage of these communication channels may be easily controlled. We consider these channels as *high*-level.
- **OS.** This is the level of the operating system that is exposed through native calls that exploit information present in the operating system. We believe that at this level some channels are impossible to close, others, if closed, could hamper backward compatibility severely.
- **Hardware.** This is the level that is exposed through exploiting hardware functionalities of the smartphone. It is highly dependent on individual hardware specifications of smartphone models. These communication channels cannot be closed without severe performance degradation of the system. We consider these channels as *low*-level.

Different levels usually also imply different throughput and stealth. In particular, we notice that throughput is usually directly proportional to the level, with higher throughput associated to *high*-level communication

channels. Stealth (i.e., the difficulty to detect a communication channel), on the other hand, is usually inversely proportional to the level, with stealthier channels associated to *low*-level communication channels.

## 10.3 Overt and Covert Channels in Android

We explore possible covert and overt channels on Android smartphones. We analyzed some known channels and identified a number of new channels specific to smartphones not yet presented in the literature.

To analyze overt and covert channels, we implemented a framework to measure the throughput, the bit error rates and the synchronization times for each implemented communication channel. The results of our study are presented in Tables 10.1 and 10.2.

The values shown in both tables are averaged over 10 independent runs for each implemented channel executing on a Nexus One or a Samsung Galaxy S smartphone. During the tests the phone was running on battery power and not charging. Each time the *source* application tries to send 4, 8 and 135 byte (to mimic the transfer of contact information as explained in Section 10.3.4) messages to the *sink* that, if the channel is open, would record them successfully. For each covert channel that requires tight synchronization between the *source* and the *sink* application (i.e., timing channels), we implement a synchronization protocol and run it before starting to send data. In general the synchronization protocol is used to estimate the noise present in the system when the applications want to share data and to correctly start the measurements to exchange data at the same time. Such a protocol is implemented on top of the covert channel (i.e., using the same mechanism) and allows us to reach higher accuracy. A covert channel's accuracy is measured in bit error rate, with perfectly accurate channels reaching 0% bit error rate during transmission. While synchronization time values are reported in Table 10.2 for completeness, we believe that they can be further optimized to yield overall slightly faster communication channels.

### 10.3.1 Overt Channels

We now briefly describe the implementation of overt channels to give an intuition of how they work.

*Shared Preferences (Application):* the *sink* application uses an API to create an Android preference XML file that is world-readable and world-writable. The *source* application writes ASCII data to it and the *sink* reads it. This

channel does not require any synchronization to operate as the two applications do not need to be run simultaneously.
*Internal Storage (Application):* the *source* application writes a world-readable file to the internal storage, the *sink* application reads its contents. Similarly the *External Storage* simply uses a file on the external storage. For the external channel to work, the *source* application requires an extra permission: `WRITE_EXTERNAL_STORAGE`. Again, similar to the *Shared Preferences* communication channel, these channels do not require synchronization between the applications.
*Broadcast Intents (Application):* the *source* application communicates by adding private data as extra payload to a broadcast message sent to the system. Broadcast intents are a particular type of messages that are used in the Android OS to enable one form of communication between applications. The operating system, upon receiving such a message with its payload, broadcasts it to all the applications that requested to be notified when such a message is received (i.e., by registering themselves for a particular ID that is used to identify the message). The *sink* application registers itself with the system and receives the message sent by the *source*. While both applications need to be running at the same time, no synchronization is required in order for the channel to work.
*System Log (Application):* the *source* writes a specially-crafted message to the system log that the *sink* then reads to extract the information. The extra `READ_LOGS` permission is required by the *sink* application in order to be able to read the system logs. Messages longer than 4000 characters must be split and binary data must be encoded, because data is otherwise lost when inserted into the log. Given that the log has a finite number of entries that are held at any time, the *sink* application must be activated before the message sent by the *source* is deleted. Alternatively, the *source* could repeatedly insert the message at time intervals to increase the chance that the *sink* receives it. Potentially the channel can be rendered stealthy by filling the log with seemingly meaningful logging data after the communication takes place.
*UNIX Socket Communication (OS):* the *source* sends the data through a UNIX socket that the *sink* application opened. For this channel to work correctly, both applications must be simultaneously active.

### 10.3.2 Covert Channels

We now describe the covert channels that we implemented and measured. As the storage of these channels is not persistent, all these channels are synchronous. This means that before starting to exchange data over the

channel a synchronization protocol between the *source* and the *sink* must be run in order to achieve better accuracy during the data-exchange phase. For channels where accuracy is not specifically stated, our implementation reached perfect accuracy.

*Single and Multiple Settings (Application):* the *source* modifies a general setting on the phone and the *sink* reads it as described in [233]. Multiple settings can be changed at the same time to achieve higher throughput. Most settings in Android can be changed and read without requesting any permissions. This particular covert channel can be closed by disabling or requiring extra permissions in order to change particular settings.

*Type of Intents (Application):* the *source* sends a broadcast message (similar to the *Broadcast Intents* overt channel) to the *sink* and encodes the data to be transmitted into the type of the intent (i.e., flags, action, particular extra data), rather than directly exchanging the data as the extra payload of the message. In contrast to the similar overt channel that uses *Broadcast Intents*, this covert channel is not detectable by tainting mechanisms or similar solutions. The *sink* application still needs to register with the system in order to receive the intents.

*Automatic Intents (Application/OS):* the *source* modifies particular settings (i.e., the vibration setting [233]) that trigger automatic broadcasts by the system to all applications that registered to be notified when such a change happens. The *sink* receives the messages and infers the data depending on the contents of the received broadcasts. For instance, changing the vibration setting of the phone triggers a broadcast which contains 1-bit of information (vibration on equals to 1, vibration off equals to 0).

*Threads Enumeration (OS):* the *source* spawns a number of threads and the *sink* reads how many threads are currently active for the source application by looking into the `/proc` directory of *source*. This particular covert channel can be closed by controlling application access to the `/proc` filesystem or by mediating the access through a system service.

| Covert Channel | Throughput (bps) | | | | Synchronization (ms) | | | |
|---|---|---|---|---|---|---|---|---|
| | Nexus One | | Samsung Galaxy S | | Nexus One | | Samsung Galaxy S | |
| **Type of Intents** | 3350.85 | (± 134.11) | 4324.13 | (± 555.32) | 716.8 | (± 168.2) | 473.0 | (± 249.0) |
| **UNIX Socket Discovery** | 2610.92 | (± 305.25) | 1647.78 | (± 170.70) | 5.2 | (± 0.8) | 13.9 | (± 2.2) |
| Multiple Settings | 239.76 | (± 9.41) | 284.91 | (± 1.90) | 314.9 | (± 21.8) | 302.1 | (± 11.0) |
| **Threads Enumeration** | 157.73 | (± 0.97) | 139.39 | (± 7.40) | 71.6 | (± 7.1) | 110.1 | (± 8.8) |
| Automatic Intents [233] | 51.38 | (± 0.41) | 90.67 | (± 0.39) | 1083.2 | (± 75.1) | 435.1 | (± 180.8) |
| Single Settings [233] | 46.88 | (± 0.31) | 65.89 | (± 0.73) | 267.5 | (± 3.2) | 273.4 | (± 11.9) |
| Free Space on Filesystem | 13.07 | (± 0.00) | 9.80 | (± 0.00) | 1038.2 | (± 5.1) | 1442.7 | (± 15.6) |
| **Reading `/proc/stat`** | 7.82 | (± 0.00) | 3.26 | (± 0.00) | 6923.4 | (± 8.1) | 16669.2 | (± 48.7) |
| **Processor Frequency** | 4.88 | (± 0.00) | 0.47 | (± 0.09) | 8203.9 | (± 7.2) | 78866.1 | (± 9156.8) |
| Timing Channel | 3.70 | (± 0.00) | 3.69 | (± 0.01) | 10286.8 | (± 16.1) | 68057.6 | (± 105259.4) |

Table 10.2: List of implemented *covert* channels in the Android OS with their corresponding throughput (95% confidence intervals are shown in parenthesis). The displayed values are averaged over 10 runs for both the Nexus One and the Samsung Galaxy S. Channels listed in **bold** are new channels that we engineered and for which we did not find references in the open literature. In the synchronization column we present the time required to run the synchronization protocol before starting to transmit data through the channel.

*UNIX Socket Discovery (OS):* the *source* uses two sockets, a synchronization socket and a communication socket. The *sink* checks if the *source* communication socket's state is open, and infers the transferred bit. The synchronization socket is open if the communication socket can be checked. *Free Space on Filesystem (OS):* the *source* application writes or deletes data on the disk to encode the information for the *sink*. The channel throughput depends on the noise in the system; for example the sink application could infer a larger amount of information depending on how many free blocks are available. The data presented in Table 10.2 was generated by having the *source* allocate three blocks to encode a '1' and clearing three blocks to encode a '0'. The *sink* checked the available blocks in the system at predefined time intervals (75ms for the Nexus One and 100ms for the Galaxy S). The current implementation yields bit-errors percentages between 0.01% (Nexus One) and 0.03% (Samsung Galaxy S). A possible solution for preventing this channel is to enforce a quota on the available space for each application (that could potentially vary depending on the number of applications on the system) and report the free blocks remaining of each quota rather than the free blocks in the overall system.

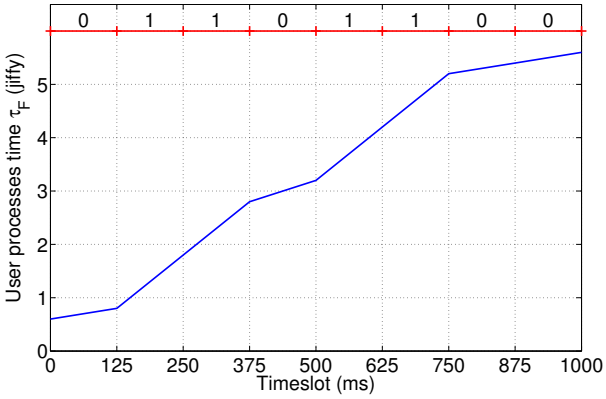



Figure 10.2: Exemplification schematic rise of the value $\tau_F$ (number of jiffies spent for every user process) over time when sending the bits written on the top.

*Reading* `/proc/stat` *(OS):* the *source* application performs some computations, while the *sink* monitors the processor usage statistics. These are available in the `/proc/stat` virtual file where the Linux kernel provides information about the current system load (as the number of jiffies used for

all user processes). A schematic representation of how the values ($\tau_F$) read in `/proc/stat` change depending on the bit that the *source* wants to send is presented in Figure 10.2. The overall idea is that sending a '1' causes, in the values read, a steeper slope than sending a '0'. Figure 10.3 presents the trade-off between throughput and accuracy of this channel. Other channels behave similarly to this one, with higher throughput resulting into lower accuracy. The current implementation yields bit-error percentages between 0% (Samsung Galaxy S) and 0.10% (Nexus One). Similarly to the *Threads Enumeration* covert channel, this channel could be closed by preventing read access to the `/proc` filesystem.

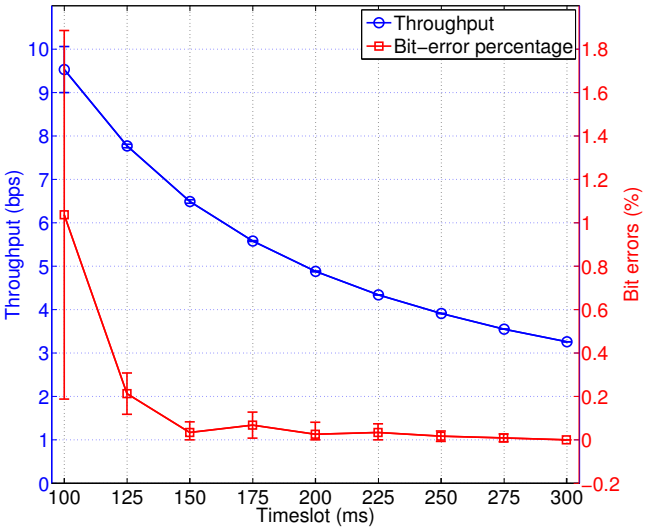



Figure 10.3: The graph shows the trade-off between throughput and accuracy (measured in bit errors) for the `/proc/stat` channel. Values are averaged over 5 independent runs.

*Timing Channel (Hardware):* the data transmission between the source and the sink is performed by varying the load on the system. The *source* runs CPU-intensive tasks to send the bit '1', on the other hand it does not perform any CPU-intensive operation to send the bit '0'. The *sink* continuously runs computation-intensive operations and records the time required to complete them. The *sink* uses this time to infer the presence of computation by the *source* thus inferring the transmitted bit. For reliable differentiation of bits based on the time, an initial *learning period* is used

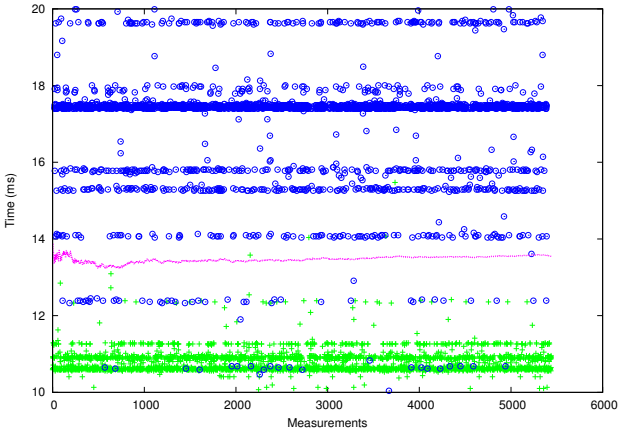



Figure 10.4: Measurements taken by the *sink* application to infer information sent over a Timing Channel. The blue dots show measurements while the *source* is sending a '1', green crosses while it is sending a '0'. 5400 measurements are needed to receive a 135-byte message (5 measurements are used to assign a value to a bit through a majority-vote mechanism). The red line shows a moving average used as a threshold value.

to benchmark the system behavior. Finally, to eliminate the noise in the system, we use a majority vote (out of five measurements) at the *sink* to decide the value of a particular bit depending on a threshold value updated with a moving average. In our implementation, the time difference between transmitting a '1' and a '0' is approximately 6 ms in the case of the Nexus One. The current implementation yields bit-errors percentages between 0.10% (Nexus One) and 0.05% (Samsung Galaxy S).

