Privacy-Enhancing Technologies for Mobile Applications and Services

Thèse N° 9226

Présentée le 15 février 2019

à la Faculté informatique et communications
Laboratoire pour les communications informatiques et leurs applications 1
Programme doctoral en informatique et communications

pour l'obtention du grade de Docteur ès Sciences

par

THI VAN ANH PHAM

Acceptée sur proposition du jury

Prof. C. González Troncoso, présidente du jury
Prof. J.-P. Hubaux, Prof. K. C. Huguenin, directeurs de thèse
Dr N. Taft, rapporteuse
Dr Y.-A. Pignolet, rapporteuse
Prof. S. Capkun, rapporteur

2019

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

He who has a why to live can bear almost any how.
Friedrich Nietzsche

To my parents

Abstract

Over a third of the world's population owns a smartphone. As generic computing devices that support a large and heterogeneous collection of mobile applications (apps), smartphones provide a plethora of functionalities and services to billions of users. But the use of these mobile apps and services often introduces privacy risks for users. First, app developers often fail to take into account the fact that as smartphones are generic computing devices, they do not provide adequate privacy and security guarantees for certain types of apps (e.g., those that collect and process sensitive data). As a result, new security and privacy threats are introduced by such apps. Second, due to the lack of privacy by design, apps often over-collect information about users, and can misuse the collected data. Our goal in this thesis is to identify privacy risks in mobile apps and services, and to design privacy-enhancing technologies to mitigate the identified threats. To broaden the impact of our research efforts, we focus on popular apps and services that are currently used by millions of users, and potentially by billions in the future. These include three use-cases: (mobile-health) mHealth apps, ride-hailing services, and activity-tracking services.

First, in the case of mHealth apps, we find that the Android operating system allows apps to easily fingerprint and identify other apps installed on the same phone. This causes privacy problems for mHealth apps, because the presence of an mHealth app can already reveal the medical conditions of its users. We investigate the apps' ability to fingerprint other apps and present HideMyApp, a practical system for hiding the presence of sensitive apps on Android. HideMyApp works on stock Android, and no firmware modification or root privilege is required.

Second, in ride-hailing services (e.g., Uber), to use the services, riders have to share their pick-up and drop-off locations with service providers. Consequently, the service providers can infer sensitive information about riders' activities (e.g., their visits to health clinics). Therefore, we propose ORide, a privacy-preserving ride-hailing service that efficiently supports the anonymous matching of riders and drivers. Also, ORide still supports functionalities that are often considered as important as privacy, including accountability, payment and reputation-rating operations.

Third, in activity-tracking services (e.g., RunKeeper), users report their location-based activities to service providers and obtain rewards based on their performance. However, from the location traces, the service providers can infer sensitive information about the users, e.g., their home/work addresses. Also, to obtain better rewards, the users might try to falsely report their activities to the service providers. Therefore, we propose SecureRun, a solution that enables service providers to compute accurate

activity summaries of the users, while guaranteeing the users' location privacy *vis-à-vis* the service providers.

In short, in this thesis, we identify privacy risks in popular mobile apps and services. We show that it is possible to provide them with added privacy, while preserving their rich functionality and usability. We are confident that the contributions presented in this thesis will inspire more future work on protecting users' privacy, not only in the context of mobile apps and services, but also in many other contexts brought about by new technological advances.

Keywords : privacy, ride-hailing, Android, fitness apps, location, app virtualization, data obfuscation, homomorphic encryption.

Résumé

Plus d'un tiers de la population mondiale possède un smartphone. En tant que plateforme informatique générique prenant en charge une vaste collection d'applications mobiles, les smartphones fournissent une multitude de fonctionnalités et de services à des milliards d'utilisateurs. Cependant, ces appareils présentent des risques pour la vie privée de leurs utilisateurs. Premièrement, les développeurs d'applications omettent souvent le fait que les smartphones sont des outils informatiques génériques et qu'ils ne fournissent pas nativement les garanties de confidentialité et sécurité suffisantes pour certains types d'applications (celles traitant des données sensibles, par exemple). En conséquence, de telles applications introduisent souvent de nouvelles menaces pour la sécurité et la vie privée. Deuxièmement, la sphère privée n'est pas toujours prise en compte dès la conception du service, poussant souvent leur fournisseur à collecter plus d'informations que nécessaire sur les utilisateurs et, potentiellement, à en abuser. Par conséquent, l'objectif de cette thèse est d'identifier les risques de confidentialité dans les applications et services mobiles et de concevoir des technologies atténuant lesdits risques. Pour maximiser l'impact de nos efforts de recherche, nous nous concentrons sur les applications et services populaires utilisés par des millions d'utilisateurs aujourd'hui, et peut-être par des milliards dans le futur. Celles-ci comprennent notamment trois catégories : les applications de santé mobile (mHealth), les services VTC (voiture de transport avec chauffeur) et les applications de suivi d'activités.

Dans le cas des applications mHealth, nous constatons que la plateforme Android permet aux applications de profiler et identifier d'autres applications installées sur le même téléphone. Cela entraîne des problèmes de confidentialité, car la simple présence d'une application mHealth peut déjà révéler les conditions médicales de ses utilisateurs. Nous quantifions la prévalence du problème et présentons HideMyApp, un système pratique qui fournit une défense robuste contre les attaques par profilage ciblant les applications sensibles sur Android.

Pour utiliser les services VTC, les usagers doivent partager leurs positions de départ et d'arrivée avec les prestataires de services. En conséquence, ces prestataires peuvent en déduire des informations sensibles sur les activités des usagers. Nous proposons ORide, un service VTC respectueux de la sphère privée et qui prend efficacement en charge la correspondance anonyme des usagers et des chauffeurs. ORide fourni ce service tout en garantissant les fonctionnalités requises des applications VTC, telles que les paiements, la possibilité d'évaluer les participants, ainsi qu'un mécanisme anti-fraude.

Les utilisateurs des applications de suivi des activités comme RunKeeper communiquent leurs données de localisation aux prestataires de services et obtiennent des

récompenses en fonction de leurs performances. Cependant, à partir de cet historique, les prestataires de services peuvent déduire des informations sensibles sur les utilisateurs, telles que leur adresse de domicile ou travail, par exemple. De plus, pour obtenir de meilleures récompenses, les utilisateurs peuvent parfois mentir sur leurs résumés d'activités. Nous proposons SecureRun, une solution qui garantit la confidentialité des données de localisations vis-à-vis des prestataires de services tout en permettant d'établir des statistiques précises des activités des utilisateurs comme, par exemple, la distance totale parcourue pendant une activité.

Cette thèse analyse donc en profondeur les risques de confidentialité pour les utilisateurs d'applications et de services mobiles populaires. Nous montrons qu'il est possible de doter ces derniers d'une plus grande confidentialité, tout en préservant leurs fonctionnalités et leur aisance d'utilisation. Nous sommes convaincus que les contributions présentées dans cette thèse inspireront des travaux futurs sur la protection de la sphère privée des utilisateurs d'applications ou services mobiles, mais aussi dans de nombreux autres domaines liés aux nouvelles technologies.

Mots-clés : protection de la sphère privée, service VTC, Android, applications de fitness, localisation, protection des données, encryption homomorphique.

Acknowledgements

First and foremost, I would like to express my sincere gratitude to my advisor, Prof. Jean-Pierre Hubaux, for his guidance over the five years of my PhD studies. His insightful advice greatly helped me to define and to shape my research. With his mentorship and encouragement, I improved not only my technical proficiency but also my communication, management, and interpersonal skills.

I am extremely grateful to my co-advisor, Prof. Kévin Huguenin, for his insight and useful critiques of my research. I would like to thank him especially for his immense support during the first two years of my PhD studies, when I did not know what research was, let alone how to do it. I am also extremely thankful to Dr. Italo Dacosta for being with me at every step of my research during the last four years, for the countless hours of discussions and brainstorming, and for our great joint-work that I very much enjoyed.

I would also like to thank my jury committee, Prof. Carmela Troncoso, Prof. Srdjan Čapkun, Dr. Yvonne-Anne Pignolet and Dr. Nina Taft, for finding the time in their busy schedules to serve on my committee.

I thank my collaborators for our fruitful research results: Juan Ramón Troncoso-Pastoriza, Virgil Gligor, Igor Bilogrevic, Florian Tramèr, Guillaume Endignoux, Bastien Jacot-Guillarmod, John Stephan, Taha Hajar and Eleonora Losiouk. I also thank my colleagues for their help and for the joys we shared throughout my time here: Zhicong Huang, Alexandra Olteanu, Jean Louis Raisaro, Ludovic Barman, David Froelicher, Christian Mouchet, Joao Sá Sousa, Mathias Humbert, Sébastien Henri and Apostolos Pyrgelis. I would also like to thank our lab secretaries, Patricia Hjelt and Angela Devenoge, for making all the administrative tasks so smooth, and Holly Cogliati-Bauereis for helping me proofread and improve my writing. Also, I extend my thanks to our sys-admins, Marc-André Luthi and Yves Lopes, for their help with the computing infrastructure.

Finally, I would like to express gratitude to my family for their continuous support, understanding, and patience. I would also like to thank Mai for being a great friend who is always with me in each and every low and high in my life. Last, but certainly not least, I would like to show my appreciation to Marek for being the best teammate I could ever wish for and for making this life even more enjoyable.

Contents

Chapter 1

Introduction

> *It will be a box less than an inch thick and smaller than a deck of cards. [...] The box will have a high-res color screen, a microphone, a plug for a headset or earphones, a camera lens, wireless connectivity, a cellphone and beeper functions, a television and radio receiver, a digital recorder, and it will have enough processing power and memory to function as a desktop system.*
>
> David Gerrold - Star Trek's author, in 1999,
> when he was asked to predict about the future of computing.

In 2007, the first smartphone was publicly introduced, and only a decade later, more than a third of the world's population owns a smartphone [1]. Due to their increasing processing capabilities, smartphones have become a generic computing platform that enables a myriad of mobile apps. A plethora of useful services has been delivered to users via these mobile apps.[1] The availability of app-development tools, app frameworks and third-party libraries has made it easy to develop mobile apps. Also, any app developer can publish an app through distribution platforms called app stores. Everyday, more than 7000 new apps are published on – the two largest app stores – the Google Play Store and Apple App Store [2], and users have, on average, at least 80 apps installed on their phones [3]. As illustrated by Apple's catchphrase, think of anything and "there's an app for it!". Indeed, there is an app for anything that a person could ever imagine, ranging from simple functionality (e.g., turning a smartphone into a flashlight) to sensitive and critical tasks (e.g., using a smartphone for continuous tracking of users' medical conditions).