To better visualize the majority vote mechanism used in our implementation, in Figure 10.4 we present an example trace of the measurements taken at the *sink*. The blue dots represent measurements done while the *source* was sending a '1', the green crosses while it was sending a '0'. 5400 measurements were performed to send the test message of 135 bytes (1080 bits) as it is shown in the figure.

*Processor Frequency (Hardware):* this channel is an improvement over the basic *Timing Channel*; in this particular instance we take into account Dynamic Frequency Scaling (used on the smartphones that we tested) to improve the throughput and reduce the synchronization time (for the Nexus One). While the *source* behavior remains the same as in the case of

the *Timing Channel*, the *sink* instead monitors the trend of the processor frequency by repeatedly querying it from the system and thereby decodes the current bit. Afterwards the *source* waits a fixed amount of time to allow the CPU to "slow down" again before the next bit transmission is started. The current implementation yields bit error percentages between 0.14% (Nexus One) and 4.67% (Samsung Galaxy S).

### 10.3.3 Communication Channel With External Agents

We extend the concept of colluding applications and consider the scenario in which there is only one application installed on the system that has access to private data and wants to disclose it to a third-party web service without requesting the permissions to connect to the network. Furthermore, we want to ensure the successful transmission of the private data through a channel that is hard to detect. Here, the colluding *sink* application resides on a web page executing intensive JavaScript operations, that is opened within the system browser. The phone will show the page on the screen, therefore, to decrease detection by the user, the operation can be carried out when the phone screen is off (for example, during night time). To reach the *sink*, the *source* application uses a covert timing channel similar to the *Processor Frequency* covert channel. However the *sink* cannot directly query the processor frequency, as it is inside the JavaScript sandbox. Such channel is visualized in Figure 10.1(b).

We have implemented and tested this proposed covert channel as follows: depending on the current bit to transfer, the *source* either tries to increase the processor frequency or sleeps. Afterwards the *sink* measures how many dummy RC4 operations it can perform in a fixed time period, thereby getting the processor frequency and the transmitted bit. The possibility to use the browser to send private data has been described in [177] but their proposed method is easily detected by flow-tracking techniques (such as TaintDroid). Our proposed covert channel—while having a low throughput of roughly 1.29bps on the Nexus One—is also much harder to detect and cannot be detected by today's tools.

### 10.3.4 Results of the Analysis

The experiment results reported in Tables 10.1 and 10.2 indicate that the attacker's choice of one channel over another, depends on the nature and size of data that needs to be transmitted between applications. For example GPS coordinates usually consist of two floating point numbers (represented by 32 bits of data), in contrast contacts, for example, might

have a varying number of characters: in order to simulate a few full names and corresponding phone numbers we transmit 135 bytes of information.

Given a rough estimate of the size of the data that can be shared between applications, we conclude that even the covert channels with low throughput, such as the *Timing* or the *Processor Frequency* channels (respectively at around 3.70 and 4.88 bps) enable the sharing of reasonable amounts of data on the smartphone. For example, exchanging GPS coordinates requires roughly 19.4 or 14.8 seconds respectively; sharing 135 byte contacts requires roughly 304.9 or 231.1 seconds respectively. Covert channels with higher throughput, such as the *Type of Intents* or *UNIX Socket Discovery*, reaching up to 4324.13 and 2610.92 bps, enable the exchange of GPS coordinates or contact information in less than a second.

Another interesting result of the analysis is that most channels, when tested on the more powerful Samsung Galaxy S, did not perform better than on the Nexus One. For CPU-bound channels these results come from the fact that Samsung ships the device with a larger number of active services which influence the different channels. For channels based on Processor Frequency it is based on the fact that there is a different frequency governor. For IO-bound channels these results come from the fact that the Samsung device uses a Samsung-developed file system rather then the standard YAFFS2 used on the Nexus One.

Overall, the results show that application collusion attacks, through the usage of different communication channels depending on the amount of data that needs to be transmitted, are a realistic attack and therefore a serious threat.

## 10.4 Analysis of Existing Tools

In this Section we test two recently proposed tools that try to solve the information leakage on modern smartphones, in particular TaintDroid [89] and XManDroid [44]. We use the Nexus One as the test phone where we successfully install both tools and report the findings.

### 10.4.1 TaintDroid

TaintDroid [89] tries to track information flows within an application and between applications; it is implemented as a modification of the Android operating system. Using dynamic taint-tracking, the modified OS follows the information flow of tagged data, that is, data which is generated from sources of private information, including the user contacts and the GPS location.

Inside the Dalvik VM, TaintDroid employs variable tracking and propagates taint through primitive data types, exception handling routines and array lookups. Tainting information, though, does not follow through in native code (such as JNI native libraries) execution. Due to this limitation, at the moment of writing, trying to use native libraries not residing in the `/system` folder results in an application crash. Additionally, taint is propagated through IPC messages, by performing message-level tracking.

Whenever tainted data reaches a sink (such as the network), a notification is shown informing the user about the application that is leaking data, the originating data class and the network transmission. Interestingly enough, the implementation of TaintDroid notifies the user when the *sink* application uses Java's `HttpURLConnection` to send the data off the device, but no detection happens when it uses a UDP connection (i.e., through the Java `DatagramSocket` class). We believe this is just an implementation detail overlooked by the authors rather than a design flaw of the proposed solution.

In our study, TaintDroid was able to correctly report the transmission of sensitive data for the following overt channels: *Internal Storage* and *Broadcast Intents*. The *External Storage* channel was not detected: this happens because taint information is propagated using extended attributes and external storage uses the FAT file-system which does not support them. Surprisingly the remaining overt channels (*Shared Preferences* and *System Log*), which should be detected by TaintDroid were not detected. Further analysis shows that the implementation of the logging mechanism in Android OS is carried out in native code (i.e., C). As previously stated, TaintDroid is not currently capable to extend tagging to native code and therefore cannot detect this channel.

Given that the authors explicitly state that the TaintDroid mechanisms can be circumvented through the use of implicit flows, it is clear that the covert channels implemented in our framework remain undetected due to their bit-wise nature. To remove the taint from tainted variables, such that higher-throughput overt channels can be used successfully, we propose four different techniques. We implement each technique and test it on a Nexus One to report the throughput.

*n-way Switch Statement [51]:* an n-way Switch Statement can be used to strip the taint off $log(n)$ bits. The Switch Statement reads the tainted value and writes the corresponding constant into a new untainted memory location. The taint does not propagate, because constant values are used. We measured the throughput of this technique to be, roughly, 27.65 Mbps (megabits per second).

*Java Exception Handling:* Here we encode a tainted bit in the existence of a Java exception. If the tainted bit is '1', an exception is thrown that causes the untainted bit as well to be set to '1' by the exception handler [196]. We measured the throughput of this technique to be, roughly, 107.42 kbps (kilobits per second).
*File-based:* This technique encodes the tainted bit in the existence of a file inside the application's private directory. Depending on the tainted bit a special file is either created or not. The untainted bit is set depending on the results of the following existence check for the special file. We measured the throughput of this technique to be, roughly, 680 bps (bits per second).
*Timing-based:* The application's own execution time encodes the tainted bit in this example. The application delays its own execution by sleeping in order to signal a one. Timing measurements determine the value of the untainted bit. We measured the throughput of this technique to be, approximately, 98 bps (bits per second).

Given the throughput of each untainting technique, and the fact that covert channels remain undetected by TaintDroid, we conclude that employing a dynamic flow-tracking technique does not prevent application collusion attacks.

### 10.4.2 XManDroid

XManDroid was first presented in [43] and later extended in [44]. It aims at implementing different techniques to successfully mitigate the problems of *confused deputy* attacks and direct *application collusion* attacks.

The authors propose a security framework to enable policy enforcement at different system levels on Android. The instantiation of the framework extends various parts of the Android OS, in particular they port and extend TOMOYO Linux [145]. The security framework modifies Android *reference monitor* to check for direct IPC calls at runtime between applications and indirect communication through Android system components (i.e., the settings manager). Furthermore, kernel-level MAC (Mandatory Access Control) monitors access to different resources such as the file system, UNIX sockets and internet sockets. System policies are expressed in a high-level language and specify which flows are to be denied.

The prototype with which we experimented was able to block a subset of channels that are potentially detected by XManDroid. In particular the prototype successfully detected all the *overt* channels except the *System Log* channel. It also successfully detected the *Type of Intents* and *UNIX Socket Discovery* covert channels, as they work over explicit communication between applications. Further, the *Reading* `/proc/stat` and *Threads*

*Enumeration* covert channels are detected by the fact that they work by accessing the `/proc` file system, blocked by the TOMOYO Linux access control.

Similarly, it is safe to assume that XManDroid would be able to detect the *Broadcast Intents* and *UNIX Sockets Communication* channel, because they work over explicit communication between applications. The *System Log* channel could also be detected because it works over simultaneous access of a shared file (i.e., similarly to the *storage*-based channels that are detected).

This leaves our analysis with a small subset of covert channels that are not detected by XManDroid: *Free Space on Filesystem*, *Processor Frequency* and finally *Timing Channel*. In particular the last two that are *hardware-level* communication channels are not in the scope of XManDroid.

One limitation of using XManDroid and similar tools is that they might report false-positive results when two non-malicious applications try to share legitimate data, as the communication is blocked even if non-sensitive data is shared (i.e., XManDroid is agnostic of the transmitted data). While the authors claim that this is not an issue because non-malicious applications generally do not tend to share data, it might be interesting to understand if it is possible to render XManDroid data-gnostic. Similarly, restricting access through policies to parts of the system (i.e., `/proc`) might result in some applications to malfunction in case they rely extensively on access to such resources. Finally, XManDroid works by specifically adding hooks to system functions or to the kernel as new channels are discovered, and is therefore a reactive solution.

Given that some channels remain undetected under existing state-of-the-art tools, we conclude that application collusion attacks remain a threat and stress that research should focus on closing these (obviously harder to deal with) channels.

## 10.5 Mitigation Techniques and Limitations

Solving the confinement problem, and in particular closing all possible covert channels in a system, is known to be a difficult problem [71, 178]. It is further complex in the case of smartphones, where performance, application markets openness and exposed API features are key to user and developer adoption. Mitigation can be achieved either at design time (by reducing access to sensitive APIs or by limiting communication possibilities) or by analyzing static and dynamic properties of applications and their interactions off-line or at run-time.

### 10.5.1 Design Time Mitigation Techniques

**General Purpose Techniques.** There are a number of techniques that could be considered by smartphone operating system designers:
*User control on private data access:* As in Windows Phone 7, involving user action on each data access helps to mitigate the impact of colluding applications (and more generally, malicious applications). However, this also limits applications capabilities; for example, in such an environment, it is impossible for third-party developers to create applications that perform automated backups of private data.
*Limiting APIs:* When designing APIs exposed to third-party developers, designers should carefully consider the possibility that the API may create a communication channel between applications. If an overt or covert channel is found, it should be either mitigated or its access should be controllable through the system's policies.
*Limiting Multitasking:* Reduces the possibility of covert channels resulting from competition for access to resources (CPU time, cache and bus contention). However, this limits the diversity of applications that can be implemented on the system.
*Application Review:* Performed to detect colluding applications before publication of applications on the markets. However, this approach requires dedicated techniques to detect application collusion.
*Policy-Based Installation Strategy:* Could be used in a corporate scenario where installation of applications can be limited through some policies e.g., deny access to applications that read contacts.
**Application-Level Channels.** Communication channels constructed at this level are dependent on the APIs exposed by the underlying operating system. Careful design of permissions used to access data sources as well as data sharing points (i.e., sharing of files or preferences, settings, broadcast intents) could draw attention towards applications that require an excessive number of permissions. Furthermore, some of these channels could be closed by removing unnecessary APIs after an analysis of the used and unused ones. This would enable tighter security while maintaining a reasonable amount of freedom for the developers. For instance, the *System Log* channel can be closed by allowing access to the log file only when "USB Debugging" is active. This is a mode to which the phone switches to when connected to a PC, for instance, for development purposes.
**Operating-System-Level Channels.** Covert channels that can be established at this level might not be detectable by information flow analysis and their prevention requires further investigation. Such channels usually utilize mechanisms offered by the underlying kernel (i.e., sockets, threads,

child processes) and therefore, removing such functionality by preventing developers from using it might impede certain applications and their potential optimization. Other system information made available (for example, through the `/proc` filesystem by the Linux kernel) could be restricted (for example, as done in GRSecurity [138], TOMOYO Linux [145] or SEAndroid [244]) or mediated by operating system services that could directly control access to such information.

**Hardware-Level Channels.** Covert channels (e.g., timing) established at this level are the usually hardest to remove without serious performance degradation or functionality impact. Solutions, such as preventing multitasking or flushing caches between process scheduling, limit the overall performance or responsiveness of the system and increase its power consumption. Furthermore, common data tainting or information flow control techniques are ineffective in this scenario since communication happens at the bit-level of the transmitted data. Closing these channels requires novel approaches, e.g., design of information flow secure systems from the bottom up [255], however redesigning current smartphone systems from scratch is likely to have a prohibitive cost.

One possible solution for timing-based channels (similarly proposed for different systems in [150]) is to add a new permission to the Android OS (for example, it can be named `REQUIRE_PRECISE_TIMING`). Applications requiring such permission, upon requesting timing information, would be given precise timing information (i.e., games require precise timing for physics engines or graphics display). Applications without such permission would be presented with a rough estimate of timing (i.e., ±5 seconds). This modification would disrupt the correct functioning of communication channels that require precise timing for their operation such as, in our reported channels, the *Timing Channel* and the *Processor Frequency* channel. For example, in our implementation the *Timing Channel* works over 6 ms differences (to send a '1' or a '0'), therefore if the system would report time to applications at a granularity higher than 6 ms the channel would be disrupted.

If further analysis shows that precise timing is indeed required for the correct functioning of a large number of applications, another viable solution to disrupt timing channels is to limit the number of times applications can request timing information.

### 10.5.2 Application Analysis Techniques

**Black-box Analysis.** One strategy in trying to detect collusion is to add a data monitor between separate applications on the device. This would

remove the need to detect covert channels by only monitoring data leakage itself. In such an architecture, when data from one application is used, the monitor would store it (or a fingerprint of it) and track the data sent to the colluding application. While this approach seems promising, it is inherently limited: malware can encode data in a way that it leaves the mobile device still encoded (e.g., encryption using a public key), defeating the monitoring. While it is clear that black-box analysis may detect some trivial attempts to evade the system security policy, it clearly does not provide a complete solution.

**Exclusive access to sensitive resources.** Techniques to limit access to sensitive resources (e.g., the microphone) from third-party applications when a sensitive operation is ongoing (e.g., a phone call), as presented in [233], only prevents malware from accessing that particular data at that instant. Such techniques cannot be applied generally: for example, access to the GPS data would always be considered a privacy invasive operation and therefore would never be allowed.

**Offline application analysis.** Since colluding applications are communicating on an unexpected channel, it is likely that when colluding applications are executed simultaneously on a device, they would show a different behavior than when executed independently. For example, they would detect each others presence and engage into communication over a covert channel. Behavioral analysis could be used to detect such a change of behavior, for example executing applications on an emulator alone or in pair and comparing execution traces and coverage. However, given the vast number of potential pairs of colluding applications, this solution does not scale. This can be addressed by strategies which include evaluating applications according to their popularity or according to "replicated" installations [216].

## 10.6 Related Work

In chapter 8 we discussed smartphone malware attacks and countermeasures in general. In the following we review related work focusing on overt and covert communication channels.

Lampson first described the confinement problem [170] as the problem of preventing unauthorized communication, over overt or covert channels, between two subjects on a system. It is recognized to be a difficult problem in practice, for example Denning and Denning state that "Cost-effective methods of closing all covert channels completely probably do not exist" [71].

While overt channels can be managed by security policies, covert channels are communication channels built from resources that are not intended for communication, and so they cannot be mitigated with the same techniques. Covert channels were also used to perform covert communications over networks [117, 214], however in this work we mainly focused on inter-process covert channels. Inter-process covert channels can be classified as either software (sometimes referred to as TCB channels) or hardware (also known as fundamental channels) and communicate over timing or storage channels. However, this distinction is more empirical than theoretical [118].

Software covert channels can be mitigated by a careful analysis of the usage of visible and alterable variables used by system calls [259] or using a formal model for analyzing programs [239] using a semi-automated technique. However, hardware-related covert channels (e.g., timing, competition to access resources, paging) are difficult to prevent and recent processor designs have been shown to increase the number and efficiency of covert channels [269].

As an example, multi-core application processors are already available for smartphone devices, which would render covert channels over cache highly reliable [269]. Possible mitigation techniques include using fuzzy time [150] and preventing multitasking.

## 10.7 Summary and Future Work

In the previous chapter we focused on application phishing attacks, where a malicious application masquerades as a legitimate one to phish user credentials. In this chapter we looked at application collusion attacks. In such attacks two applications exchange information through an overt or covert channel and evade the permissions-based security mechanism used, for example, on Android devices.

We demonstrated that application collusion attacks are a serious threat given different implementation of communication channels at different system levels. In particular we identified multiple channels at the application, OS and hardware level. For each implemented channel we tested its throughput and required synchronization phase, if any. The results of the throughput measurements show that even covert channels are still sufficient to exchange private information stored on a smartphone, such as GPS locations or contact information. Finally, we confirmed that both proposed and implemented techniques and tools do not provide a complete solution against different communication channels, and are therefore insufficient to

prevent application collusion attacks. Although these attacks are harder to implement and carry out in practice, it is clear that they still pose a threat and remain an open problem.

### 10.7.1 Future Work

We implemented a number of overt and covert channels on the Android platform and showed the threat they pose. We now look at possible future research.

**Clock-based Detection.** One possible countermeasure against timing channels is that to check the access patterns to the clock device on the smartphone. We performed only a primary evaluation of this solution and note that further studies should be carried out to validate this possible solution. Furthermore one would have to understand the performance overhead of implementing such a solution which has to run on the user's device. Finally, it would be interesting to undertand which other clock sources applications can leverage that are not based on the hardware clock (e.g., callbacks from device sensors).

**Channels Extensions.** While we implemented and tested a number of overt and covert channels, it is clear that even more exist. Recent work has analyzed a new covert communication channel [188] based on heat dissipation in the context of cloud computing. It would be interesting to test if such a channel exists also in the context of smartphones which are now using multi-core processors, and also which other channels are available on today's devices.

# Chapter 11
# Closing Remarks

Smartphones are becoming ever more present in users' daily lives. Their always-connected nature and versatility have enabled a number of applications that make them the most personal device that people carry constantly. This leads to an ever-increasing amount of personal and business data being stored on these devices. It is therefore a natural consequence that, despite the security mechanisms that have been introduced to protect smartphone operating systems and applications, an increasing number of attacks target these devices. In this thesis we focused on two aspects of smartphone security. We first looked at how smartphones can be used to strengthen the security of everyday activities from web authentication to payments at points of sale. We then focused on attacks against smartphones and in particular at application phishing and collusion attacks.

In the first part of this thesis we introduced Sound-Proof, the first two-factor authentication mechanism for the web that is both transparent to the user and deployable with current browser technologies (without needing extra software to be installed on the user's computer). In Sound-Proof, two short simultaneous audio recordings from the phone and the computer are compared to provide proof of proximity between the two devices. Only when the comparison yields a positive result is the login attempt successful. We tested our solution in a variety of environments from busy cafes and train stations to more quiet offices and homes. We showed that Sound-Proof works without the user having to interact with his smartphone, even when the phone is kept in a pocket or purse. Finally, in a user study aimed at comparing the usability of Sound-Proof and Google 2-Step Verification, we discovered that users found Sound-Proof more usable. More importantly, the majority of users expressed their intention to use Sound-Proof in scenarios where two-factor authentication is optional. Finally, participants appreciated that to complete the two-factor authentication procedure with Sound-Proof took four to five times faster than with a code-based solution.

We then looked at how to strengthen the security of payments at points of sale in order to prevent a large number of fraudulent transactions caused by cloned or counterfeit credit cards. In this new setting we considered a stronger attacker model in which the adversary is able to also compromise the victim's mobile OS. With this new attacker model we showed how the location provided by a smartphone can be used in a timely, accurate and secure manner as a second authentication factor when performing a payment.

We exploited the availability of mobile trusted execution environments and presented two novel enrollment schemes that work even when an adversary has compromised the victim's device before the enrollment takes place. We validated our solution through a series of prototype implementations that showed both deployability and usability with today's platforms.

In both cases we always kept usability and deployability as primary goals of our solutions. We strongly believe that any security solution that is proposed should gauge how hard it is to apply and how difficult it is for users to adopt. No matter how good a solution is from a security standpoint, no benefits can be gained if the user adoption is small.

In the second part of this thesis we looked at open security challenges on smartphone platforms. We started with application phishing attacks, where a malicious application masquerades as a legitimate one to steal user credentials. We analyzed different ways in which such attacks are possible on current platforms and possible countermeasures. We concluded that no solution was fit to counter all types of attacks. Still, borrowing the idea of personalized security indicators from the web, where they were found to be of little help in defeating phishing attacks, we asked ourselves if this was still the case in the new context of smartphone applications. We carried out a user study to understand their effectiveness in preventing application phishing attacks. We found that although they cannot be considered a silver bullet, they are much more effective in this scenario than previously found on the web. While further studies are needed to perfect their usage, our results show that they can be used as an effective method to counter phishing attacks.

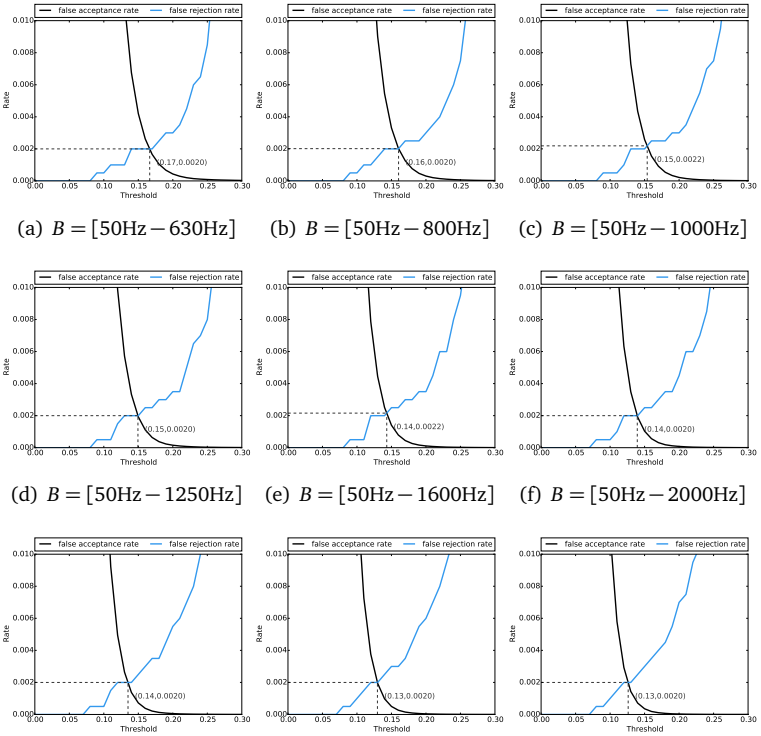
Finally, we looked at application collusion attacks. Through the use of covert and overt channels malicious smartphone applications can evade the permissions-based security mechanisms employed on modern smartphones. That is, since the user approves the installation of one application independently from others, he is led to believe that the permissions granted reflect which resources the application has access to. Through application collusion attacks this is not the case. We implement and analyze a variety of overt and covert channels that can be implemented on today's smartphones. Some channels have high throughput and are easier to detect and prevent. Other covert channels, have lower throughput rates and are harder to detect. They are therefore an open threat to the user's private data (such as GPS location or contacts).

We showcased how smartphones are not free of attack vectors and hence how the data stored on them is at risk, even today. Further research is required to better the security of current smartphones and to find new

applications where smartphones themselves can be used to strengthen the security of other systems. This thesis is a step in that direction.

# A Other Bands

We present all the plots for the considered band ranges we analyzed during our experimental campaign setting the EER to be 0.0020 as discussed in Chapter 5.