In many aspects, these technological advances greatly facilitate our daily activities, empower individuals, and improve society. Obviously, the smartphone's ability to pinpoint our exact location and use it to deliver context-aware services (e.g., a meal, a taxi ride or a tip) is useful and convenient. Today, a user with diabetes can use her mobile-health app to wirelessly connect to an external glucose-monitoring device, for easily keeping track of her medical condition. With their capability to collect and rapidly spread information on a large scale, smartphones can facilitate political processes. A prominent

[1] Throughout this thesis, we use the term *app* to refer to a software program running on smartphones, and it can connect to remote servers to obtain services for its users.

example of this is the pro-democracy movement in many countries, such as Hong Kong and Egypt, where smartphones were used to coordinate protests, to document and share photos and videos about the state oppression with the world, in real time. In this sense, it looks like we are able to watch over Big Brother too; this is complementary to George Orwell's prophecy in his book 1984 where Big Brother would watch over us. This, however, is just one side to the story. The convenience and the utility of mobile apps and services come with privacy risks for users; as a consequence, they put users under new forms of surveillance, control, and manipulation.

First, app developers often fail to take into account the fact that as smartphones are generic computing devices, they do not provide adequate privacy and security guarantees for certain types of apps. As a result, new security and privacy threats are often introduced by such apps. For example, the Bluetooth authentication mechanism in Android, which occurs at the device level, makes the data transferred by one app over Bluetooth available to all apps on the same phone [4]. Or, as we show in Chapter 2, the Android operating system allows apps, even those with no permissions, to detect the presence of other installed apps on the phone. This could lead to devastating consequences for users of sensitive apps. For instance, in the case of protesters in Hong Kong using a mobile app (called FireChat) to coordinate protests, if authorities knew that a user had the app installed, they would know that she probably participated in the protests. The presence of an app could also reveal the sexual orientation of its users; in some countries, people are arrested if they are known to be homosexual [5]. In the context of mobile health (mHealth), the presence of an mHealth app might already reveal the medical conditions of its users. Furthermore, seemingly innocuous public resources of the phones, such as network statistics, can be used to infer sensitive information about users, such as their locations, identities and diseases [6]. These aforementioned threats exist, regardless of tremendous efforts that operating-system providers put on strengthening the smartphones' operating systems and their app frameworks. For example, Android, which was built based on the Linux kernel, has been updated with many critical security and privacy features, such as new sandboxing mechanisms, permission models and SELinux, to make it difficult for apps to access data from other apps. However, as novel use-cases develop, new threats and vulnerabilities will emerge. Therefore, it is important to continuously examine and understand the privacy protection provided by smartphones for different scenarios of apps, and accordingly develop defense mechanisms, when needed.

Second, due to the lack of privacy by design, apps and online service providers often put users' privacy at risk. They often encourage users to overshare the data collected by their smartphones. Each day, 500 million tweets are sent to Twitter, 95 million photos are uploaded to Instagram and 15 million Uber rides are taken [7, 8]. Moreover, apps might stealthily collect and misuse user's sensitive personal data *without* users' consent or knowledge. For example, in 2017, the popular weather app AccuWeather, with a million-user customer base, was found to collect and share precise geolocation data of its users, without their consent, with advertising firms [9]. Another example, Facebook, the world's largest online social network with more than 2 billion users, was found to collect users' contact information, SMS and call records for years [10]. This high volume of personal data can be used to infer detailed profiles of users, hence potentially lead to discrimination. For example, some online services were known to approve housing ads that excluded users based on their religion and gender [11]. The collected data can also be used by authorities or service providers for purposes that users were not informed. For

example, governments can use facial-recognition technologies to identify protesters. They can also track online activities of their citizens (e.g., comments on online social networks) to assign social-credit scores for them [12]. Another example, ride-hailing services with millions of users, such as Uber and Lyft, collected detailed location traces of riders and their identities, and this data was later used to infer riders' sensitive activities [13]. Furthermore, the lack of privacy by design not only affects personal privacy, but also national security. In January 2018, Strava, a popular activity-tracking service, released a data visualisation map that shows all the activities tracked by its users, and the map was detailed enough to give away the precise locations of classified military bases and CIA rendition sites [14]. These examples show that privacy problems caused by mobile apps and online services are real and they concern a significant fraction of users. Therefore, it is crucial to enable users to control and limit, in a more effective way, the information that apps collect. Apps and services should adhere to the data-minimization and privacy-by-design principles, i.e., they should collect and process only the personal data that they truly need to achieve their declared purposes. This is required by the EU general data protection regulation 2016/679 (GDPR) which started to take effect on 25 May 2018 [15]. At the same time, it is important to balance the trade-off between users' privacy and apps' usability, as privacy does not constitute a goal as such, but rather a necessity and a right.

Contributions

In this thesis, we identify some of the privacy risks that mobile apps and services might pose to users. We then design and thoroughly evaluate our protection mechanisms by implementing prototypes of our solutions and by measuring their performance with real-world data-sets and, when possible, user-studies. We focus on popular apps and services that are currently used by millions of users, and potentially by billions in the future. These include three use-cases: mHealth (mobile health) apps, ride-hailing services, and activity-tracking services.

We begin with the case of mHealth apps, where we design *a system for hiding the presence of such apps on Android*. It is known that millions of users rely on mobile health (mHealth) apps to manage their health and medical conditions. Although the popularity of such apps continues to grow, several privacy and security challenges can hinder their success. In particular, the simple fact that an mHealth app (e.g., diabetes app) is installed on a user's phone, can reveal sensitive information about the user's health (e.g., that he has diabetes). Given Android's open design, any app, even without permissions, can easily check for the presence of a specific app or to collect the entire list of installed apps on the phone; this is possible even if multiple user accounts or profiles are used. Many third parties are interested in such information; our survey of almost 3000 popular apps in the Google Play Store shows that more than half of these apps explicitly query for the list of installed apps. To date, no mechanism exists to effectively hide the use of sensitive apps. Therefore, we designed and implemented HideMyApp, an effective and practical solution for hiding the presence of mHealth and other sensitive apps from nosy apps on the same phone. HideMyApp relies on user-level virtualization techniques, thus avoiding changes to the operating system (OS) or to apps and still supporting their key functionalities. By using a diverse dataset of both free and paid mHealth apps, our experimental evaluation shows that HideMyApp supports main features in most apps

and introduces acceptable delays at runtime (several milliseconds); such findings were validated by our user study. In short, we show that the practice of collecting information about installed apps is widespread and that our solution, HideMyApp, provides a robust protection against such a threat.

We then look into the case of ride-hailing services and design *a privacy-preserving yet accountable solution* for them. Ride-hailing services such as Uber and Lyft have become increasingly popular, serving millions of users per day. However, they raise significant privacy concerns, because service providers are able to track the precise locations of all riders and drivers. We present the first analysis of the privacy threats in ride-hailing services. Our analysis exposes high-risk privacy threats that do not occur in conventional taxi services. Therefore, we propose ORide (Oblivious Ride), a privacy-preserving ride-hailing service based on somewhat-homomorphic encryption with optimizations such as ciphertext packing and transformed processing. With ORide, a service provider can match riders and drivers without learning their identities and their location information. ORide provides riders with fairly large anonymity sets (several thousands), even in sparsely populated areas. In addition, ORide supports key features of ride-hailing services such as easy payment, reputation scores, accountability, and retrieval of lost items. Using real data-sets that consist of millions of rides, we show that the computational and network overhead introduced by ORide is acceptable. For example, ORide adds only several milliseconds to ride-hailing operations, and the extra driving distance for a driver is less than 0.5 km in more than 75% of the cases evaluated. In short, we show that a ride-hailing service can offer strong privacy guarantees to both riders and drivers while maintaining its usability and functionality.

Finally, we look into the case of activity-tracking services and we design *a system for secure and private proofs of location-based activity summaries.* Activity-tracking services, where people record and upload information about their location-based activities (e.g., the routes of their activities), are increasingly popular. Such services enable users to share information and compete with their friends on activity-based social networks but also, in some cases, to obtain discounts on their health insurance premiums by proving they conduct regular fitness activities. However, they raise privacy and security issues: the service providers know the exact locations of their users, and the users can report fake location information to the service providers (e.g., to unduly brag about their performance). Therefore, we present SecureRun, a secure privacy-preserving system for generating and reporting location-based activity summaries (the total distance covered and the elevation gain). SecureRun is based on a combination of cryptographic techniques and geometric algorithms, and it relies on existing Wi-Fi access-point networks deployed in urban areas. We evaluate SecureRun by using real data-sets from the FON hotspot community networks and from the Garmin Connect activity-based social network, and we show that it can achieve tight (up to a median accuracy of more than 80%) verifiable lower-bounds of the distance covered and of the elevation gain, while protecting the location privacy of the users with respect to both the service providers and the access point network operator(s). The results of our online survey, targeted at RunKeeper users recruited through the Amazon Mechanical Turk platform, highlight the lack of awareness and significant concerns of the participants about the privacy and security issues of activity-tracking services. They also show a good level of satisfaction regarding SecureRun and its performance.

Thesis Outline

Following the three main contributions described above, the remainder of this thesis contains four chapters. First, we show in Chapter 2 that any app on Android can check for the presence of other apps installed on the same phone. This is problematic for sensitive apps such as mHealth or religion-related apps. Therefore, we propose a practical and effective solution for hiding the presence of sensitive apps. In Chapter 3, we design and evaluate a system that guarantees privacy for riders and drivers in ride-hailing services (e.g., Uber and Lyft), and still enables them to enjoy the convenience, usability and accountability as in current ride-hailing services. In Chapter 4, we propose a novel system for protecting location privacy of users in activity-tracking services. We present our concluding remarks in Chapter 5.

Publications

Chapter 2 is based on the results of [16]. Chapter 3 is a combination of [17] and [18]. Chapter 4 rests on the results of [19] and [20].

Chapter 2

HideMyApp: Hiding the Presence of Sensitive Apps on Android

Given their diverse utility and potential, smartphones are being adopted by many applications and services. But this is often done without taking into account that, as general computing devices, smartphones do not provide sufficient security and privacy protection for certain categories of apps. In this chapter, we show that the Android operating system permits apps, even those with zero permissions, to query for the presence of other apps installed on the same phone. This is problematic for sensitive apps, especially in the context of mobile health (mHealth), where the presence of an app can reveal the medical conditions of its users. To the best of our knowledge, we are the first to provide a usable and practical solution that effectively hides the use of sensitive apps with respect to a strong adversary (i.e., a nosy app that has access to both static and runtime metadata of the sensitive apps).

2.1 Introduction

Mobile health (mHealth), the use of technologies such as smartphones and wearable sensors for wellness and medical purposes, promises to improve the quality and reduce the costs of medical care and research. An increasing number of people relies on mHealth apps to manage their wellness and to prevent and manage diseases. For instance, more than a third of physicians in the US recommend mHealth apps to their patients [21], and there are around 325,000 mHealth apps available in major app stores [22].

Given the sensitivity of medical data, the threats of privacy leakage are one of the main hindrances to the success of mHealth technologies [23]. In this area, a serious and often overlooked threat is that an adversary can infer sensitive information simply from the presence of apps on a user's phone. Previous studies have shown that private information, such as age, gender, race, and religion, can be inferred from the list of installed apps [24, 25, 26, 27]. With the increasing popularity of mHealth apps, an adversary can now infer even more sensitive information. For example, learning that a user has apps to manage diabetes and depression can reveal the user's medical conditions; such information could be misused to profile, discriminate or blackmail the user. When

inquired about this threat, 87% of the participants in our user-study expressed concern about it (Section 2.10.5).