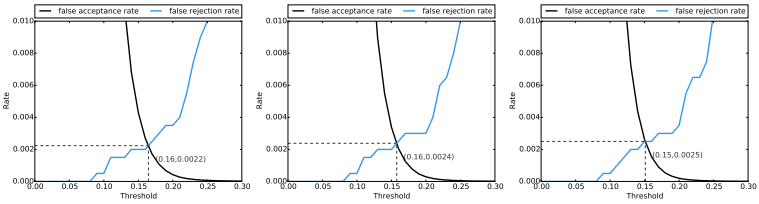



(a) $B = [50\text{Hz} - 630\text{Hz}]$ (b) $B = [50\text{Hz} - 800\text{Hz}]$ (c) $B = [50\text{Hz} - 1000\text{Hz}]$

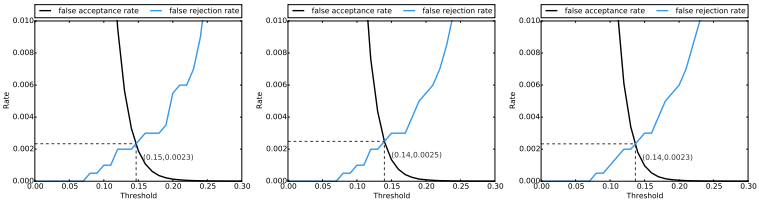
(d) $B = [50\text{Hz} - 1250\text{Hz}]$ (e) $B = [50\text{Hz} - 1600\text{Hz}]$ (f) $B = [50\text{Hz} - 2000\text{Hz}]$

(g) $B = [50\text{Hz} - 2500\text{Hz}]$ (h) $B = [50\text{Hz} - 3150\text{Hz}]$ (i) $B = [50\text{Hz} - 4000\text{Hz}]$

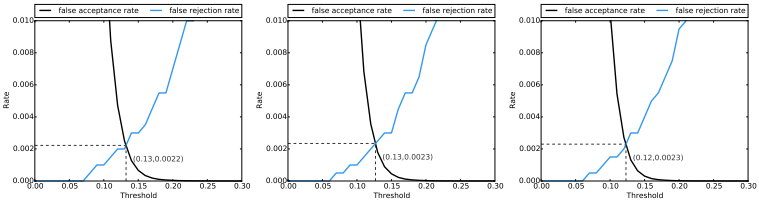
Figure A.1: False Rejection Rate and False Acceptance Rate as a function of the threshold $\tau_C$ for bands in the $B = [50\text{Hz} - [630\text{Hz} - 4\text{kHz}]]$ range




(a) $B = [63\text{Hz} - 630\text{Hz}]$ (b) $B = [63\text{Hz} - 800\text{Hz}]$ (c) $B = [63\text{Hz} - 1000\text{Hz}]$

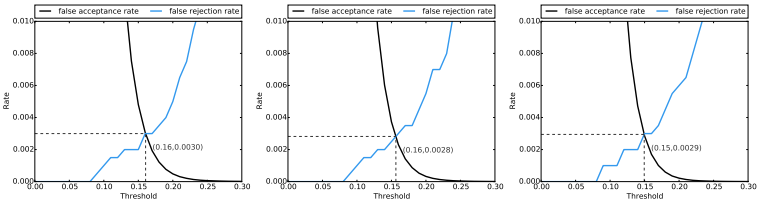



(d) $B = [63\text{Hz} - 1250\text{Hz}]$ (e) $B = [63\text{Hz} - 1600\text{Hz}]$ (f) $B = [63\text{Hz} - 2000\text{Hz}]$

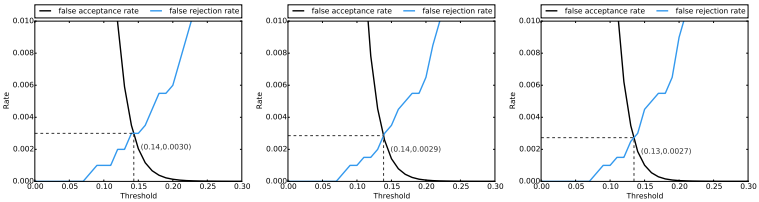



(g) $B = [63\text{Hz} - 2500\text{Hz}]$ (h) $B = [63\text{Hz} - 3150\text{Hz}]$ (i) $B = [63\text{Hz} - 4000\text{Hz}]$

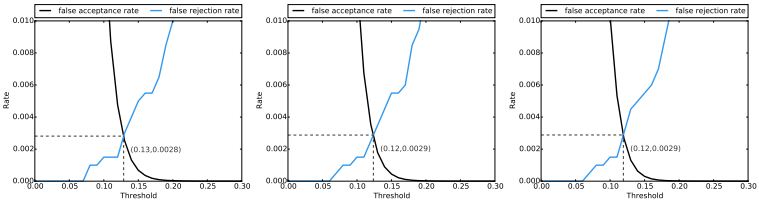
Figure A.2: False Rejection Rate and False Acceptance Rate as a function of the threshold $\tau_C$ for bands in the $B = [63\text{Hz} - [630\text{Hz} - 4\text{kHz}]]$ range




(a) $B = [80\text{Hz} - 630\text{Hz}]$ (b) $B = [80\text{Hz} - 800\text{Hz}]$ (c) $B = [80\text{Hz} - 1000\text{Hz}]$

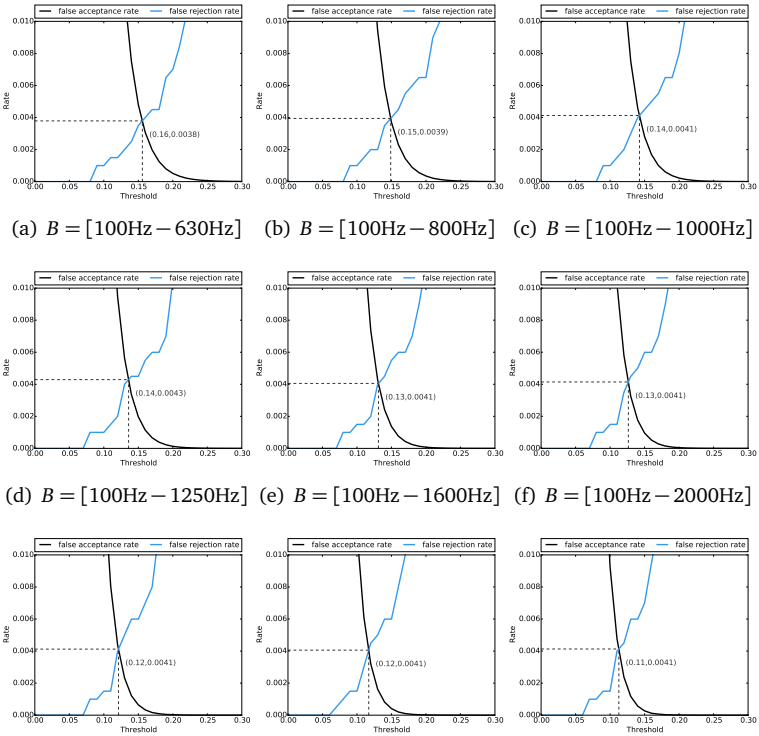



(d) $B = [80\text{Hz} - 1250\text{Hz}]$ (e) $B = [80\text{Hz} - 1600\text{Hz}]$ (f) $B = [80\text{Hz} - 2000\text{Hz}]$




(g) $B = [80\text{Hz} - 2500\text{Hz}]$ (h) $B = [80\text{Hz} - 3150\text{Hz}]$ (i) $B = [80\text{Hz} - 4000\text{Hz}]$

Figure A.3: False Rejection Rate and False Acceptance Rate as a function of the threshold $\tau_C$ for bands in the $B = [80\text{Hz} - [630\text{Hz} - 4\text{kHz}]]$ range




(a) $B = [100\text{Hz} - 630\text{Hz}]$ (b) $B = [100\text{Hz} - 800\text{Hz}]$ (c) $B = [100\text{Hz} - 1000\text{Hz}]$

(d) $B = [100\text{Hz} - 1250\text{Hz}]$ (e) $B = [100\text{Hz} - 1600\text{Hz}]$ (f) $B = [100\text{Hz} - 2000\text{Hz}]$

(g) $B = [100\text{Hz} - 2500\text{Hz}]$ (h) $B = [100\text{Hz} - 3150\text{Hz}]$ (i) $B = [100\text{Hz} - 4000\text{Hz}]$

Figure A.4: False Rejection Rate and False Acceptance Rate as a function of the threshold $\tau_C$ for bands in the $B = [100\text{Hz} - [630\text{Hz} - 4\text{kHz}]]$ range

# B System Usability Scale

We report the items of the System Usability Scale [42]. All items were answered with a 5-point Likert-scale from *Strongly Disagree* to *Strongly Agree*.

Q1 I think that I would like to use this system frequently.
Q2 I found the system unnecessarily complex.
Q3 I thought the system was easy to use.
Q4 I think that I would need the support of a technical person to be able to use this system.

Q5 I found the various functions in this system were well integrated.
Q6 I thought there was too much inconsistency in this system.
Q7 I would imagine that most people would learn to use this system very quickly.
Q8 I found the system very cumbersome to use.
Q9 I felt very confident using the system.
Q10 I needed to learn a lot of things before I could get going with this system.

# C Post-test Questionnaire

We report the items of the post-test questionnaire. All items were answered with a 5-point Likert-scale from *Strongly Disagree* to *Strongly Agree*.

Q1 I thought the audio-based method was quick.
Q2 I thought the code-based method was quick.
Q3 If Two-Factor Authentication were mandatory, I would use the audio-based method to log in.
Q4 If Two-Factor Authentication were mandatory, I would use the code-based method to log in.
Q5 If Two-Factor Authentication were optional, I would use the audio-based method to log in.
Q6 If Two-Factor Authentication were optional, I would use the code-based method to log in.
Q7 I would feel comfortable using the audio-based method at home.
Q8 I would feel comfortable using the audio-based method at my workplace.
Q9 I would feel comfortable using the audio-based method in a cafe.
Q10 I would feel comfortable using the audio-based method in a library.
Q11 I would feel comfortable using the code-based method at home.
Q12 I would feel comfortable using the code-based method at my workplace.
Q13 I would feel comfortable using the code-based method in a cafe.
Q14 I would feel comfortable using the code-based method in a library.

# D User Comments

This section lists some of the comments that participants added to their post-test questionnaire.

> "Sound-Proof is faster and automatic. Increased security without having to do more things"

> "I would use Sound-Proof, because it is less complicated and faster. I do not need to unlock the phone and open the application. In a public place it would feel a bit awkward unless it becomes widespread. Anyway, I am already logged in most websites that I use."
>
> "I like the audio idea, because what I hate the most about two-factor authentication is to have to take my phone out or find it around."
>
> "Sound-Proof is much easier. I am security-conscious and already use 2FA. I would be willing to switch to the audio-based method."
>
> "I already use Google 2SV and prefer it because I think it's more secure. However, Sound-Proof is seamless."

# E Emails

We reported the e-mails that were sent to participants during the duration of our user study on the effectiveness of personalized security indicators to thwart application phishing attacks.

## E.1 Advertising Email

We advertised a user study on the usability of a mobile banking application without mentioning any security-related feature. We also explicitly mention the duration of the user study as well as the number of tasks.

*Dear all,*

*We are a research group at ETH Zurich and we are looking for participants for a new user study. We test the usability of a mobile banking application.*

*If you will be selected to participate, you will receive a 20.- CHF voucher redeemable at online stores (Amazon and Coop). Alternatively, participants can receive 20.- CHF cash.*

*Study participants should speak English and have an iPhone (model 4S,5,5S,5C,6 and iOS version 7.1 or newer).*

*The study will last for approximately one week during which we will ask participants to complete 4 tasks, each taking roughly 5 minutes. Participants will also fill in a pre-test and post-test questionnaires. Participants must install our test application to their iPhone.*

*Your identity will remain confidential and we will not collect any private data on your iPhone. Your email address will be only used by us to contact you and will not be shared with third parties. Should you choose to participate, you may discontinue participation at any time.*

*If you wish to participate, please reply to this email and we will send you further instructions. We appreciate your help!*

## E.2 Email to Selected Participants

We report the email sent to all participants that completed the pre-test questionnaire. The email included a link to the application and the participant's credentials to access his account at SecBank.

*Dear participant,*

*Thank you for participating in this user study.*

*In this study, you act as a customer of a fictitious bank called SecBank. You have an account with SecBank where you have deposited fake money. To access your account, you use a mobile banking application called "SecBank".*

*You chose SecBank as your bank because it advertises its mobile banking platform as very secure. You want to ensure that your username and password to access your SecBank account do not fall into the wrong hands!*

*In the following 7 days, we will ask you to complete 4 tasks. Each task is a mobile banking operation to be carried out with the SecBank application. For each task, you will receive an email with detailed instructions. To complete each task, your iPhone must be connected to the Internet. Each task should take less than 5 minutes to complete. We kindly ask you to complete the task within 24 hours from receiving the email.*

*To start off, you will need to download and install the SecBank application on your iPhone. Username and password for your mobile banking account at SecBank are:*

*Username: -UNAME-*
*Password: -PWD-*

*The first time you start the application, please enter the email address to which you received this email: -EMAIL-*

*From your iPhone, visit https://www.smartphoneuserstudy.com/ to install the SecBank application.*

*Best regards,*

*Your SecBank team*

## E.3 Task Email

We report the email sent to complete task number one. (Emails for the remaining tasks closely resemble this one.)

*Dear participant,*

*It is time to perform the first mobile banking task with the SecBank application.*

*Task number 1 consists of starting the SecBank application on your iPhone and making a money transfer of $200 to "Anna Smith".*

*The account information of "Anna Smith" has already been saved to the list of your favourite recipients under "Money Transfer". Also we made sure that you have enough money in your account to complete the money transfer.*

*Once the SecBank application confirms the money transfer, the task is over.*
*Best regards,*
*Your SecBank team*

# F Instructions for participants in the experimental groups.

We report the priming text for participants in the experimental groups B, C and D. The paragraph was shown on the webpage where participants could download the application. The text is similar to the one provided by [31, 254] to the user of their e-banking platform.

*"As a major banking institution, SecBank is committed to prevent fraudulent smartphone applications from stealing your password. The SecBank mobile application uses a novel login mechanism based on personal images. The first time you login, you will be asked to pick a personal image from the photos stored in your phone. From that moment, the SecBank application will display your personal image every time it asks for your username and password. The presence of the correct personal image guarantees that you are not using a fraudulent look-alike application. You should enter your username and password only when you see your personal image. (Your personal image remains on your iPhone and is not sent to our servers.)"*

# G Post-test Questionnaire: Secure Setup of Indicators

We report the items of the post-test questionnaire for the user study on the indicator setup protocol (Section 9.6). All items were answered with a 5-point Likert-scale from *Strongly Disagree* to *Strongly Agree*.

Q1 The instruction sheet was clear and easy to understand
Q2 The task was easy to complete
Q3 I believe that I have successfully completed the task
Q4 I would use this mechanism to improve the security of my mobile banking application

# H Bank Letter

**Re:** *Secure set up of your personal image.*

Dear Customer,

The Android Bank application uses personal images to improve the security of its mobile banking platform. Before using our application, you must set up a personal image.

**Please read carefully the instructions below and follow them to securely set up your personal image. We advise you to read this letter until the end before starting.** Note that during the procedure you will need the following information:

| | |
|---|---|
| Service ID: | **ax:timross** |
| PIN: | **94455** |

Instructions to securely set up your personal image.

1. Double-tap the button “HOME” to start the *Secure Verifier* application.
2. Type in your **Service ID** and **PIN** shown above, then tap the button “Launch Appliction”.
3. The *Android Bank* application displays your PIN (**94455**). **Attention!** If the PIN is **missing** or **different** from **94455**, tap the button “HOME”. The installed application is not the legitimate Android Bank application. You **must not** select any personal image and should retain from using the application. The test is over.
4. If the PIN displayed is **identical** to your PIN (94455), tap the button “Select Personal Image”. The installed application is the legitimate Android Bank application. You can safely select your personal image.
5. Select one of the images displayed and tap the button “Confirm Selected Image”. The test is over.

Best regards,
Android Bank

# Bibliography


[1] 3GPP. 3GPP TS 23.040 - Technical realization of the Short Message Service (SMS), last access 2015. http://www.3gpp.org/ftp/Specs/html-info/23040.htm.

[2] D. Akhawe and A. P. Felt. Alice in warningland: A large-scale field study of browser security warning effectiveness. In *USENIX Security Symposium*, USENIX'13, 2013.

[3] aleph one. Smashing the stack for fun and profit, 1996. http://phrack.org/issues/49/14.html.

[4] Amazon Inc. Amazon Underground, last access 2015. http://www.amazon.com/b?node=11350978011.

[5] C. Amrutkar, P. Traynor, and P. C. van Oorschot. Measuring SSL indicators on mobile browsers: Extended life, or end of the road? In *International Information Security Conference*, ISC'12, 2012.

[6] J. Anderson, J. Bonneau, and F. Stajano. Inglorious Installers: Security in the Application Marketplace. In *Workshop on the Economics of Information Security*, WEIS '10, 2010.

[7] Apple Inc. Accelerate framework reference, last access 2015. https://developer.apple.com/library/mac/documentation/Accelerate/Reference/AccelerateFWRef/.

[8] Apple Inc. Apple Push Notification Service, last access 2015. https://developer.apple.com/library/ios/documentation/NetworkingInternet/Conceptual/RemoteNotificationsPG/Chapters/ApplePushService.html.

[9] Apple Inc. iOS Developer Library, last access 2015. https://developer.apple.com/library/ios/navigation/.

[10] Apple Inc. iOS security, last access 2015. https://www.apple.com/business/docs/iOS_Security_Guide.pdf.

[11] Apple Inc. iTunes AppStore, last access 2015. https://itunes.apple.com/us/store.

[12] Apple Inc. Kernel and Device Drivers Layer, last access 2015. https://developer.apple.com/library/mac/documentation/MacOSX/Conceptual/OSX_Technology_Overview/SystemTechnology/SystemTechnology.html.

[13] Apple Inc. Swift programming language, last access 2015. https://developer.apple.com/swift/.

[14] Apple Inc. The WebKit Open Source Project, last access 2015. https://www.webkit.org.

[15] W. A. Arentz and U. Bandara. Near ultrasonic directional data transfer for modern smartphones. In *International Conference on Pervasive and Ubiquitous Computing*, UbiComp'11, 2011.

[16] ARM Ltd. Building a Secure System using TrustZone Technology, 2009. http://www.arm.com.

[17] ARM Ltd. ARM and Thumb-2 Instruction Set Quick Reference Card, last access 2015. http://infocenter.arm.com/help/topic/com.arm.doc.qrc0001m/QRC0001_UAL.pdf.

[18] ARM Ltd. ARM Coretile Express, last access 2015. http://www.arm.com/products/tools/development-boards/versatile-express/coretile-express.php.

[19] ARM Ltd. ARM Motherboard Express, last access 2015. http://www.arm.com/products/tools/development-boards/versatile-express/motherboard-express.php.

[20] ARM Ltd. ARM NEON, last access 2015. http://www.arm.com/products/processors/technologies/neon.php.

[21] ARM Ltd. mbed TLS, last access 2015. https://tls.mbed.org.

[22] ARM Ltd. Project Ne10 - An Open Optimized Software Library Project for the ARM Architecture, last access 2015. http://projectne10.github.io/Ne10/.

[23] ARM Ltd. Securing the system with trustzone ready program, last access 2015. http://www.arm.com/files/pdf/Tech_seminar_TrustZone_v7_PUBLIC.pdf.

[24] Authy Inc. Authy, last access 2015. https://www.authy.com.

[25] J. Azema and G. Fayad. M-Shield mobile security technology: Making wireless secure, 2008. http://focus.ti.com/pdfs/wtbu/ti_mshield_whitepaper.pdf.

[26] T. Azim and I. Neamtiu. Targeted and depth-first exploration for systematic testing of Android apps. In *International Conference on Object Oriented Programming Systems Languages and Applications*, OOPSLA'13, 2013.

[27] M. Backes, T. Chen, M. Dürmuth, H. P. A. Lensch, and M. Welk. Tempest in a teapot: Compromising reflections revisited. In *IEEE Symposium on Security and Privacy*, SP'09, 2009.

[28] M. Backes, M. Dürmuth, and D. Unruh. Compromising reflections-or-how to read LCD monitors around the corner. In *IEEE Symposium on Security and Privacy*, SP'08, 2008.

[29] S. Baluja and M. Covell. Waveprint: Efficient wavelet-based audio fingerprinting. *Pattern Recognition*, 41(11), 2008.

[30] A. Bangor, P. T. Kortum, and J. T. Miller. An empirical evaluation of the system usability scale. *International Journal of Human-Computer Interaction*, 24(6), 2008.

[31] Bank of America. SiteKey Authentication, last access 2015. https://www.bankofamerica.com/privacy/online-mobile-banking-privacy/sitekey.go.

[32] Barclays. Mobile PINsentry, last access 2015. http://www.barclays.co.uk/BarclaysMobileBanking/MobilePINsentry/P1242616134119.

[33] D. Barrera, H. G. Kayacik, P. C. van Oorschot, and A. Somayaji. A methodology for empirical analysis of permission-based security models and its application to Android. In *ACM Conference on Computer and Communications Security*, CCS'10, 2010.

[34] A. Bianchi, J. Corbetta, L. Invernizzi, Y. Fratantonio, C. Kruegel, and G. Vigna. What the app is that? deception and countermeasures in the Android user interface. In *IEEE Symposium on Security and Privacy*, SP'15, 2015.

[35] BlackBerry. BlackBerry Java SDK - Security, last access 2015. http://docs.blackberry.com/en/developers/deliverables/29301/BlackBerry_Java_SDK-Development_Guide--1560066-0531040818-001-7.0_Beta-US.pdf.

[36] blexim. Basic integer overflows, 2002. http://phrack.org/issues/60/10.html.

[37] J. Bonneau, C. Herley, P. C. van Oorschot, and F. Stajano. The quest to replace passwords: A framework for comparative evaluation of web authentication schemes. In *IEEE Symposium on Security and Privacy*, SP'12, 2012.

[38] J. Brainard, A. Juels, R. L. Rivest, M. Szydlo, and M. Yung. Fourth-factor authentication: Somebody you know. In *ACM Conference on Computer and Communications Security*, CCS'06, 2006.

[39] C. Bravo-Lillo, L. F. Cranor, J. S. Downs, and S. Komanduri. Bridging the gap in computer security warnings: A mental model approach. *IEEE Security & Privacy*, 9(2), 2011.

[40] Broadcom Corp. 5G WiFi 802.11ac Client - BCM4354, last access 2015. https://www.broadcom.com/products/wireless-connectivity/wireless-lan/bcm4354.

[41] Broadcom Corp. Integrated Monolithic GPS And GLONASS Receiver IC - BCM47521, last access 2015. http://www.broadcom.com/products/wireless-connectivity/gps/bcm47521.

[42] J. Brooke. SUS - A quick and dirty usability scale. *Usability evaluation in industry*, 189(194), 1996.

[43] S. Bugiel, L. Davi, A. Dmitrienko, T. Fischer, and A.-R. Sadeghi. XManDroid: A new Android evolution to mitigate privilege escalation attacks. Technical Report TR-2011-04, Technische Universität Darmstadt, April 2011.

[44] S. Bugiel, L. Davi, A. Dmitrienko, T. Fischer, A.-R. Sadeghi, and B. Shastry. Towards taming privilege-escalation attacks on Android. In *Annual Network and Distributed System Security Symposium*, NDSS'12, 2012.

[45] S. Bugiel, L. Davi, A. Dmitrienko, S. Heuser, A.-R. Sadeghi, and B. Shastry. Practical and lightweight domain isolation on Android. In *ACM workshop on security and privacy in smartphones and mobile devices*, SPSM'11, 2011.

[46] J. Burns. Developing secure mobile applications for Android, 2008. `https://www.nccgroup.trust/globalassets/our-research/us/whitepapers/isec_securing_android_apps.pdf`.

[47] BusinessWire. Impermium study unearths consumer attitudes toward internet security, last access 2015. `http://www.businesswire.com/news/home/20130627005473/en/Impermium-Study-Unearths-Consumer-Attitudes-Internet-Security#.VXryixNrg4M`.

[48] C. Busold, A. Taha, C. Wachsmann, A. Dmitrienko, H. Seudié, M. Sobhani, and A.-R. Sadeghi. Smart keys for cyber-cars: secure smartphone-based NFC-enabled car immobilizer. In *ACM conference on Data and application security and privacy*, CODASPY'13, 2013.

[49] L. Cai and H. Chen. Touchlogger: Inferring keystrokes on touch screen from smartphone motion. In *USENIX Summit on Hot Topics in Security*, HotSec'11, 2011.

[50] Y. Cao, Y. Fratantonio, A. Bianchi, M. Egele, C. Kruegel, G. Vigna, and Y. Chen. Edgeminer: Automatically detecting implicit control flow transitions through the Android framework. In *Annual Network and Distributed System Security Symposium*, NDSS'15, 2015.

[51] L. Cavallaro, P. Saxena, and R. Sekar. On the limits of information flow techniques for malware analysis and containment. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, DIMVA'08, 2008.

[52] V. Chandrasekhar, M. Sharifi, and D. A. Ross. Survey and evaluation of audio fingerprinting schemes for mobile query-by-example applications. In *12th International Society for Music Information Retrieval Conference*, ISMIR'11, 2011.

[53] Chaos Computer Club. Chaos Computer Club breaks Apple TouchID, last access 2015. `http://www.ccc.de/en/updates/2013/ccc-breaks-apple-touchid`.

[54] Chaos Computer Club. How to fake fingerprints?, last access 2015. http://dasalte.ccc.de/biometrie/fingerabdruck_kopieren.en.

[55] R. Chatterjee, J. Bonneau, A. Juels, and T. Ristenpart. Cracking-resistant password vaults using natural language encoders. In *IEEE Symposium on Security and Privacy*, SP'15, 2015.

[56] A. Chaudhuri. Language-based security on Android. In *ACM Workshop on Programming Languages and Analysis for Security*, PLAS'09, 2009.

[57] L. Chen. Attacking WebKit applications by exploiting memory corruption bugs. CanSecWest, 2015. https://cansecwest.com/slides/2015/Liang_CanSecWest2015.pdf.

[58] T.-C. Chen, S. Dick, and J. Miller. Detecting visually similar web pages: Application to phishing detection. *ACM Trans. Internet Technol.*, 10(2), 2010.

[59] CherryPy Team. CherryPy - A Minimalistic Python Web Framework, last access 2015. http://www.cherrypy.org/.

[60] P. H. Chia, Y. Yamamoto, and N. Asokan. Is this app safe?: A large scale study on application permissions and risk signals. In *International Conference on World Wide Web*, WWW'12, 2012.

[61] E. Chin, A. P. Felt, K. Greenwood, and D. Wagner. Analyzing inter-application communication in Android. In *International Conference on Mobile Systems, Applications, and Services*, MobiSys'11, 2011.

[62] Comcetera Ltd. Number Portability Lookup, last access 2015. http://www.numberportabilitylookup.com/.

[63] A. Czeskis, M. Dietz, T. Kohno, D. S. Wallach, and D. Balfanz. Strengthening user authentication through opportunistic cryptographic identity assertions. In *ACM Conference on Computer and Communications Security*, CCS'12, 2012.

[64] Daniel C. Burnett and Adam Bergkvist and Cullen Jennings and Anant Narayanan. Media Capture and Streams (W3C Working Draft), last access 2015. http://www.w3.org/TR/mediacapture-streams/.

[65] A. Das, J. Bonneau, M. Caesar, N. Borisov, and X. Wang. The tangled web of password reuse. In *Annual Network and Distributed System Security Symposium*, NDSS'14, 2014.

[66] L. Davi, A. Dmitrienko, M. Egele, T. Fischer, T. Holz, R. Hund, S. Nürnberger, and A. Sadeghi. Mocfi: A framework to mitigate control-flow attacks on smartphones. In *Annual Network and Distributed System Security Symposium*, NDSS'12, 2012.

[67] L. Davi, A. Dmitrienko, A.-R. Sadeghi, and M. Winandy. Privilege escalation attacks on Android. In *International Conference on Information Security*, ISC'10, 2011.