Due to Android's open design, a zero-permission app can easily infer the presence of specific apps, or even collect the full list of installed apps on the phone through standard APIs [6]. Our analysis shows that Android exposes a considerable amount of static and runtime metadata about installed apps (Section 2.4); this information can be misused by nosy apps to accurately fingerprint other apps installed on the phone. Such attacks are possible even when the sensitive apps and the nosy apps are installed on different user accounts or profiles on a single phone. In 2014, Twitter was criticized for collecting the list of installed apps in order to offer targeted ads [28].[1] But Twitter is not the only app interested in such information. Our static and dynamic analysis of 2917 popular apps in the US Google Play Store shows that approximately 57% of these apps include calls to API methods that explicitly collect the list of installed apps (Section 2.5). Our analysis, corroborating the findings of previous studies [30, 26], also shows that free apps are more likely to query for such information and that third-party libraries (libs) are the main requesters of the list of installed apps. As users can have on average 80 apps installed on their phones [3], most of them being free, there is a high chance of untrusted third-parties obtaining the list of installed apps.

Since 2015, Android started to classify as potentially harmful apps (PHA)[2] the apps that collect information about other installed apps without user consent [32]. To avoid this classification, however, developers simply need to provide a privacy policy that describes how the app collects, uses and shares sensitive user data [33]. We find it interesting that only 7.7% of the evaluated apps declared that they collect the list of installed apps in their privacy policies; several of these policies use similar generic statements (probably based on a common template), and some state that such a list is non-personal information (Section 2.5.5). Moreover, few users read and understand privacy policies [34], as our user-study also confirmed (Section 2.10.5). Android recently announced that their security services will display warnings on apps that collect without consent users' personal information, including the list of installed apps [35]. This is a welcomed step, but the effectiveness of security warnings is known to be limited [36, 37] and it is unclear how queries by third-party libs will be handled. It is also unclear if such mechanisms will be able to detect more subtle attacks, where a nosy app checks for the existence of a small set of sensitive apps by using more advanced fingerprinting techniques (Section 2.4). In addition, Android does not provide mechanisms to hide sensitive apps installed on their phones; a few third-party tools, designed for other purposes, can provide only partial protection to some users (Section 2.6). Therefore, it is crucial to have a solution that effectively hides the presence of sensitive apps from other apps on the same phone.

We propose HideMyApp (HMA), the first system that enables organizations and developers to distribute sensitive apps to their users while considerably reducing the risk of such apps being detected by nosy apps on the same phone. Apps protected by HMA expose significantly less identifying metadata, hence it is more difficult for nosy apps to detect their presence, even when the nosy apps have all Android permissions and debugging privilege (adb). With HMA, an organization such as a consortium of hospitals sets up an HMA app store where authorized developers collaborating with the hospitals

[1] Twitter recently announced that it excludes apps dealing with health, religion and sexual orientation [29].

[2] Now reclassified as Mobile Unwanted Software (MUwS) [31].

can publish their mHealth and other sensitive apps. Users employ a client app called HMA Manager to (un)install, use and update the apps selected from the HMA app store. HMA transparently works on stock Android devices, it does not require root access and preserves the app-isolation security model of the Android OS. HMA also preserves the key functionality of mHealth apps, such as connecting to external devices via Bluetooth, sending information to online services over the Internet, and storing information in a database. Moreover, HMA has only negligible effects on the usability of the apps.

With HMA, users launch a sensitive app inside the context of a container app, without requiring the sensitive app to be installed. A container app is a dynamically generated wrapper around the Android application package (APK) of the sensitive app, and it disguises the sensitive app at installation and runtime. To launch the APK from the container app, HMA relies on techniques described in existing works: the dynamic loading of compiled source code and app resources from the APKs and user-level app-virtualization techniques, e.g., [38, 39]. However, note that app-virtualization alone is not sufficient to provide robust protection against fingerprinting attacks, as many of the information leaks uncovered by our analysis are still possible when just app-virtualization is used. Therefore, our main contribution is the design and evaluation of mechanisms built on top of app-virtualization to reduce the information leaks that could be exploited to fingerprint sensitive apps. For instance, at runtime, the container app dynamically obfuscates identifiable information output by the sensitive app. Also, we provide developers with guidelines on how to reduce the identifiable metadata in the app. Moreover, we are the first to identify the security and functional limitations of using app-virtualization for the purpose of hiding apps.

Our thorough evaluation of HMA on a diverse data-set of both free and paid mHealth apps on the Google Play Store shows that HMA is practical, and that it introduces reasonable operational delays to the users. For example, in 90% of the cases, the delay introduced by HMA to the cold start of an mHealth app (i.e., the app is launched for the first time since the device booted, or since the system terminated it) by a non-optimized proof-of-concept implementation of HMA is less than one second. At runtime, the delay introduced is only several milliseconds and is unlikely to be noticed by users. Moreover, our user study, involving $N = 30$ participants, suggests that HMA is user-friendly and of interest to users.

Our main contributions in this chapter are as follows.

- *Systemized knowledge*: We are the first to investigate the different techniques that a nosy app can use to fingerprint another app in the same phone. Also, through our static and dynamic analysis of paid and free apps across all app categories in the Google Play Store, we gain understanding about the prevalence of the problem.

- *Design and implementation of a solution for hiding sensitive apps*: We present HMA, a practical system that provides robust defense against fingerprinting attacks that target sensitive apps on Android. HMA works on stock Android, and no firmware modification or root privilege is required.

- *Thorough evaluation of our solution*: The evaluation of HMA's prototype on apps from the Google Play Store suggests that our solution is practical. Moreover, the

results of our user study suggest that HMA is perceived as usable and of interest to users.

2.2 Related Work

Researchers have actively investigated security and privacy problems in the Android platform. Existing works show that third-party libraries often abuse their permissions to collect users' sensitive information [40, 41], and that apps have suspicious activities such as collecting call logs, phone numbers and browser bookmarks [30, 4, 26]. Zhou et al. [6] show that Android's open design has made publicly available a number of seemingly innocuous phone resources, including the list of installed apps; these resources could be used to infer sensitive information about their users, such as users' gender and religion [42, 24, 27, 26, 25]. Similarly, Chen et al. present a way to fingerprint Android apps based on their power consumption profiles. Moreover, a significant amount of work has been devoted to fingerprinting Android apps based on their (encrypted) network traffic patterns [43, 44, 45, 46]. Researchers have also shown that it is possible to perform re-identification attacks based on a small subset of installed apps [27, 47]. Demetriou et al. [26], in the same line as our work, performed a static analysis to quantify the prevalence of the collections of the list of installed apps and their meta-data by third-party libs. We go beyond their work, however, by systematically investigating all possible information leaks that nosy apps can exploit to fingerprint other apps, by performing a dynamic analysis and privacy-policy analysis and by designing a solution to address the problem.

Existing mechanisms for preventing nosy apps from learning about the presence of another app are not sufficient (Section 2.6). As we will show in Section 2.8, user-level virtualization techniques that enable an app (called *target app*) to be encapsulated within the context of another app (called *container app*) can be used as a building block for HMA. These techniques are used to sandbox untrusted target apps (e.g., [38, 39]) or to compartmentalize third-party libs from the host apps (e.g., [48]). As they were designed for a different problem, however, they do not directly help hide the presence of a sensitive target app on a phone: They either require the target apps to be first installed on the phone, thus exposing them to nosy apps through public APIs (Section 2.4), or they run multiple target apps inside the same container app, thus violating the Android's app-isolation security model. Furthermore, they do not provide any insight into the possible information leaks that a nosy app can exploit to fingerprint the target apps and how their app-virtualization techniques can be used for hiding the presence of apps.

2.3 Background

In this section, we provide background knowledge on Android apps and Android security models.

2.3.1 Android Security Model

Android requires each app to have a unique package name that is defined by its app developers and cannot be changed during its installation or execution. Upon installation, the Android OS automatically assigns a unique user ID (UID) to each app and creates

a private directory where only this UID has read and write permissions. Additionally, each app is executed in their dedicated processes. Thus, apps are isolated, or sandboxed, both at the process and file levels. Apps interact with the underlying system (i.e., the app framework and OS) via methods defined by the Java API framework and the Linux-layer interfaces. Depending on the API methods and commands, apps might need to be granted certain permissions [49] or privileges such as Android Debug Bridge (adb)[50]. Since Android 6.0, users could decide to grant or deny permissions to apps at runtime.

2.3.2 Android Apps and APK Files

Android apps are usually written in Java, then compiled into Dalvik bytecode (.dex format) to be executed on Android devices. Each app must contain a set of *mandatory* information, including a unique package name, an icon, a label, a folder containing the app's resources, and at least one of the following components: activity, service, broadcast receiver and content provider. An activity represents a single screen, and a service performs long-running operations in the background, i.e., without user interface. A broadcast receiver enables an app to subscribe and respond to specific system-wide events. A content provider manages the sharing of data between components in the same app or with other apps. Apps can also support *optional* features, including intent filters (an intent filter is an expression in an app's manifest file; it specifies the type of intents that the app would like to receive), permissions, shared libraries, customized app configurations (e.g., supported SDKs, themes and styles), custom permissions, and assets.

Apps are distributed in the form of APK files and must be digitally signed by their developers. An APK is a signed zip archive that contains the compiled code and resources of the app. Each APK also includes a manifest configuration file, called `AndroidManifest.xml`; this file contains a description of the app (e.g., its package name and its components).

2.4 Fingerprintability of Android Apps

In this section, we show that Android apps disclose a significant amount of information that can be used by nosy apps to perform fingerprinting attacks. This information includes *static information* (i.e., information available after apps are installed and that typically does not change during apps' lifetimes), and *runtime information* (i.e., information generated or updated by apps at runtime). To be exhaustive in our analysis, we focus on information leaks through both the Java API framework and the Linux-layer interface, with respect to the granted permissions and privileges of the nosy apps. The analysis was conducted on Android 8.0, and its findings are summarized in Table 2.1 and detailed below.

2.4.1 Information Leaks through the Java API Framework

Static Information. *Without permissions*[3], a nosy app can directly check if a specific package name exists on the phone. Note that a package name is mandatory and

[3]Note: Granting permissions to a zero-permission app does not enable it to collect additional static information about other apps.

	Java API Framework		Linux-Layer Interface	
	without Permissions	**with Permissions**	**Default App Privilege**	**Debugging Privilege (adb)**
Static Information	*Mandatory information:* + Package name + Components' names *Optional features:* + Intent filters + Requested permissions	(*) See note	+ Package names	+ Package names + APK path + APK file
Runtime Information	None	+ Files in external storage + Storage consumption + Running processes + Network traffic + Package names + Notifications	+ UI states† + Power consumption† + Memory footprints†	+ Files in external storage + Network consumption + Running processes + Screenshots + System log + System diagnostic outputs

Table 2.1: Identifying information about installed apps that a nosy app can learn, with respect to its permissions and privileges, through the Java API framework and the Linux-layer kernel. Analysis was conducted on Android 8.0, † means that the information could be learnt in older versions of Android, but later versions (e.g., Android 8.0) require the calling app to have adb privilege. **(*) Note:** Granting permissions to a zero-permission nosy app does not enable it to obtain more static information about apps.

unique for each app. This can be done by invoking two methods getInstalledApplications() and getInstalledPackages(). These methods return the entire list of installed package names. Moreover, apps can register broadcast receivers, such as `PACKAGE_INSTALLED` and `PACKAGE_ADDED`, to be notified when an app is installed. Apps can also use various methods of the `PackageManager` class, such as `getResourcesForApplication()`, as an oracle to check for the presence of an app. These methods take a package name as a parameter and return *null* if the app does not exist.