[68] L. V. Davi, A. Dmitrienko, S. Nürnberger, and A.-R. Sadeghi. Gadge me if you can: Secure and efficient ad-hoc instruction-level randomization for x86 and arm. In *ACM Symposium on Information, Computer and Communications Security*, ASIACCS '13, 2013.

[69] X. de Carné de Carnavalet and M. Mannan. From very weak to very strong: Analyzing password-strength meters. In *Annual Network and Distributed System Security Symposium*, NDSS'14, 2014.

[70] G. Delugré. Reverse-engineering a qualcomm baseband. In *Chaos Communication Congress*, CCC'11, 2011.

[71] D. E. Denning and P. J. Denning. Data security. *ACM Comput. Surv.*, 11(3), Sept. 1979.

[72] R. Dhamija and J. D. Tygar. The battle against phishing: Dynamic security skins. In *Symposium on Usable Privacy and Security*, SOUPS'05, 2005.

[73] R. Dhamija, J. D. Tygar, and M. Hearst. Why phishing works. In *SIGCHI Conference on Human Factors in Computing Systems*, CHI'06, 2006.

[74] T. P. Diakos, J. A. Briffa, T. W. C. Brown, and S. Wesemeyer. Eavesdropping near-field contactless payments: A quantitative analysis. *The Journal of Engineering*, 2013.

[75] Digital Trends. Do not use iMessage chat for Android, it's not safe, last access 2015. `http://www.digitaltrends.com/mobile/imessage-chat-android-security-flaw/`.

[76] A. Dmitrienko, A.-R. Sadeghi, S. Tamrakar, and C. Wachsmann. SmartTokens: Delegable Access Control with NFC-enabled Smartphones. In *International Conference on Trust and Trustworthy Computing*, TRUST'11, 2011.

[77] Doug Gross. 50 million compromised in Evernote hack, last access 2015. `http://edition.cnn.com/2013/03/04/tech/web/evernote-hacked/`.

[78] A. Dua, N. Bulusu, and W. chang Feng. Towards trustworthy participatory sensing. In *USENIX Summit on Hot Topics in Security*, HotSec'09, 2009.

[79] Duo Security, Inc. Duo Push, last access 2015. `https://www.duosecurity.com/product/methods/duo-mobile`.

[80] C. Dwork. Differential privacy. In *International Colloquium on Automata, Languages and Programming*, ICALP'06, 2006.

[81] S. Eberz, K. B. Rasmussen, V. Lenders, and I. Martinovic. Preventing lunchtime attacks: Fighting insider threats with eye movement biometrics. In *Annual Network and Distributed System Security Symposium*, NDSS'15. The Internet Society, 2015.

[82] M. Egele, C. Kruegel, E. Kirda, and G. Vigna. PiOS: Detecting privacy leaks in iOS applications. In *Annual Network and Distributed System Security Symposium*, NDSS'11, 2011.

[83] S. Egelman, L. F. Cranor, and J. Hong. You've been warned: an empirical study of the effectiveness of web browser phishing warnings. In *SIGCHI Conference on Human Factors in Computing Systems*, CHI'08, 2008.

[84] S. Egelman, J. King, R. C. Miller, N. Ragouzis, and E. Shehan. Security user studies: Methodologies and best practices. In *Extended Abstracts on Human Factors in Computing Systems*, CHI EA'07, 2007.

[85] EMC Corporation. RSA SecurID, last access 2015. `http://www.emc.com/security/rsa-securid/index.htm`.

[86] EMC Inc. RSA SecurID, last access 2015. `https://www.emc.com/security/rsa-securid.htm/`.

[87] EMV. Integrated Circuit Card Specifications for Payment Systems, Book 1-4, Version 4.3, last access 2015. https://www.emvco.com/specifications.aspx?id=223.

[88] Encap Security. Encap Security, last access 2015. https://www.encapsecurity.com/.

[89] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth. TaintDroid: an information-flow tracking system for realtime privacy monitoring on smartphones. In *USENIX Conference on Operating Systems Design and Implementation*, OSDI'10, 2010.

[90] W. Enck, M. Ongtang, and P. McDaniel. On lightweight mobile phone application certification. In *ACM Conference on Computer and Communications Security*, CCS'09, 2009.

[91] Entersekt. Transakt, last access 2015. https://www.entersekt.com/Technology-Transakt.

[92] S. Esser. iOS 6 exploitation 280 days later. CanSecWest, 2013. https://cansecwest.com/slides/2013/CSW2013_StefanEsser_iOS6_Exploitation_280_Days_Later.pdf.

[93] ETSI. Enhanced Data rates for Global Evolution, EDGE, last access 2015. http://www.etsi.org/index.php/technologies-clusters/technologies/mobile/edge.

[94] ETSI. General Packet Radio Service, GPRS, last access 2015. http://www.etsi.org/index.php/technologies-clusters/technologies/mobile/gprs.

[95] ETSI. Long Term Evolution, LTE, last access 2015. http://www.etsi.org/technologies-clusters/technologies/mobile/long-term-evolution.

[96] ETSI. Mobile Technologies GSM, last access 2015. http://www.etsi.org/technologies-clusters/technologies/mobile/gsm.

[97] European Central Bank. Report on card fraud, July 2012. http://www.ecb.int/pub/pdf/other/cardfraudreport201207en.pdf.

[98] Express Logic Inc. ThreadX, last access 2015. http://rtos.com/products/threadx/.

[99] eyeLock Inc. Advanced iris authentication and recognition solutions, last access 2015. http://www.eyelock.com.

[100] F-secure. Warning on possible Android mobile trojans, 2010. www.f-secure.com/weblog/archives/00001852.html.

[101] A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner. Android permissions demystified. In *ACM Conference on Computer and Communications Security*, CCS'11, 2011.

[102] A. P. Felt, M. Finifter, E. Chin, S. Hanna, and D. Wagner. A survey of mobile malware in the wild. In *ACM workshop on security and privacy in smartphones and mobile devices*, SPSM'11, 2011.

[103] A. P. Felt, E. Ha, S. Egelman, A. Haney, E. Chin, and D. Wagner. Android permissions: User attention, comprehension, and behavior. In *Symposium on Usable Privacy and Security*, SOUPS'12, 2012.

[104] A. P. Felt and D. Wagner. Phishing on mobile devices. In *Web 2.0 Security and Privacy Workshop*, W2SP'11, 2011.

[105] I. Fette and A. Melnikov. The WebSocket protocol (RFC 6455), 2011. http://tools.ietf.org/html/rfc6455.

[106] I. Fette, N. Sadeh, and A. Tomasic. Learning to detect phishing emails. In *International Conference on World Wide Web*, WWW'07, 2007.

[107] FIDO Alliance. Fido U2F specifications, last access 2015. https://fidoalliance.org/specifications/.

[108] Forbes. Alleged "Nazi" Android FBI Ransomware Mastermind Arrested In Russia, last access 2015. http://www.forbes.com/sites/thomasbrewster/2015/04/13/alleged-nazi-android-fbi-ransomware-mastermind-arrested-in-russia/.

[109] P. Fourez and Mastercard International Inc. Location controls on payment card transactions - Patent No. WO/2011/022062, 2011. http://patentscope.wipo.int/search/en/WO2011022062.

[110] M. Frank, R. Biedert, E. Ma, I. Martinovic, and D. Song. Touchalytics: On the applicability of touchscreen input as a behavioral biometric for continuous authentication. *Information Forensics and Security, IEEE Transactions on*, 8(1), Jan 2013.

[111] Fujitsu. PalmSecure, last access 2015. `http://www.fujitsu.com/us/Images/palmsecure_datasheet.pdf`.

[112] S. Garera, N. Provos, M. Chew, and A. D. Rubin. A framework for detection and measurement of phishing attacks. In *ACM Workshop on Recurring Malcode*, WORM'07, 2007.

[113] General Dynamics Mission Systems. OKL4 Microvisor, last access 2015. `http://gdmissionsystems.com/cyber/products/trusted-computing-cross-domain/microvisor-products/`.

[114] Giesecke & Devrient GmbH. G&D Makes Mobile Terminal Devices Even More Secure with New Version of Smart Card in MicroSD Format, last access 2015. `http://www.gi-de.com/en/about_g_d/press/press_releases/G%26D-Makes-Mobile-Terminal-Devices-Secure-with-New-MicroSD-Card-g3592.jsp`.

[115] P. Gilbert, L. P. Cox, J. Jung, and D. Wetherall. Toward trustworthy mobile sensing. In *ACM International Workshop on Mobile Computing Systems and Applications*, HotMobile'10, 2010.

[116] P. Gilbert, J. Jung, K. Lee, H. Qin, D. Sharkey, A. Sheth, and L. P. Cox. Youprove: authenticity and fidelity in mobile sensing. In *International Conference on Embedded Networked Sensor Systems*, SenSys'11, 2011.

[117] C. G. Girling. Covert Channels in LAN's. *IEEE Transactions on Software Engineering*, 13(2), 1987.

[118] V. Gligor. A guide to understanding covert channel analysis of trusted systems, version 1 (light pink book). NCSC-TG-030, Library No. S-240,572, 1993. `http://www.radium.ncsc.mil/tpep/library/rainbow/NCSC-TG-030.pdf`.

[119] V. D. Gligor, E. L. Burch, C. S. Chandersekaran, R. S. Chapman, L. J. Dotterer, M. S. Hecht, W.-D. Jiang, G. L. Luckenbaugh, and N. Vasudevan. On the design and the implementation of secure xenix workstations. In *IEEE Symposium on Security and Privacy*, SP'86, 1986.

[120] GlobalPlatform. Card Specification Version 2.2.1, 2011. http://www.globalplatform.org/specificationscard.asp.

[121] GlobalPlatform. GlobalPlatform Device Specifications, last access 2015. http://www.globalplatform.org/specificationsdevice.asp.

[122] G. Gong. Fuzzing android system services by binder call to escalate privilege. BlackHat USA, 2015. https://www.blackhat.com/docs/us-15/materials/us-15-Gong-Fuzzing-Android-System-Services-By-Binder-Call-To-Escalate-Privilege-wp.pdf.

[123] Google Inc. Android 4.1 APIs - Jelly Bean, last access 2015. http://developer.android.com/about/versions/jelly-bean.html.

[124] Google Inc. Android and Security, last access 2015. http://googlemobile.blogspot.ch/2012/02/android-and-security.html.

[125] Google Inc. Android Open Source Project, last access 2015. https://source.android.com/index.html.

[126] Google Inc. Android OS, last access 2015. http://developer.android.com/.

[127] Google Inc. Android Security, last access 2015. https://source.android.com/devices/tech/security/index.html.

[128] Google Inc. ART and Dalvik, last access 2015. https://source.android.com/devices/tech/dalvik/.

[129] Google Inc. Google 2-Step Verification, last access 2015. http://www.google.com/landing/2step/.

[130] Google Inc. Google Cloud Messaging for Android, last access 2015. https://developer.android.com/google/gcm/index.html.

[131] Google Inc. Google play, last access 2015. https://play.google.com/store.

[132] Google Inc. Google wallet, last access 2015. http://www.google.com/wallet/.

[133] Google Inc. OpenGL ES, last access 2015. http://developer.android.com/guide/topics/graphics/opengl.html.

[134] Google Inc. Protect against harmful apps, last access 2015. https://support.google.com/accounts/answer/2812853.

[135] Google Inc. SlickLogin, last access 2015. http://www.slicklogin.com/.

[136] Google Inc. The Chromium Projects - Blink, last access 2015. http://www.chromium.org/blink.

[137] Google Inc. WebRTC, last access 2015. http://www.webrtc.org/.

[138] GRSecurity. The GRSecurity project, last access 2015. http://grsecurity.net/features.php.

[139] GSM Association. Requirements for Single Wire Protocol NFC Handsets, 2011. http://www.gsma.com/mobilenfc/.

[140] R. Guida, R. Stahl, T. Bunt, G. Secrest, and J. Moorcones. Deploying and using public key technology: lessons learned in real life. *IEEE Security Privacy*, 2(4), 2004.

[141] N. Gunson, D. Marshall, H. Morton, and M. A. Jack. User perceptions of security and usability of single-factor and two-factor authentication in automated telephone banking. *Computers & Security*, 30(4), 2011.

[142] J. Haitsma, T. Kalker, and J. Oostveen. An efficient database search strategy for audio fingerprinting. In *Workshop on Multimedia Signal Processing*, MMSP'02, 2002.

[143] J. A. Halderman, B. Waters, and E. W. Felten. A convenient method for securely managing passwords. In *International Conference on World Wide Web*, WWW'05, 2005.

[144] T. Halevi, D. Ma, N. Saxena, and T. Xiang. Secure proximity detection for NFC devices based on ambient sensor data. In *European Symposium on Research in Computer Security*, ESORICS'12, 2012.

[145] T. Harada, T. Horie, and K. Tanaka. Task oriented management obviates your onus on linux (TOMOYO Linux), 2004. Linux Conference.

[146] E. Haselsteiner and K. Breitfuss. Security in near field communication (NFC). In *RFIDSec '06*, 2006.

[147] M. Hazas and A. Ward. A novel broadband ultrasonic location system. In *International Conference on Pervasive and Ubiquitous Computing*, UbiComp'02, 2002.

[148] A. Herzberg and R. Margulies. My authentication album: Adaptive images-based login mechanism. In *Information Security and Privacy Research*, 2012.