If Android restricts access to apps' package names (for example, by requiring permissions), a nosy app can still retrieve other static information for fingerprinting attacks: for example, apps' mandatory information (e.g., components' names, icons, labels, developers' signatures and signing certificates) and optional features (e.g., intent filters, requested permissions, shared libraries, custom permissions, assets and apps' configurations such as supported SDKs, themes and styles). This information can be obtained through a number of methods in the `PackageManager` class, such as `getPackageInfo()`.

Runtime Information. *Without permissions*, apps cannot obtain runtime information generated by other apps. However, *with permissions*, apps can obtain certain identifying information. For instance, the `PACKAGE_USAGE_STATS` permission permits an app to obtain the list of running processes (method `getRunningAppProcesses()`), and statistics about network and storage consumption of all installed apps, including their package names, during a time interval (method `queryUsageStats()`). Apps granted with the `READ_EXTERNAL_STORAGE` permission, a frequently requested permission, can inspect for unique folders and files in a phone's external storage (a.k.a. SD card). Also, apps with the `BIND_NOTIFICATION_LISTENER_SERVICE` permission can receive notifications sent to other apps. Moreover, apps with VPN capabilities (permission `BIND_VPN_SERVICE`) can intercept network traffic of other apps; existing work shows that network traffic, even encrypted, can be used to fingerprint apps with good accuracy [44, 51, 46].

2.4.2 Information Leaks through the Linux-Layer Interface

Static Information. With *default app privilege*, apps can retrieve the list of all package names on the phone. This can be done by obtaining the set of UIDs in the `/proc/uid_stat` folder and using the `getNameForUid()` API call to map a UID to a package name. With debugging privilege (*adb*), an app can retrieve the list of package names (command `pm list packages`) and learn the path to the APK file of a specific app (command `pm path [package name]`). Moreover, adb privilege enables a nosy app to retrieve the APK files of other apps (command `pull [APK path]`); the nosy app can then use a method, such as `getPackageArchiveInfo()`, to extract identifying information from the APK files.

Runtime Information. With *default app privilege*, an app can infer the UI states (e.g., knowing that another app is showing a login screen) [52], memory footprints (sequences of snapshots of the app's data resident size) [53] and power consumption [54] of other installed apps. Note that access to this information has been restricted in later versions of Android (e.g., Android 8.0 requires the apps to have adb privilege). Additionally, with *adb privilege*, apps can learn detailed information about runtime behaviors of other apps by inspecting the system logs and diagnostic outputs (commands `logcat` and `dumpsys`). Moreover, with adb privilege, apps can directly retrieve the list of running processes (command `ps`), take screenshots [55] or gain access to statistics about network usage of other apps (folder `/proc/uid_stat/[uid]`).

Our analysis shows that Android's open design makes it easy for a nosy app to fingerprint other apps on the same phone. The diverse source of identifying information leaks makes it difficult to add permissions or block all related methods, particularly because most of these methods have valid use cases and are widely used by apps; restricting or blocking them could break a significant number of apps and anger developers. We are the first to provide a solution for hiding the presence of sensitive apps with respect to a strong adversary that has access to all the aforementioned information (Section 2.7).

2.5 Apps Inquiring About Other Apps

We analyze popular apps from the Google Play Store to estimate how common it is for apps to inquire about other apps on the same phone. Our analysis focuses on apps that contain API calls to directly retrieve the list of installed apps (hereafter called LIA): getInstalledApplications() and getInstalledPackages() (hereafter abbreviated as getIA() and getIP(), respectively). We chose these methods because they are known for being used by developers to learn about the presence of other apps, whereas other methods presented in Section 2.4, such as `getPackageInfo()`, can be used in valid use cases or for app-fingerprinting attacks. Therefore, the results presented in this section can be interpreted as a *lower-bound* on the number of apps that collect identifying information about other installed apps.

2.5.1 Data Collection

We gathered the following datasets for our analysis:

APK Dataset. We collected APK files of popular free apps in the Google Play Store (US site). For each category of apps in the store (55 total), we gathered the 60 most

Analysis method	Call origin	getIA() (%)	getIP() (%)	getIA() or getIP() (%)
Static	Third-party libs + Apps	36.4	43.6	57.0
Static	Apps only	8.05	8.43	13.9
Dynamic	Third-party libs + Apps	6.54	15.0	19.2

Table 2.2: Proportion of free apps that invoke getIA() and getIP(), to collect LIAs with respect to different call origins.

popular apps. After eliminating duplicate entries (an app can belong to multiple categories), apps already installed on Android by default, and brand-specific apps, we were left with 2917 distinct apps.

Privacy-Policy Dataset. We collected privacy policies that corresponded to the apps in our dataset. Out of 2917 apps, we gathered 2499 privacy policies by following the links included in the apps' Play Store pages, i.e., 418 apps did not have privacy policies. Note that Google requires developers to post a privacy policy in both the designated field in the app's Play Store page and within the app itself, if their apps handle personal or sensitive user-data (e.g., the list of installed apps) [56].

2.5.2 Static Analysis

For our static analysis, by using the well-known tool Apktool [57], we decompiled the APKs in our dataset to obtain their smali code, a human-readable representation of the app's bytecode. We searched in the smali code for occurrences of two methods getIA() and getIP().[4] API calls can be located in three parts of the decompiled code: in the code of Android/Google libs and SDKs (e.g., Android AppCompat supporting library), in the code of third-party libs and SDKs (e.g., Flurry analytics, Facebook SDK), or in the code of the app itself. To differentiate among these three origins, we applied the following heuristics. First, methods found in paths containing the "com/google", "com/Android" or "Android/support" substrings, are considered part of Android/Google libs and SDKs. Second, methods found in paths containing the name of the app are considered part of the code of the app. We believe this is a reasonable heuristic, because to begin their package names, Android follows the Java package-name conventions with the reversed Internet domain of the companies, which generally has a length of two words. If the methods do not match the first two categories, then they are considered part of the code of a third-party library or SDK. Note that this approach, also used in previous work [26], cannot precisely classify obfuscated code or code in paths with no meaningful names (e.g., developers might not follow naming conventions). Such cases, which represent a small fraction (less than 5%), are classified as code from third-party libs or SDKs. Still, this approach provides us with a good estimation of the origin of most method calls.

Table 2.2 shows the proportions of apps that invoke two sensitive methods getIA() and getIP() with respect to different call origins. Of the 2917 apps evaluated, 1663 apps (57.0%) include at least one invocation of these two sensitive methods in the code from third-party libs and the apps. These results show a significant increase in comparison with the results presented in 2016 by Demetriou et al. [26]. These results also show that most sensitive requests come from third-party libs or SDKs; app developers might not be aware of this activity, as has been the case for other sensitive data such as location [58].

[4]Note that we also found many occurrences of other methods presented in Section 2.4, but we did not know the purposes of the calling apps.

Static analysis has two main limitations. First, it might be the case that methods appeared in the code are never executed by the app. Second, it is possible that the sensitive methods do not appear in the code included in the APK, rather in the code loaded dynamically by the app or the third-party libs at runtime. To address these issues, we also performed a dynamic analysis of the apps in our dataset, as we describe next.

2.5.3 Dynamic Analysis

For our dynamic analysis, by using XPrivacy [59] on a phone with Android 6.0, we intercepted the API calls made by apps. For the analysis to scale, for each app, we installed it and granted it all the permissions requested (declared in the app's Manifest file). Next, we launched all the runnable activities declared by the app for 10 minutes, trying to approximate the actions a user might trigger while using the app. Although this approach has limitations as it only has a short period of time per app and it cannot emulate all the activities a user could do, it is sufficient to estimate a *lower-bound* on the number of apps that query for LIAs at runtime.

Our results, presented in Table 2.2, show that 190 apps (6.54%) called getIA(), 436 apps (15.0%) called getIP(), and 19.2% of the apps in our dataset called at least one of these two sensitive methods. Because it is not possible to infer the origin of the request (Google/Android library, third-party library, or the app itself) from the data gathered, we performed some additional steps. For each app, we used the results of our static analysis (Section 2.5.2) and searched for occurrences of getIA() and getIP() in the code belonging to Google/Android libs. We found that most apps did not include calls to these sensitive methods in the code belonging to Google/Android libs: 181 out of 190 for getIA() and 412 out of 436 for getIP(). This means that, with good certainty, we can conclude that these sensitive requests came from third-party libs or from the code of the apps.

Interestingly, we found 49 apps that called at least one of the two methods in our dynamic analysis, but not in our static analysis. This could be because the decompiler tool produced incorrect smali code, or because these sensitive requests were dynamically loaded at runtime. Still, this represents a small percentage of the apps found through our analysis.

Our static and dynamic analysis shows that a significant number of popular free apps in the Google Play Store actively queries for LIAs: between 19.2% (dynamic analysis) and 57% (static analysis) of the apps in our dataset. In addition, our results show that these sensitive queries are mainly originated from third-party libs. Therefore, we can conclude that apps are interested in knowing about the presence of other installed apps, and that, if Android blocked the two methods getIA() and getIP(), nosy apps would likely make use of other methods presented in Section 2.4.

2.5.4 Analysis of Paid Apps

To estimate if there are differences between free and paid apps with respect to collecting LIAs, we performed a similar analysis (static and dynamic) with a set of 28 popular paid apps from different categories in the Google Play Store (see Table 2.3). We found that 17.8% of the paid apps included at least one call to getIA() or getIP() methods in their code (*upper-bound*) and that 7.4% of the paid apps called at least one of these two methods at runtime (*lower-bound*). Although the number of paid apps evaluated is much

1. org.xtramath.mathfacts	15. com.maxmpz.audioplayer.unlock
2. co.wordswag.wordswag	16. com.microphone.earspy.pro
3. com.vicman.photolabpro	17. com.tdr3.hs.Android
4. com.ultimateguitar.tabs	18. com.etermax.preguntados.pro
5. com.burleighlabs.babypics	19. com.real.bodywork.muscle.trigger.points
6. com.ellisapps.itrackbitesplus	20. com.devolver.spaceplan
7. com.intsig.lic.camscanner	21. org.twisevictory.apps
8. com.period.tracker.deluxe	22. com.samruston.weather
9. com.mojang.minecraftpe	23. au.com.shiftyjelly.pocketcasts
10. com.wolfram.Android.alpha	24. com.azumio.instantheartrate.full
11. slide.cameraZoom	25. udk.Android.reader
12. com.digipom.easyvoicerecorder.pro	26. radiotime.player
13. com.melodis.midomiMusicIdentifier	27. com.flyersoft.moonreaderp
14. com.laurencedawson.reddit_sync.pro	28. kr.aboy.tools

Table 2.3: List of paid apps evaluated in our study. Only a few paid apps (17.8% static analysis, 7.4% dynamic analysis) seems to request information about LIAs (Section 2.5).

smaller than of free apps, our results still indicate that paid apps are less likely to query for LIAs, probably due to the fact that they rely less on third-party libs, particularly ad libraries.

2.5.5 Analysis of Privacy Policies

Google's privacy-policy guidelines require apps that handle personal or sensitive user data to comprehensively disclose how the apps collect, use and share user-data. An example of a common violation, included in these guidelines, is "*An app that doesn't treat a user's inventory of installed apps as personal or sensitive user data*" [56]. In this section, we explain what developers understand about the guidelines and the declared purposes of the collection of LIAs by apps. For the sake of simplicity, in this section, we define a *nosy app* as an app that calls at least one sensitive method in the static (Section 2.5.2) *or* dynamic (Section 2.5.3) analysis.

As mentioned in Section 2.5.1, out of 2917 apps in our dataset, we found and collected 2499 privacy policies. From the 1674 nosy apps found in the static and dynamic analysis, 1524 apps have privacy policies. We semi-automated the policy analysis as follows. We built a set of keywords consisting of nouns and verbs that might be used to construct a sentence or paragraph to express the intention of collecting LIAs: retrieve, collect, fetch, acquire, gather, package, ID, installed, app, name, application, software, and list. For each privacy policy, we extracted the sentences that contain at least one of the keywords. From the extracted sentences, we manually searched for specific expressions, for example "installed app", "app ID" and "installed software". Thereafter, we read the matched sentences and the corresponding privacy policy. During the process, we discovered new patterns and we updated the expression list. We stopped when the results converged.

From the set of 2499 policies, we found 162 policies that explicitly mention the collection of LIAs. Among these, 129 belong to the set of 1674 nosy apps (7.7%). Interestingly, some apps have exactly the same privacy policies, even though they are from different companies (e.g., [60] and [61]). Also, 33 apps mentioned the collections of LIAs, but we did not find them in both static and dynamic analyses. For these apps, we performed a more thorough dynamic analysis: we used them as a normal user, while intercepting API calls. We did not capture, however, any calls to the two sensitive methods. This

might be because developers copy the privacy policies from other apps, or because the apps will make these calls in the future.

Among the generic declared purposes of the collections of LIAs by apps, for example for improving the service (e.g., [62]) or for troubleshooting the service in case the app crashes (e.g., [63]), some apps explicitly declare that they collect LIAs for targeted ads (e.g., [64, 65]), even more specifically targeted ads by third-party ad networks (e.g., [66]). Unexpectedly, we found that among the 162 privacy policies that mention the collections of LIAs, 76 of them categorize LIAs as *non-personal* information, whereas Google defines this as personal information. This shows a serious misunderstanding between developers and Google's guidelines.

2.6 Existing Protection Mechanisms

To the best of our knowledge, there are no robust mechanisms available to help users hide the existence of their sensitive apps from nosy apps. Below, we present some mechanisms that can offer partial protection.

2.6.1 Mechanisms by Google

Android does not provide users with a mechanism to hide the existence of apps from other apps. However, users can repurpose existing Android mechanisms for partially hiding apps.

Multiple Users. Android supports multiple users on a single phone by separating user accounts and app data [67]. Sensitive apps could then be installed and used in one or more secondary accounts. This approach will prevent nosy apps in the primary account from learning which apps are installed on secondary accounts, if nosy apps use the getIA() and getIP() methods. Unfortunately, multiple user accounts have a negative impact on apps' functionality and usability. For example, while the primary account is in the foreground, apps on secondary accounts cannot provide notifications to users or use Bluetooth services (important for mHealth apps). Furthermore, apps in different accounts cannot be used simultaneously without switching accounts, an operation that introduces a noticeable delay. More importantly, this is not an effective protection mechanism, because to bypass the isolation provided by multiple user accounts, a nosy app only needs to include additional parameters or uses different methods, as we detail below.

- On Android 7 or earlier, including an additional parameter flag (`MATCH_UNINSTALLED_PACKAGES`) in methods getIA() and getIP() will reveal the apps installed in secondary user accounts.

- On Android 9 (the latest version of Android), a nosy app can use API methods such as `getPackageUid()`, `getPackageGidS()` or `getApplicationEnabledSetting()` as oracles to check if an app is installed on the device, even if it is installed on a secondary account or on a work profile. The nosy app only needs to include the package name of the targeted sensitive app as a parameter to these methods. If the targeted app is installed on the device, these methods will return valid values, otherwise they return an error message or null.

- A nosy app can guess the UIDs of the apps installed on all the accounts and work profiles, by looking at the `/proc/uid` directory to learn the ranges of current UIDs in the system. It then guesses the UIDs of other apps and uses the `getNameForUid()` method to learn the package name. This method will return a package name given a UID as a input parameter; if the app does not exist, it returns null. As a result, it can be used as an oracle to retrieve the list of installed apps on the device. This was tested on Android 6, 8.1 and 9.

- A nosy app with adb privilege can easily verify if a sensitive app is running on the device, independently of the account or profile it was installed on, by using the shell command: *pidof <PackageName>*. This approach was tested on Android 9.

- A nosy app with adb privilege can obtain the list of installed apps, which includes apps on secondary accounts and work profiles, by using the shell command *dumpsys*. This approach was tested on Android 9.

Android for Work. Android supports an enterprise solution called Android for Work; this solution separates work apps from personal apps [68]. Our tests, using similar methods as with multiple users (presented above), also confirmed that, as with multiple users accounts, it is easy to identify which apps are in the work profile. In addition, Android for Work is only available to enterprise users.

Recently, Android introduced a new feature called *Instant Apps* [69]; this feature enables users to run apps instantly without installing them. Such an approach could be used to hide sensitive apps, however, it only supports a limited subset of permissions, and it does not support features that are crucial for mHealth apps, such as storing users' data or connecting to external Bluetooth-enabled devices [70]. Our solution (Section 2.7) uses this idea of running an app without installing it in order to hide its presence.

Google classifies the list of installed apps as *personal* information hence requires apps that collect this information to include in their privacy policies the purpose for collecting this data. Apps that do not follow this requirement are classified as Potentially Harmful Apps (PHAs) or Mobile Unwanted Softwares (MuWS) [32, 31]. Android security services, such as Google Play Protect [71] and SafetyNet [72], periodically scan users' phones and warn users if apps behave as PHAs or MuWS. Such mechanisms, however, do not seem to effectively protect against the unauthorized collection of the list of installed apps. Our analyses show that only 7.7% of the apps declare their collections of such information in their privacy policies, and some claim that a list of installed app is non-personal information (Sections 2.5). Moreover, these mechanisms might fail to detect more targeted attacks, for example, a nosy app might want to check if a small subset of sensitive apps exists on the phone.

2.6.2 Mechanisms by Third Parties

Samsung Knox [73] relies on secure hardware to offer isolation between personal and work-related apps, similar to Android for Work. Hence, it could be used to hide sensitive apps. Unfortunately, we were not able to evaluate the robustness of the protection offered by Knox with respect to hiding apps, because Samsung discontinued its support for work and personal spaces for private users; only enterprise users can use such a feature. Nevertheless, this solution is device-specific and only hides apps from other apps in a

different isolated environment, but not from apps in the same environment (apps in the same isolated environment can come from different, untrusted sources). That is, a solution that provides per-app isolation is preferable.

There are apps in the Google Play Store that help users to hide the icons of their sensitive apps from the Android app launcher (e.g., [74]). Even though they help hide the presence of the sensitive apps from other human users (e.g., nosy partners), these sensitive apps are still visible to other apps on the phone, through public APIs. Along the line of user-level virtualization techniques, on the Google Play Store, we found apps that use these techniques to enable users to run in parallel multiple instances of an app on their phones and to partially hide the app, (e.g., [75, 76, 77]). However, these solutions require the hidden app to be installed first on the phone before protecting it, thus triggering installation and uninstallation broadcast events that can be detected by a nosy app (see Section 2.4). In addition, these apps provide only a single isolated space, i.e., they do not protect apps from other apps in the same environment. Our preliminary evaluation of these apps also shows that their protection is limited, e.g., the names of the hidden apps can be found in the list of running processes.

2.7 Our System: HideMyApp

In this section, we present HideMyApp (HMA), a system for hiding the presence of sensitive apps.

2.7.1 System Model

The scenario envisioned for HMA is as follows. A hospital or a hospital consortium (hereafter called hospitals) sets up an app store, called HMA App Store, where app developers working for the hospitals publish their mHealth apps. Hospitals want their patients to use their mHealth apps without disclosing their use to other apps on the same phone.

To enable the users to manage the apps provided by the HMA App Store, the HMA App Store provides the users with a client app, called HMA Manager. This app can be distributed through any available app stores, such as the Google Play Store. To allow the HMA Manager app to install apps downloaded from the HMA App Store, similarly to other Google Play Store alternatives such as Amazon [78] and F-Droid [79], users need to enable the "allow apps from unknown sources" setting on their phones. Since Android 8.0, Google made this option more fine-grained by turning it into the "Install unknown apps" permission [80]. That is, users only need to grant this permission to the HMA Manager app to enable it to install apps from the HMA App Store.

2.7.2 Adversarial Model

We assume the Android OS on the users' phones to be trusted and secure, including the Linux kernel and the Java API framework. We assume there is a nosy app that wants to learn if a specific app is present on the phone, and the nosy app can have all permissions and it can manage to have adb privilege, e.g., by using methods presented by Lin et al. [55]. As a result, the nosy app can have access to all static and dynamic information related to installed apps on the phone, as presented in Section 2.4. We assume that the HMA App Store and the HMA Manager, provided by the hospitals, are trusted and

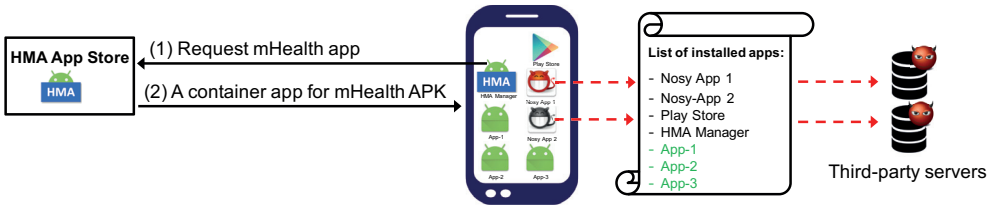


Figure 2.1: Overview of the HMA architecture. Nosy apps would only learn the generic names of the container apps.

secure, and that they follow the prescribed protocols of the system. We discuss mechanisms to relax the trust assumptions on the HMA App Store and HMA Manager app in Section 2.9.2.

2.7.3 Design Goals

The goal of HMA is to effectively hide the presence of sensitive apps, while preserving their usability and functionality.