[149] J. Hong. The state of phishing attacks. *Commun. ACM*, 55(1), 2012.

[150] W.-M. Hu. Reducing timing channels with fuzzy time. In *IEEE Symposium on Security and Privacy*, SP'91, 1991.

[151] W.-M. Hu. Lattice scheduling and covert channels. In *IEEE Symposium on Security and Privacy*, SP'92, 1992.

[152] IDC Corporate USA. Smartphone OS Market Share, 2015 Q2, last access 2015. `http://www.idc.com/prodserv/smartphone-os-market-share.jsp`.

[153] A. Inc. 1Password, last access 2015. `https://agilebits.com/onepassword`.

[154] InvenSense Inc. ICM-20608-G - 6-axis MotionTracking device, last access 2015. `http://www.invensense.com/products/motion-tracking/6-axis/icm-20608-2/`.

[155] D. Jablon. The SPEKE Password-Based Key Agreement Methods, 2003. `http://tools.ietf.org/html/draft-jablon-speke-02`.

[156] C. Jackson, D. R. Simon, D. S. Tan, and A. Barth. An evaluation of extended validation and picture-in-picture phishing attacks. In *International Conference on Financial cryptography and International conference on Usable Security*, FC'07/USEC'07, 2007.

[157] C. Jackson, D. R. Simon, D. S. Tan, and A. Barth. An evaluation of extended validation and picture-in-picture phishing attacks. In *Financial Cryptography and Data Security*, FC'07, 2007.

[158] Jaikumar Vijayan. Hackers crack more than 60% of breached LinkedIn passwords, last access 2015. http://www.computerworld.com/article/2504078/cybercrime-hacking/hackers-crack-more-than-60--of-breached-linkedin-passwords.html.

[159] S. Jana and V. Shmatikov. Memento: Learning secrets from process footprints. In *IEEE Symposium on Security and Privacy*, SP'12, 2012.

[160] Z. Jorgensen and T. Yu. On mouse dynamics as a behavioral biometric for authentication. In *ACM Symposium on Information, Computer and Communications Security*, ASIACCS'11, 2011.

[161] A. Juels and R. L. Rivest. Honeywords: making password-cracking detectable. In *ACM Conference on Computer and Communications Security*, CCS'13, 2013.

[162] JVL Ventures, LLC. Softcard, last access 2015. https://en.wikipedia.org/wiki/Softcard.

[163] N. Karapanos and S. Capkun. On the effective prevention of TLS man-in-the-middle attacks in web applications. In *USENIX Security Symposium*, USENIX'14, 2014.

[164] K. Kostiainen and N. Asokan. Credential life cycle management in open credential platforms (short paper). In *ACM workshop on Scalable trusted computing*, STC'11, 2011.

[165] K. Kostiainen, N. Asokan, and A. Afanasyeva. Towards user-friendly credential transfer on open credential platforms. In *International conference on Applied cryptography and network security*, ACNS'11, 2011.

[166] K. Kostiainen, J.-E. Ekberg, N. Asokan, and A. Rantala. On-board credentials with open provisioning. In *ACM Symposium on Information, Computer and Communications Security*, ASIACCS'09, 2009.

[167] K. Kostiainen, E. Reshetova, J.-E. Ekberg, and N. Asokan. Old, new, borrowed, blue – a perspective on the evolution of mobile platform security architectures. In *ACM conference on Data and application security and privacy*, CODASPY'11, 2011.

[168] M. Kumar, T. Garfinkel, D. Boneh, and T. Winograd. Reducing shoulder-surfing by using gaze-based password entry. In *Symposium on Usable Privacy and Security*, SOUPS'07, 2007.

[169] J.-F. Lalande and S. Wendzel. Hiding privacy leaks in Android applications using low-attention raising covert channels. In *2013 International Conference on Availability, Reliability and Security*, ARES'13, 2013.

[170] B. W. Lampson. A note on the confinement problem. *Commun. ACM*, 16(10), Oct. 1973.

[171] J. Lee, L. Bauer, and M. L. Mazurek. Studying the effectiveness of security images in Internet banking. *IEEE Internet Computing*, 13(1), 2014.

[172] Lee Munson. Yahoo prompts password reset after mass attack on email service, last access 2015. `https://nakedsecurity.sophos.com/2014/01/31/yahoo-prompts-password-reset-after-mass-attack-on-email-service/`.

[173] Legion of the Bouncy Castle. Bouncy Castle Crypto APIs, last access 2015. `http://www.bouncycastle.org`.

[174] C. Lever, M. Antonakakis, B. Reaves, P. Traynor, and W. Lee. The core of the matter: Analyzing malicious traffic in cellular carriers. In *Annual Network and Distributed System Security Symposium*, NDSS'13, 2013.

[175] A. Libonati, J. M. McCune, and M. K. Reiter. Usability testing a malware-resistant input mechanism. In *Annual Network and Distributed System Security Symposium*, NDSS'11, 2011.

[176] C.-C. Lin, H. Li, X. Zhou, and X. Wang. Screenmilker: How to milk your Android screen for secrets. In *Annual Network and Distributed System Security Symposium*, NDSS'14, 2014.

[177] A. M. Lineberry. These aren't the permissions you're looking for. BlackHat USA, 2010.

[178] S. B. Lipner. A comment on the confinement problem. In *ACM Symposium on Operating Systems Principles*, SOSP'75, 1975.

[179] H. Liu, S. Saroiu, A. Wolman, and H. Raj. Software abstractions for trusted sensors. In *International Conference on Mobile Systems, Applications, and Services*, MobiSys'12, 2012.

[180] Lookout Inc. 2014 Mobile Threat Report, last access 2015. `https://www.lookout.com/resources/reports/mobile-threat-report`.

[181] M2SYS Technology. M2-PalmVein - Secure Palm Vein Scanner, last access 2015. `http://www.m2sys.com/palm-vein-reader/`.

[182] P. MacKenzie. On the security of the speke password-authenticated key exchange protocol. In *In IACR ePrint archive*, 2001. `http://eprint.iacr.org/2001/057`.

[183] MacRumors. "Masque Attack" vulnerability allows malicious third-party iOS apps to masquerade as legitimate apps, 2014. `http://www.macrumors.com/2014/11/10/masque-attack-ios-vulnerability/`.

[184] L. Malisa. Detecting mobile application spoofing attacks by leveraging user visual similarity perception, 2015. Cryptology ePrint Archive: Report 2015/709.

[185] M. Mannan, B. H. Kim, A. Ganjali, and D. Lie. Unicorn: two-factor attestation for data security. In *ACM Conference on Computer and communications security*, CCS'11, 2011.

[186] C. Marforio, N. Karapanos, C. Soriente, K. Kostiainen, and S. Capkun. Secure enrollment and practical migration for mobile trusted execution environments. In *ACM workshop on security and privacy in smartphones and mobile devices*, SPSM'13, 2013.

[187] Mastercard International Inc. Mastercard Developer Zone, last access 2015. `https://developer.mastercard.com/`.

[188] R. J. Masti, D. Rai, A. Ranganathan, C. Müller, L. Thiele, and S. Capkun. Thermal covert channels on multi-core platforms. In *USENIX Security Symposium*, USENIX'15, 2015.

[189] M.-E. Maurer and L. Höfer. Sophisticated Phishers Make More Spelling Mistakes: Using URL Similarity against Phishing. In *International Conference on Cyberspace Safety and Security*, CSS'12, 2012.

[190] J. M. McCune, A. Perrig, and M. K. Reiter. Safe passage for passwords and other sensitive data. In *Annual Network and Distributed System Security Symposium*, NDSS'09, 2009.

[191] Mentor Graphics. Nucleus RTOS, last access 2015. `http://www.mentor.com/embedded-software/nucleus/`.

[192] Microsoft Inc. Bringing interoperable real-time communications to the web, last access 2015. `http://blogs.skype.com/2014/10/27/bringing-interoperable-real-time-communications-to-the-web/`.

[193] Microsoft Inc. Security for Windows Phone, last access 2015. `https://msdn.microsoft.com/library/windows/apps/ff402533(v=vs.105).aspx`.

[194] Microsoft Inc. Security for Windows Phone 8, last access 2015. `https://msdn.microsoft.com/en-us/library/windows/apps/ff402533(v=vs.105).aspx`.

[195] Mozilla. Location-Aware Browsing, last access 2015. `https://www.mozilla.org/en-US/firefox/geolocation/`.

[196] S. K. Nair, P. N. D. Simpson, B. Crispo, and A. S. Tanenbaum. A virtual machine based information flow control system for policy enforcement. *Electron. Notes Theor. Comput. Sci.*, 197(1), Feb. 2008.

[197] Network Time Foundation. NTP: The Network Time Protocol, last access 2015. `http://www.ntp.org/`.

[198] Y. Niu, F. Hsu, and H. Chen. iPhish: Phishing Vulnerabilities on Consumer Electronics. In *Conference on Usability, Psychology, and Security*, UPSEC'08, 2008.

[199] No Author Given. United Bankers' Bank Authenticates Customers Online. *Biometric Technology Today*, 12(6), 2004.

[200] No Author Given. Bank of Utah Adopts Keystroke Dynamics. *Biometric Technology Today*, 15(5), 2007.

[201] Nuance Communications Inc. Voice biometrics, last access 2015. `http://www.nuance.com/for-business/customer-service-solutions/voice-biometrics/index.htm`.

[202] J. Oberheide. Android Hax. SummerCon 2010, June 2010. http://jon.oberheide.org/files/summercon10-androidhax-jonoberheide.pdf.

[203] J. Oberheide and F. Jahanian. When mobile is harder than fixed (and vice versa): demystifying security challenges in mobile environments. In *Workshop on Mobile Computing Systems and Applications*, HotMobile'10, 2010.

[204] J. Oberheide and C. Miller. Dissecting the Android Bouncer, last access 2015. https://jon.oberheide.org/files/summercon12-bouncer.pdf.

[205] M. Ongtang, S. McLaughlin, W. Enck, and P. McDaniel. Semantically rich application-centric security in Android. In *Annual Computer Security Applications Conference*, ACSAC'09, 2009.

[206] osmocomBB. The OsmocomBB project, last access 2015. http://bb.osmocom.org.

[207] OWASP Foundation. Man-in-the-browser attack, last access 2015. https://www.owasp.org/index.php/Man-in-the-browser_attack.

[208] E. Owusu, J. Han, S. Das, A. Perrig, and J. Zhang. Accessory: Password inference using accelerometers on smartphones. In *Workshop on Mobile Computing Systems & Applications*, HotMobile'12, 2012.

[209] R. Pandita, X. Xiao, W. Yang, W. Enck, and T. Xie. Whyper: Towards automating risk assessment of mobile applications. In *USENIX Security Symposium*, USENIX'13, 2013.

[210] F. S. Park, C. Gangakhedkar, and P. Traynor. Leveraging cellular infrastructure to improve fraud prevention. In *Annual Computer Security Applications Conference*, ACSAC'09, 2009.

[211] B. Parno. Bootstrapping trust in a "trusted" platform. In *USENIX Summit on Hot Topics in Security*, HotSec'08, 2008.

[212] B. Parno, C. Kuo, and A. Perrig. Phoolproof phishing prevention. In *International Conference on Financial Cryptography and Data Security*, FC'06, 2006.

[213] PayPal. PayPal Security Key, last access 2015. https://www.paypal.com/webapps/mpp/security/security-protections.

[214] F. A. Petitcolas, R. J. Anderson, and M. G. Kuhn. Information hiding—a survey. *Proceedings of the IEEE, special issue on protection of multimedia content*, 87(7), July 1999.

[215] T. Petsas, G. Tsirantonakis, E. Athanasopoulos, and S. Ioannidis. Two-factor authentication: Is the world ready?: Quantifying 2FA adoption. In *European Workshop on System Security*, EuroSec'15, 2015.

[216] G. Portokalidis, P. Homburg, K. Anagnostakis, and H. Bos. Paranoid Android: versatile protection for smartphones. In *Annual Computer Security Applications Conference*, ACSAC'10, 2010.

[217] R. Pries, T. Hobfeld, and P. Tran-Gia. On the suitability of the short message service for emergency warning systems. In *Vehicular Technology Conference*, VTC'06, 2006.

[218] Proxama. ARM Click and Pay, last access 2015. http://www.proxama.com/resource-centre/case-studies/arm-click-pay/.

[219] R. Raguram, A. M. White, D. Goswami, F. Monrose, and J. Frahm. iSpy: Automatic reconstruction of typed input from compromising reflections. In *ACM Conference on Computer and Communications Security*, CCS'11, 2011.

[220] H. Raj, D. Robinson, T. Tariq, P. England, S. Saroiu, and A. Wolman. Credo: Trusted computing for guest vms with a commodity hypervisor. Technical Report MSR-TR-2011-130, Microsoft Research, 2011.

[221] K. B. Rasmussen and S. Capkun. Realization of RF distance bounding. In *USENIX Security Symposium*, USENIX'10, 2010.

[222] K. B. Rasmussen, M. Roeschlin, I. Martinovic, and G. Tsudik. Authentication using pulse-response biometrics. In *Annual Network and Distributed System Security Symposium*, NDSS'14. The Internet Society, 2014.

[223] N. Ratha, R. Bolle, V. Pandit, and V. Vaish. Robust fingerprint authentication using local structural similarity. In *IEEE Workshop on Applications of Computer Vision*, 2000.

[224] R. A. Rensink. Change detection. *Annual review of psychology*, 53(1), 2002.