- *(G1) Privacy protection.* It should be difficult for a nosy app to fingerprint and identify sensitive apps on the same phone.
- *(G2) No firmware modifications.* The solution should run on stock Android phones. That is, it should not require the phones to run customized versions of Android firmware, such as extensions to Android's middleware or the Linux kernel. This also means that the solution should not require the phones to be rooted.
- *(G3) Preserving the app-isolation security model of Android.* Each app should have its own private directory and run in its own dedicated process.
- *(G4) No app modifications.* The solution should not require access to the source-code of the sensitive apps, i.e., only their APK files are needed.
- *(G5) Usability.* The solution should preserve the key functionality of sensitive apps and their usability.

2.7.4 HMA Overview

From a high-level point of view, HMA achieves its aforementioned design goals by enabling its users to install a container app for each sensitive app (as illustrated in Fig. 2.1). Each container app has a generic package name and obfuscated app components. As a result, nosy apps cannot fingerprint a sensitive app by using the information about its container app. At runtime, the container app will launch the APK file of the sensitive app within its context by relying on user-level virtualization techniques. That is, the sensitive app is not registered in the OS.

To do so, HMA requires the hospitals to bootstrap the system by setting up the HMA App Store and distributing the HMA Manager app to users (Section 2.8.1). Through the HMA Manager app, users can (un)install, open, and update sensitive apps without being discovered by the OS and other apps. We detail these operations in Section 2.8.2.

2.8 HMA System Description

In this section, we detail the design of HMA, including its system-bootstrapping procedure and operations.

2.8.1 HMA System Bootstrapping

To bootstrap the system, the hospitals need to set up the HMA App Store and provide users with the HMA Manager app.

HMA Manager App

Recall, to hide their presence, sensitive apps are not registered in the OS; instead, their container apps are registered. Consequently, if users open their default Android app launcher, they will not see their sensitive apps; instead, they will see container apps with generic icons and random names. To solve this usability issue, at installation time (Section 2.8.2), the HMA Manager app keeps track of the one-to-one mappings between sensitive apps and their container apps. Using the mappings, the HMA Manager app can display the container apps to the users with the original icons and labels of their sensitive apps. To provide unlinkability between users and their sensitive apps with respect to the HMA App Store, the HMA Manager app *never* sends any identifying information of the users to the HMA App Store, and all the communications between the HMA App Store and the HMA Manager are anonymous. This is a reasonable assumption because the HMA Manager app can be open-sourced and audited by third parties. Also, in most cases, users do not have fixed public IP addresses; they access the Internet via a NAT gateway offered by their cellular provider. If needed, a VPN proxy or Tor could be used to hide network identifiers.

HMA App Store

The HMA App Store receives app-installation and app-update requests from HMA Manager apps and returns container apps to them. To reduce the delays introduced to the app-installation and app-update requests, the HMA App Store defines a set of P generic package names for container apps, e.g., app-1, app-2, ..., app-P. This set of generic names is shared by all sensitive apps, thus there is no one-to-one mapping between a sensitive app and a generic name or a subset of generic names.[5] As a result, for each sensitive app, the HMA App Store can generate beforehand P container apps corresponding to P predefined generic package names and store them in its database. Below, we explain the procedure followed by the HMA App Store to create a container app. Details about the app-installation and update requests from the HMA Manager apps are explained in Section 2.8.2.

HMA Container-App Generation. To generate a container app for a sensitive APK, the HMA App Store performs the following steps. Note that this operation cannot be performed by the HMA Manager app, because the Android OS does not provide tools for apps to decompile and compile other apps.

[5] P is defined based on the estimation about the number of sensitive apps that users of the HMA App Store can have, because Android does not permit duplicate package names for apps. Average users have around 80 apps installed on their phones [3], therefore P is *at most* 80.

- The HMA App Store creates an empty app with a generic app icon, a random package name and label, and it imports into the app the lib and the code for the user-level virtualization, i.e., to launch the APK from the container app. Note that the lib and the code are independent from the APK.
- The HMA App Store extracts the permissions declared by the sensitive app and declares them in the Manifest file of the container app.
- To enable the container app to launch the sensitive APK, app components (activities, services, broadcast receivers, and content providers) declared by the sensitive app need to be declared in the Manifest file of the container app. This information, however, can be retrieved by nosy apps to fingerprint sensitive apps (Section 2.4). To mitigate this problem, the container app declares activities and services of the sensitive app with random names. At runtime, the container app will map these random names to the real names. For broadcast receivers, the container app dynamically registers them at runtime. The case of content providers is discussed in Section 2.9.
- The HMA App Store compiles the container app to obtain its APK and signs it.

Note that for the sake of simplicity, here we present only a solution that protects mandatory features of Android apps. A resolute nosy app might try to fingerprint sensitive apps based on, for instance, the runtime information produced by their container apps. We discuss this in Section 2.9.

HMA User-Level Virtualization. To launch the APK of a sensitive app without installing it, its container app spawns a randomly named child-process in which the APK will run, i.e., the APK is executed under the same UID as its container app. Thereafter, the container app loads the APK dynamically at runtime, and it intercepts and proxies the interactions between the sensitive app and the underlying system (the OS and the app framework). To do so, we rely on the technique implemented by DroidPlugin [81], an open-source lib for app-virtualization.

2.8.2 HMA Operations

In this section, we detail the procedure followed by a user when she (un)installs, updates, or uses sensitive apps.

App Installation

An illustration of the app-installation procedure is shown in Figure 2.2. To install a sensitive app, the user opens her HMA Manager app to retrieve the set of apps provided by the HMA App Store. Once she selects a sensitive app, the HMA Manager app sends an installation request consisting of the name of the sensitive app and her desired generic package name for the container app to the HMA App Store. The HMA App Store finds in its database or creates a container app for the requested sensitive app (see Section 2.8.1), and it sends the container app, together with the original label and icon of the sensitive app, to the HMA Manager. The HMA Manager prompts the user for her confirmation about the installation. Once the user accepts, the installation occurs as in standard app installation on Android. In addition, the HMA Manager saves a record of the package

name of the container app and the package name, the original icon and label of the sensitive app in a database in its private directory.

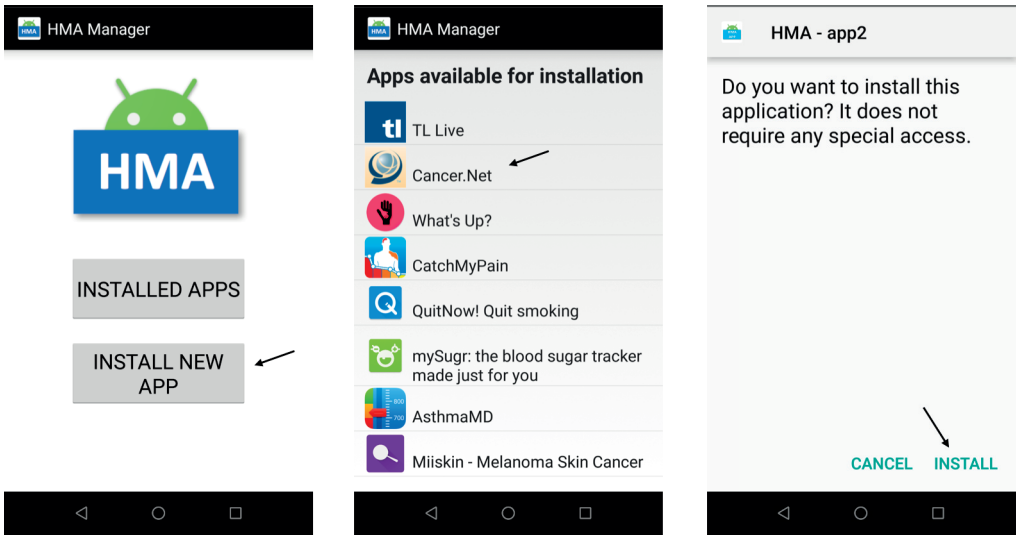


Figure 2.2: Steps to install an example app (called Cancer.Net) by using the HMA Manager app. The screenshots were taken from our proof-of-concept implementation (Section 2.10).

App Launch

To launch a sensitive app, the user opens her HMA Manager app to be shown with the set of container apps installed on her phone. Using the information stored in its database about the mappings between container apps and sensitive apps (Section 2.8.2), the HMA Manager displays to the user the container apps with the original labels and icons of the corresponding sensitive apps. Therefore, the user can easily identify and select her sensitive apps.

The *first time* a container app runs, it needs to obtain the sensitive APK from the HMA App Store, and then stores the APK in its private directory. This incurs some delays to the first launch of the sensitive app. However, it is needed to prevent the sensitive app from being fingerprinted: if the sensitive APK was included in the resources or assets folders of its container app so that the container app could copy and store the APK in its private directory at installation time, a nosy app would be able to obtain the sensitive APK. Recall, any app can obtain the resources and assets of other apps (Section 2.4). Also, Android does not permit apps to automatically start their background services upon their installation.

At runtime, the container app dynamically loads the sensitive APK. Thereafter, it intercepts and proxies API calls and system calls between the sensitive app and the underlying system, as described in Section 2.8.1. If the version of the Android OS is at least 6.0, permissions requested by the sensitive app will be prompted by its container app at runtime. As a result, they will be shown with the generic package name of the container app. This, however, does not affect the usability and comprehensibility of the permission requests, as shown by our user study (Section 2.10.5).

App Update

When a sensitive app on the HMA App Store has an update, for each predefined generic container-app package name, the HMA App Store generates a corresponding container app for the updated sensitive app. This step is needed, because the configuration file of the container app needs to be updated with respect to the updates introduced by the sensitive app. The HMA App Store then sends a push notification to all HMA Manager clients to notify them about the update. If a user has the sensitive app on her phone, her HMA Manager sends the package name of its existing container app to the HMA App Store. In return, it receives the corresponding updated container app from the HMA App Store. It then prompts the user for her confirmation about the installation. Once the user accepts, the updated container app is installed on the phone, similarly to the standard app-update procedure on Android (i.e., the updated app reuses its existing directory and UID).

App Uninstallation

To uninstall a sensitive app, the user opens her HMA Manager app to be shown with the set of container apps installed on her phone. Once she selects the app, the HMA Manager prompts her to confirm the uninstallation. Thereafter, the uninstallation occurs similarly to the standard app-uninstallation procedure on Android.

2.9 Privacy and Security Analysis

Here, we present an analysis of HMA to show that it effectively achieves its privacy and security goals (Section 2.7.3) with respect to the source of possible information leaks described in Section 2.4.1.

2.9.1 Privacy

Mandatory Static Information. Obviously, HMA effectively protects the *mandatory* information of sensitive apps by obfuscating the metadata associated with the sensitive app. First, a nosy app cannot obtain the package name of a sensitive app, because the sensitive app is never registered on the system; instead, its container app with a generic package name is installed. With adb privilege, the nosy app can obtain the path to the APK of the container app, but this path does not contain any information about the sensitive app. Moreover, the nosy app can retrieve the APK file of the container app, but it cannot retrieve the APK file of the sensitive app. This is because the sensitive APK is not included in the container-app's APK; instead, the sensitive APK is stored inside the private directory of its container app. Second, the resources, shared libraries, developers' signatures and developers' signing certificates of the sensitive app cannot be learnt by the nosy app, because they are not declared or included in the container-app's APK file, i.e., they are dynamically loaded from the sensitive app's APKs at runtime. Third, the nosy app cannot learn the components' names of the sensitive app, because these names are randomized. To prevent a nosy app from being able to fingerprint sensitive apps by using the number of app components declared in their container apps, during the generation of container apps, the HMA App Store adds dummy random components such that all container apps generated by the HMA App Store declare the same number

of random components. Moreover, the HMA App Store cannot link an app-installation with an app-update event of the same user, because all users share the same set of generic package names for container apps defined by the HMA App Store. This reduces the linkability between a user and a sensitive app with respect to the HMA App Store.

Optional Static Information. A nosy app might try to fingerprint sensitive apps by using the set of permissions declared by their container apps. This problem can be mitigated if all container apps declare a union of permissions requested by sensitive apps in the HMA App Store. Note that if the device's OS is at least 6.0, container apps only request, at runtime, for permissions needed by their sensitive apps, and users can grant or decline the requests. This makes it difficult for nosy apps to fingerprint sensitive apps by using the set of permissions that their container apps have been granted. Furthermore, to hide their presence, the sensitive apps should not support content providers and implicit intents, because these features expose the sensitive apps to other apps, hence enabling fingerprinting attacks (Section 2.11.1). Moreover, to prevent a nosy app from fingerprinting sensitive apps based on their customized configurations, such as themes and screen settings, the HMA App Store can define a guideline for app developers to follow such that all apps have the same configurations. This will affect the look and feel of the sensitive apps, but it is a trade-off between usability and privacy. Note that the same approach has been used in other deployed systems, such as in the Tor browsers where all the versions have the same default window size and user-agent strings [82]. To facilitate guideline compliance, the HMA App Store can also provide developers with IDE plugins to help them write guideline-compliant code; such an approach has been proposed in existing work (e.g., [83] and [84]).

Runtime Information. A nosy app cannot fingerprint a sensitive app by using the list of running processes it can retrieve, because the sensitive app runs inside the child process of its container app with a random name. To prevent nosy apps from fingerprinting sensitive apps by using statistics about resources consumed by their container apps (including network and storage consumption, power consumption, and memory footprints, system logs and diagnostic outputs), the container apps can randomly generate dummy data to obfuscate the usage statistics of sensitive apps. Also, app developers should not write identifying information about their sensitive apps to the log. The container apps can also obfuscate the UI states of the sensitive apps by overlaying transparent frames on the real screens of the sensitive apps. Regarding files in the phone's external storage, container apps can intercept and translate calls from sensitive apps when they create or access files in the external storage. However, note that apps are not recommended to store data in external storage, especially mHealth apps. In addition, to prevent the nosy app from receiving notifications sent to other apps, the users should not grant the `BIND_NOTIFICATION_LISTENER_SERVICE` to apps that they do not trust. Apps with adb privilege can take screenshots of sensitive apps and infer their names from the screenshots. This, however, requires the nosy app to do extra and error-prone operations (e.g., image processing) to identify sensitive apps.

2.9.2 Security

As explained in Section 2.8.1, the user-level virtualization technique used by HMA to launch an APK in HMA does not require users to modify the OS of the phone. The Android's app-sandboxing security model is also preserved, because each APK runs inside

the context of its container app. Thus, it is executed in a process under the same UID as its container app, and it uses the private data directory of its container app. Similarly to other third-party stores (e.g., Amazon or F-Droid), HMA requires users to enable the "allow apps from unknown sources" setting on their phones. However, apps installed from these sources are still scanned and checked by Android security services for malware [71]. Also, recently, this setting was converted to a per-app permission [80]. As a result, granting the HMA Manager app the permission to install apps from unknown sources will not give other apps on the phone the same permission.

With HMA, developers still control and sign their apps, and the HMA App Store validates these signatures before publishing the apps on the app store. The container apps are signed by the HMA App Store, vouching for the developers' signatures. Furthermore, HMA container apps only ask users for permissions requested by sensitive apps. To relax the trust assumptions on the HMA App Store and HMA Manager, the HMA App Store can provide an API so that anyone can implement her own HMA Manager app, or the HMA Manager app can be open-source, i.e., anyone can audit the app and check if it follows the protocols as prescribed. Therefore, assuming that the metadata of the network and the lower communication layers cannot be used to identify users, for instance, by using a proxy or Tor, the HMA App Store cannot link a set of sensitive apps to a specific user.

2.10 Evaluation

To evaluate HMA, we used a real dataset of free and paid mHealth apps on the Google Play Store. We looked into three evaluation criteria: (1) overhead experienced by mHealth apps, (2) HMA runtime robustness and its compatibility with mHealth apps, and (3) HMA usability.

2.10.1 Dataset

In February 2018, we selected 50 apps from the medical category on the Google Play Store, of which 42 apps are free and 8 apps are not. To have a diverse dataset, we selected apps based on their popularity (more than 1000 downloads), their medical specialization, and their supported functionality. From the 50 apps, we filtered out apps that make calls to APIs that we did not support in our prototype implementations, including Google Mobile Services (GMS), Google Cloud Messaging (GCM) and Google Play Services APIs. Note that these services could be supported, similarly to other services, the only challenge is engineering efforts. We also filtered out apps that use Facebook SDKs, because such SDKs often use custom layouts that have not been yet supported by the user-level virtualization library that HMA uses. Exploring the interaction mechanisms between custom layouts and custom notifications with the Android app framework is an avenue for future work.

After filtering, we narrowed down the dataset to 30 apps (24 free apps and 6 paid apps, see Table 2.4). Our dataset is somewhat diverse with 15 medical conditions. Also, apps in our dataset support features that are crucial for mHealth apps, such as a Bluetooth connection with external medical devices (e.g., `Beurer HealthManager` app [85]) and an Internet connection (e.g., `Cancer.Net` app [86]).

Index	App Name	Package Name	# Downloads
1	My Ovulation Calculator	com.ecare.ovulationcalculator	1M - 5M
2	Blood Pressure Log - MyDiary	com.zlamanit.blood.pressure	500K - 1M
3	DreamMapper	com.philips.sleepmapper.root	100K - 500K
4	Breathing Zone	com.breathing.zone	5K - 10K
5	Alzheimer's Speed of Processing Game	com.TinyHappySteps.Bird	1K - 5K
6	Cancer.Net Mobile	com.fueled.cancernet	10K - 50K
7	AIDS Info Drug Database	com.aidsinfo.aidsinfoapp	5K - 10K
8	My Pain Diary	com.damonlynn.mypaindiary	5K - 10K
9	iBP Blood Pressure	com.leadingedgeapps.ibp	10K - 50K
10	Squeezy: NHS Pelvic Floor App	com.propagator.squeezy	10K - 50K
11	OneTouch Reveal	com.lifescan.reveal	500K - 1M
12	ADHD Adults	com.labshealth.tdahadults	10K - 50K
13	Baritastic - Bariatric Tracker	com.baritastic.view	100K - 500K
14	Mole Mapper	edu.ohsu.molemapper	1K - 5K
15	Respiroguide Pro	com.tremend.respiroguide	10K - 50K
16	FearTools - Anxiety Aid	com.feartools.feartools	10K - 50K
17	Back Pain Relieving Exercises	com.backpainrelieving.backpain	10K - 50K
18	OnTrack Diabetes	com.gexperts.ontrack	500K - 1M
19	Asthmatic	be.sarahvn.asthmatic	50 - 100
20	MoodTools - Depression Aid	com.moodtools.moodtools	100K - 500K
21	Pain Diary & Forum CatchMyPain	com.sanovation.catchmypain.phone	50K - 100K
22	Beurer HealthManager	com.beurer.connect.healthmanager	100K - 500K
23	Constant Therapy	com.constanttherapy.Android.main	10K - 50K
24	Self-help Anxiety Management	com.uwe.myoxygen	100K - 500K
25	Pregnancy Calendar and Tracker	ru.mobiledimension.kbr	1M - 5M
26	BELONG Beating Cancer Together	com.belongtail.belong	10K - 50K
27	AsthmaMD	com.mobilebreeze.AsthmaMD	10K - 50K
28	Propeller	com.asthmapolis.mobile	5K - 10K
29	mySugr: the blood sugar tracker made just for you	com.mysugr.Android.companion	500K - 1M
30	QuitNow! - Quit smoking	com.EAGINsoftware.dejaloYa	1M - 5M

Table 2.4: mHealth apps tested with HMA.

2.10.2 Implementation Details

Our prototype implemented the main components of HMA, including the HMA App Store and the HMA Manager app. To measure the operational delay introduced by HMA, we implemented a proof-of-concept HMA App Store on a computer (Intel Core i7, 3GHz, 16 GB RAM) with MacOS Sierra. Our HMA App Store dynamically generated container apps from APKs, and it relied on an open-source library called DroidPlugin [81] for the user-level virtualization. Our prototype container apps dynamically loaded the classes and resources of the apps from the mHealth APKs, and they supported the interception and proxy of API calls that are commonly used and crucial to mHealth apps, such as APIs related to Bluetooth connections and SQLite databases.

2.10.3 Performance Overhead

In this section, we present the delays introduced by HMA to sensitive apps during app-installation and app-launch operations. We omit the app-update operation, because app-update and app-installation operations are similar. For the evaluation of delays added by the user-level virtualization to commonly used API methods and system calls at runtime of sensitive apps, we refer the readers to existing work such as Boxify [39]; they show that such overhead is negligible, e.g., opening a camera introduces an overhead of 1.24 ms.

Results presented in this section were measured on a Google Nexus 5X phone running Android 7.0, one of the most recent Android versions. In our experiments, the HMA App Store connected to the phone through a micro-USB cable, thus network delays were not considered. However, note that, compared to the standard use of apps, HMA incurs negligible network-delay overheads, because the only bandwidth overhead intro-

duced by HMA is the container-app payload whose size is of several hundreds of kilobytes only.

App Installation

When a user wants to install an mHealth app, the HMA App Store first needs to create a container app for it. Based on our experiments, assuming the HMA App Store decompiles the mHealth APKs beforehand, for 90% of the cases, generating a container app takes, on average, 5 s. Note that a large part of the delay comes from the compilation of the container app, and the measurement was performed on a laptop computer (MacOS Sierra, 3 GHz Intel Core i7). Also note that the HMA App Store can always prepare in advance container apps for each mHealth app, as presented in Section 2.8.1. The size of the container apps is only several hundreds of kilobytes, which would take less than a second for the HMA Manager app to download (assuming a 3G or 4G Internet connection). As a result, the total delay incurred by HMA will take less than 5 s in the *worst-case* scenario, and less than a second if container apps are generated beforehand, which is acceptable.

App Launch

On Android, apps can be launched from two different states: *cold starts* where apps are launched for the first time since the phone was booted or since the system killed the apps, and *warm starts* where the apps' activities might still reside in memory, and the system only needs to bring them to the foreground, hence faster than cold starts.