[225] riq and gera. Advances in format string exploitation, 2002. `http://phrack.org/issues/59/7.html`.

[226] A. Rodríguez Valiente, A. Trinidad, J. R. García Berrocal, C. Górriz, and R. Ramírez Camacho. Extended high-frequency (9-20 khz) audiometry reference thresholds in 645 healthy subjects. *International Journal of Audiology*, 53(8), 2014.

[227] B. Ross, C. Jackson, N. Miyake, D. Boneh, and J. C. Mitchell. Stronger password authentication using browser extensions. In *USENIX Security Symposium*, USENIX'05, 2005.

[228] D. A. Russell, J. P. Titlow, and Y.-J. Bemmen. Acoustic monopoles, dipoles, and quadrupoles: An experiment revisited. *American Journal of Physics*, 67(8), 1999.

[229] G. Rydstedt, B. Gourdin, E. Bursztein, and D. Boneh. Framing attacks on smart phones and dumb routers: Tap-jacking and geo-localization attacks. In *USENIX Workshop on Offensive Technologies*, WOOT'10, 2010.

[230] A. Sabzevar and A. Stavrou. Universal multi-factor authentication using graphical passwords. In *IEEE International Conference on Signal Image Technology and Internet Based Systems*, SITIS'08, 2008.

[231] S. Saroiu and A. Wolman. I am a sensor, and i approve this message. In *ACM International Workshop on Mobile Computing Systems and Applications*, HotMobile'10, 2010.

[232] S. E. Schechter, R. Dhamija, A. Ozment, and I. Fischer. The emperor's new security indicators. In *IEEE Symposium on Security and Privacy*, SP'07, 2007.

[233] R. Schlegel, K. Zhang, X. Zhou, M. Intwala, A. Kapadia, and X. Wang. Soundcomber: A Stealthy and Context-Aware Sound Trojan for Smartphones. In *Annual Network and Distributed System Security Symposium*, NDSS'11, 2011.

[234] D. Schürmann and S. Sigg. Secure communication based on ambient audio. *IEEE Trans. Mob. Comput.*, 12(2), 2013.

[235] SD Association. smartSD Memory Cards, last access 2015. `https://www.sdcard.org/developers/overview/ASSD/smartsd/`.

[236] Secure List. The Android Trojan Svpeng now capable of mobile phishing, last access 2015. `http://www.securelist.com/en/blog/8138/The_Android_Trojan_Svpeng_now_capable_of_mobile_phishing`.

[237] D. Sehr, R. Muth, C. Biffle, V. Khimenko, E. Pasko, K. Schimpf, B. Yee, and B. Chen. Adapting software fault isolation to contemporary cpu architectures. In *USENIX Security Symposium*, USENIX'10, 2010.

[238] M. Selhorst, C. Stüble, F. Feldmann, and U. Gnaida. Towards a trusted mobile desktop. In *International Conference on Trust and Trustworthy Computing*, TRUST'10, 2010.

[239] A. B. Shaffer, M. Auguston, C. E. Irvine, and T. E. Levin. A security domain model to assess software for exploitable covert channels. In *ACM Workshop on Programming Languages and Analysis for Security*, PLAS'08, 2008.

[240] S. Shepherd. Continuous authentication by analysis of keyboard typing characteristics. In *European Convention on Security and Detection*, 1995.

[241] M. Shirvanian, S. Jarecki, N. Saxena, and N. Nathan. Two-factor authentication resilient to server compromise using mix-bandwidth devices. In *Annual Network and Distributed System Security Symposium*, NDSS'14, 2014.

[242] B. Shrestha, N. Saxena, H. Truong, and N. Asokan. Drone to the rescue: Relay-resilient authentication using ambient multi-sensing. In *Financial Cryptography and Data Security*, FC '14, 2014.

[243] Sierraware. Open Virtualization - ARM TrustZone and ARM Hypervisor Open Source Software, last access 2015. `http://www.openvirtualization.org/`.

[244] S. Smalley, NSA, and Trust Mechanisms (R2X). SEAndroid, last access 2015. `http://seandroid.bitbucket.org`.

[245] J. Sorber, M. Shin, R. A. Peterson, and D. Kotz. Plug-n-trust: practical trusted sensing for mhealth. In *International Conference on Mobile Systems, Applications, and Services*, MobiSys'12, 2012.

[246] SQLite Development Team. SQLite, last access 2015. `http://www.sqlite.org`.

[247] StatCounter. StatCounter global stats, last access 2015. `http://gs.statcounter.com/`.

[248] J. Sunshine, S. Egelman, H. Almuhimedi, N. Atri, and L. F. Cranor. Crying wolf: An empirical study of ssl warning effectiveness. In *USENIX Security Symposium*, USENIX'09, 2009.

[249] S. Tamrakar and J.-E. Ekberg. Tapping and tripping with nfc. In *International Conference on Trust and Trustworthy Computing*, TRUST'13, 2013.

[250] S. Tamrakar, J.-E. Ekberg, and N. Asokan. Identity verification schemes for public transport ticketing with nfc phones. In *ACM Workshop on Scalable Trusted Computing* , STC'11, 2011.

[251] Telegraph. EU to end mobile roaming charges next year, June 2013. `http://www.telegraph.co.uk/finance/newsbysector/mediatechnologyandtelecoms/telecoms/10119159/EU-to-end-mobile-roaming-charges-next-year.html`.

[252] The American National Standards Insitute. ANSI s1.11-2004 - Specification for octave-band and fractional-octave-band analog and digital filters, 2004.

[253] The Lookout Blog. Lookout's privacy advisor protects your private information, 2010. `http://blog.mylookout.com/2010/11/lookout%E2%80%99s-privacy-advisor-protects-your-private-information/`.

[254] The Vanguard Group. Vanguard Enhanced Logon, last access 2015. `http://www.vanguard.com/us/content/Home/RegEnhancedLogOnContent.jsp`.

[255] M. Tiwari, J. K. Oberg, X. Li, J. Valamehr, T. Levin, B. Hardekopf, R. Kastner, F. T. Chong, and T. Sherwood. Crafting a usable microkernel, processor, and I/O system with strict and provable information

flow security. In *Annual International Symposium on Computer Architecture*, ISCA'11, 2011.

[256] H. T. T. Truong, X. Gao, B. Shrestha, N. Saxena, N. Asokan, and P. Nurmi. Comparing and fusing different sensor modalities for relay attack resistance in zero-interaction authentication. In *International Conference on Pervasive Computing and Communications*, PerCom'14, 2014.

[257] H. T. T. Truong, E. Lagerspetz, P. Nurmi, A. J. Oliner, S. Tarkoma, N. Asokan, and S. Bhattacharya. The company you keep: Mobile malware infection rates and inexpensive risk indicators. In *International Conference on World Wide Web*, WWW'14, 2014.

[258] Trustonic. Trustonic for Samsung KNOX, last access 2015. `http://www.trustonic.com/assets/common/Trustonic_for_Samsung_Knox_Brochure.pdf`.

[259] C.-R. Tsai, V. D. Gligor, and C. S. Shandersekaran. On the identification of covert storage channels in secure systems. *IEEE Transactions on Software Engineering*, 16, June 1990.

[260] J. D. Tygar and A. Whitten. Www electronic commerce and java trojan horses. In *USENIX Workshop on Electronic Commerce*, WOEC'96, 1996.

[261] ValidSoft. ValidPOS, last access 2015. `http://www.validsoft.com/`.

[262] R. van Rijswijk-Deij and E. Poll. Using trusted execution environment in two-factor authentication: comparing approaches. In *Open Identity Summit*, OID'13, 2013.

[263] A. Vasudevan, E. Owusu, Z. Zhou, J. Newsome, and J. M. McCune. Trustworthy execution on mobile devices: What security properties can my mobile platform give me? In *International Conference on Trust and Trustworthy Computing*, TRUST'12, 2012.

[264] T. Vennon and D. Stroop. Threat analysis of the Android market. Technical report, GTC, June 2010. Smobile systems technical report, Available at `http://threatcenter.smobilesystems.com/wp-content/uploads/2010/06/Android-Market-Threat-Analysis-6-22-10-v1.pdf`.

[265] I. Vér and L. Beranek. *Noise and Vibration Control Engineering*. Wiley, 2005.

[266] VISA Inc. Verified by VISA, last access 2015. `http://www.visaeurope.com/en/cardholders/verified_by_visa.aspx`.

[267] VISA Inc. Visa Developer Program, last access 2015. `https://developer.visa.com/`.

[268] A. Wang. The shazam music recognition service. *Communication ACM*, 49(8), 2006.

[269] Z. Wang and R. B. Lee. Covert and side channels due to processor architecture. In *Annual Computer Security Applications Conference*, ACSAC'06, 2006.

[270] Web Bluetooth Community Group. Web Bluetooth, last access 2015. `https://webbluetoothcg.github.io/web-bluetooth/`.

[271] R.-P. Weinmann. Baseband attacks: Remote exploitation of memory corruptions in cellular protocol stacks. In *USENIX Workshop on Offensive Technologies*, WOOT'12, 2012.

[272] C. S. Weir, G. Douglas, M. Carruthers, and M. A. Jack. User perceptions of security, convenience and usability for ebanking authentication tokens. *Computers & Security*, 28(1-2), 2009.

[273] C. S. Weir, G. Douglas, T. Richardson, and M. A. Jack. Usable security: User preferences for authentication methods in ebanking and the effects of experience. *Interacting with Computers*, 22(3), 2010.

[274] Wireless Cables Inc. AIRCable, last access 2015. `https://www.aircable.net/extend.php`.

[275] M. Wu, R. C. Miller, and S. L. Garfinkel. Do security toolbars actually prevent phishing attacks? In *SIGCHI Conference on Human Factors in Computing Systems*, CHI'06, 2006.

[276] M. Wu, R. C. Miller, and G. Little. Web wallet: Preventing phishing attacks by revealing user intentions. In *Symposium on Usable Privacy and Security*, SOUPS'06, 2006.

[277] Z. Wu, Z. Xu, and H. Wang. Whispers in the Hyper-space: High-speed Covert Channel Attacks in the Cloud. In *USENIX Security Symposium*, USENIX'12, 2012.

[278] C. Xiao. Novel Malware XcodeGhost Modifies Xcode, Infects Apple iOS Apps and Hits App Store, last access 2015. http://researchcenter.paloaltonetworks.com/2015/09/novel-malware-xcodeghost-modifies-xcode-infects-apple-ios-apps-and-hits-app-store/.

[279] J. Xiao, Z. Xu, H. Huang, and H. Wang. Security Implications of Memory Deduplication in a Virtualized Environment. In *Dependable Systems and Networks*, DSN '13, 2013.

[280] W. Xu. Ah! universal android rooting is back. BlackHat USA, 2015. https://www.blackhat.com/docs/us-15/materials/us-15-Xu-Ah-Universal-Android-Rooting-Is-Back-wp.pdf.

[281] Z. Xu, K. Bai, and S. Zhu. Taplogger: Inferring user inputs on smartphone touchscreens using on-board motion sensors. In *ACM Conference on Security and Privacy in Wireless and Mobile Networks*, WISEC'12, 2012.

[282] Z. Xu and S. Zhu. Abusing notification services on smartphones for phishing and spamming. In *USENIX Workshop on Offensive Technologies*, WOOT'12, 201.

[283] Z. Ye, S. W. Smith, and D. Anthony. Trusted paths for browsers. *ACM Trans. Inf. Syst. Secur.*, 8(2), 2005.

[284] K.-P. Yee. User interaction design for secure systems. In *International Conference on Information and Communications Security*, ICICS '02, 2002.

[285] K.-P. Yee and K. Sitaker. Passpet: Convenient password management and phishing protection. In *Symposium on Usable Privacy and Security*, SOUPS'06, 2006.

[286] Yubico. Yubikey hardware, last access 2015. https://www.yubico.com/.

[287] Y. Zhang, J. I. Hong, and L. F. Cranor. Cantina: A content-based approach to detecting phishing web sites. In *International Conference on World Wide Web*, WWW'07, 2007.

[288] Y. Zhou and X. Jiang. Dissecting android malware: Characterization and evolution. In *IEEE Symposium on Security and Privacy*, SP'12, 2012.

[289] Y. Zhou and X. Jiang. Dissecting Android malware: Characterization and evolution. In *IEEE Symposium on Security and Privacy*, SP'12, 2012.

[290] Y. Zhou and X. Jiang. Android Malware Genome Project, last access 2015. `http://www.malgenomeproject.org`.

[291] Y. Zhou, Z. Wang, W. Zhou, and X. Jiang. Hey, you, get off of my market: Detecting malicious apps in official and alternative Android markets. In *Annual Network and Distributed System Security Symposium*, NDSS'12, 2012.

# Resume

CLAUDIO MARFORIO
Born in Novara, Italy on March 10, 1986
Citizen of Italy

## Education

| | |
|---|---|
| 2010 – 2015 | **Doctor of Sciences**<br>ETH Zurich |
| 2008 – 2010 | **Master of Science in Computer Science**<br>ETH Zurich |
| 2005 – 2008 | **Bachelor in Computer Science**<br>Università della Svizzera Italiana, Lugano (USI) |
| 2000 – 2005 | **Maturità Classica**<br>Liceo Classico A. Volta (Como, Italy) |

## Work Experience

| | |
|---|---|
| 2010 – 2015 | **Research Assistant and System Administrator**<br>Institute of Information Security, ETH Zurich |