Experiment Set-Up. To measure cold-start delays (i.e., the time elapsed since a user clicks on the app's icon until the first screen of the app is loaded), we rely on the Android's official launch-performance profiling method [87]. For each app, we installed its container app, copied its APK file to the container-app's private directory, and launched the container app through `adb`. Thereafter, we extracted the time information from the `Displayed` entry of the `logcat` output. To simulate a first launch, before we launched an app, we used the command `adb shell pm clear [package-name]` to delete all data associated with the app [50] (i.e., to bring the app back to its initial state). To simulate a cold start, before we launched an app, we used the command `adb shell am force-stop [package-name]` to kill all the foreground activities and background processes of the app. For each app, we collected 50 measurements per launch setting. For baseline measurements, we measured the delays experienced by mHealth apps when they were executed without HMA, i.e., we installed the apps on the phone and followed the same procedure.

To measure warm-start delays, due to the lack of Android supports for profiling warm starts, we have to instrument the source-code of the sensitive apps to log the time that the app enters different stages in its lifecycle. Because apps in our dataset are closed-source, we used an open-source app [88]. To simulate a warm start, we used the command `input keyevent 187` to bring the app to the background, and then we used the `monkey` command to bring the app back to the foreground. By subtracting the time when the `onResume()` method is successfully executed with the time before the monkey command is sent, we know the warm-start delay experienced by the app. We measured the warm-start delays experienced by the app in both settings (with and without HMA), 50 measurements per setting.

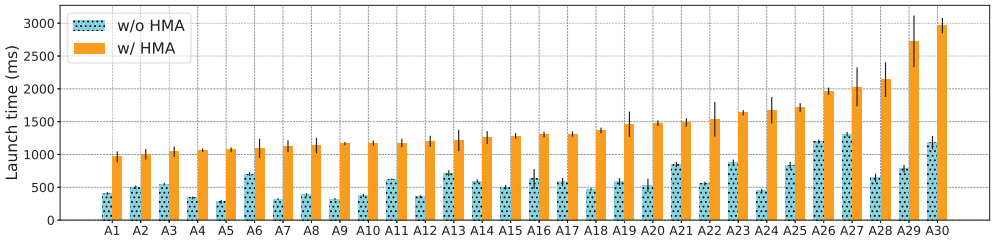


Figure 2.3: Cold-start delays experienced by mHealth apps when they are executed with and without HMA. Note that our HMA implementation is a proof-of-concept, hence not-optimized. The heights of the bars represent mean values and the error bars represent the standard deviation. For each setting, we collected 50 measurements per app. The full names of the apps can be found in Table 2.4.

Results. It is intuitive that, in HMA, the first launch of an mHealth app will experience longer delays than the subsequent cold starts, because the container app has to download the APK from the HMA App Store and stores the APK in its private directory. It also needs to process the APK and cache the information needed for the user-level virtualization. Our experiments show that the median of the first launch delays is 6.5 ± 0.16 s (as compared to 0.74 ± 0.07 s if the mHealth apps were launched without HMA). However, this delay occurs only once, therefore it is negligible with respect to the lifetime of the app on the phone.

Fig. 2.3 shows the bar plot of subsequent cold-start delays experienced by mHealth apps when they are executed with and without HMA; the heights of the bars represent the mean values and the error bars represent one standard deviation. It can be seen that the average delays are at most 3.0 ± 0.5 s and 1.3 ± 0.05 s if the apps are executed with HMA and without HMA respectively. For 90% of the cases, the average delay with HMA is less than 2.0 ± 0.3 s. Note that our prototype implementation is a proof-of-concept hence not optimized. However, the observed delays are still under the delay limit of 5 s suggested by Android [87]. Also, in our user study, 97% of participants agreed that a launch delay of around 5 s is acceptable (see Section 2.10.5).

Regarding the warm-start delays, we found that the average warm-start delays experienced by our tested app when it was launched with and without HMA were both approximately 0.55 s. This is intuitive, because the app's processes were still running and the activities still resided in the phone's memory. In case the garbage collector evicts the activities from the phone's memory, warm-start delays can be longer, due to the overheads of activity initializations. Unfortunately, we cannot simulate this case, because Android does not provide methods to control the garbage collector. However, in that case, the delay will still be less than cold-start delays (i.e., at most 3 s, see above).

2.10.4 HMA Robustness and Compatibility

In this section, we present the evaluation of HMA in terms of its robustness and its compatibility with Android versions.

Runtime Robustness. Following the approach used in previous work, (e.g., [40] and [89]), we manually tested each app in our dataset with HMA. For each mHealth app, we extracted its APK and used the HMA App Store to create a container app and installed the container app on the phone. Thereafter, we used the HMA Manager to

launch the app. We manually used most of the functionality of the mHealth app, and checked if it crashed during its execution. We found that all of the apps in our dataset work normally, except one app that threw an error when making an SQLite connection. However, we ran an example app (from [90]) that uses the official Android APIs for database (i.e., `Android.database.sqlite`) to insert and retrieve records from an SQLite database, and the app ran successfully. We suspected that the app specified the full path to the directory of the database (i.e., `/data/data/package-name/db-name`) hence failed the call, because the directory does not exist. This problem could be solved if the developers specified the relative path to the database (i.e., `./db-name`) instead of its full path.

Compatibility. We ran HMA on a series of smartphones with Android OS from version 5.0 to 8.0 and found that HMA can be successfully deployed on mainstream commercial Android devices. However, there are two apps (`Mole Mapper` and `Alzheimer's Speed of Processing Game`) that initially failed to run on our Nexus 5X (Android 7.1.1) due to the incompatibility between 32-bit and 64-bit systems. We fixed the problem by enabling the option `--abi armeabi-v7a` when installing them. From the list of 20 apps that we filtered out, we found that 3 apps (`Hearing Aid`, `What's Up` and `Cardiac diagnosis`) successfully ran on Android 5.0 and 6.0, but they failed to run in later versions of Android. We investigated the log of the three apps and found that API methods related to GMS services that we have not supported were called in the later versions of Android. This problem, however, could be solved if these services are hooked, as we discussed in Section 2.10.2.

2.10.5 HMA Usability and Desirability

To evaluate the usability of HMA and the users' interest for it (i.e., desirability), in February 2018, we conducted a user study at the laboratory for behavioral experiments at University of Lausanne (UNIL).[6] Our study involved 30 student subjects (19 males, 11 females, age: 22 $\pm$ 4.5 years old) from 18 different majors. The participants were experienced Android users: 87% of the participants have been using an Android phone for at least one year. Also, they were relatively concerned about their privacy; using the standard metric for measuring users' privacy perception (IUIPC) [91], we found that, on a scale from 1 to 5 for privacy postures, 97% of participants graded at least 3.0 and an average of 4.1.

We began the study with an entry survey about demographic information, privacy postures, and users' awareness and concerns about the problem of LIA collections. Then, we provided each participant with a fresh phone and asked them to install and use two apps: a popular app about public-transportation timetables in Lausanne called `tl-live` and an mHealth app called `Cancer.Net`. In order to precisely measure the users' perception of the delay introduced by HMA, the participants were asked to use the two aforementioned apps with and without HMA. The participants were provided with detailed instructions in English, including screenshots, on how to install and run the apps with and without HMA. 67% of the participants had used the `tl-live` app before, whereas only 7% of them had used the `Cancer.Net` app or an app of the same category. We finished the user study with an exit survey containing questions related to the usability of the solution and the users' levels of interest in HMA. The user-study

[6] Our user study was approved by our institutional ethical committee.

session took approximately 45 minutes and we paid each participant 25 CHF. The full transcript of survey questions and the user-study instructions can be found at [92]

Our study shows that the participants are concerned about the privacy of health-related data: 90% of the participants would be at least concerned if their health-related information were to be collected by mobile apps installed on their phones and shared with third parties, and 87% of participants would be at least concerned if third-parties learned that they had used health-related apps. Indeed, our study confirms the findings from previous works (e.g., [34]) that the majority of people never read privacy policies. As a consequence, the current solution of using privacy policies by Google for LIA collections is not satisfactory. These findings show the need for a solution to hide the presence of a sensitive mobile app.

Regarding the usability of HMA, only 30% of the participants noticed a difference when the two apps ran with and without HMA. Note that the delays that users experienced in the user study were the first-launch delays, which are 4.2 $\pm$0.06 s and 5.1 $\pm$ 0.07 s for the `tl-live` app and the `Cancer.Net` app, respectively. From the open-ended question in our exit survey, we found that the observed differences are mainly about the launching delay of the apps and the change in the app names in permission prompts. From the close-ended questions, which were coded with a five-point Likert scales, we made the following observations. Almost all participants agreed that these changes and delays are acceptable (97% and 93% of the participants, respectively). 93% of the participants also agreed that the use of HMA Manager to install and launch apps is at least somewhat acceptable. Also, 90% of the participants agreed that HMA does not affect the user experience of the apps that it protects, and that they are at least somewhat interested in using HMA. These results suggest that HMA is usable and desirable.

As most lab experiments, our study has some limitations, which we acknowledge here. However, it should be noted that the purpose of our study was not to evaluate the design and the usability of the solution as a final product but rather to evaluate, based on a proof of concept with a basic UI, the interest and the perception of users in terms of the general approach and its technical implications (e.g., delays). First, the participants used phones provided by the experimenters, with only two apps and in a hypothetical scenario. Second, the users were provided with detailed instructions; therefore, we did not test how intuitive the proposed solution is. Finally, when they took the exit survey, the participants had already been briefed about the privacy problems of LIA, which might have introduced a bias.

2.11 Discussion

In this section, we discuss some shortcomings of HMA and how HMA can be extended to be used in other alternative scenarios.

2.11.1 HMA Limitations

As explained in Sections 2.9 and 2.10, HMA effectively hides the mandatory features and runtime information of sensitive apps. However, as sensitive apps hide their presence, optional features that help them to intentionally expose themselves to other apps, including content providers and implicit intents, should not be used. This can limit their data-sharing capabilities. A possible workaround for this problem is to put apps that

want to share data with each other in the same container app. In addition, due to the limitation of user-level virtualization library that HMA uses, if sensitive apps use many customizations with third-party SDKs, the container apps might fail to launch them. This might be a problem with complex apps, such as dating apps, but mHealth apps are unlikely to require many customizations. In the scenario envisioned by HMA, developers build apps for hospitals, thus the HMA App Store can provide developers with a guideline about supported features so that their apps are HMA-compatible. To facilitate guideline compliance, the HMA App Store can also provide developers with IDE plugins to help them write compliant code; such an approach was proposed in existing work (e.g., [83, 84]). Lastly, if the number of apps in the HMA App Store is small, a nosy app could guess the name of the sensitive app. To increase the anonymity-set, the hospitals can include simpler, non-sensitive apps (e.g., informational) in the HMA App Store, and the popularity of apps on the store should not be public.

2.11.2 An Alternative Scenario

HMA is designed to work for the case of hospitals hosting their mHealth apps. However, there are other categories of apps that can be considered sensitive, such as apps about religion or sexual orientation. In some countries, people would be arrested if they use a gay dating app [93]. Moreover, developers might publish their mHealth and other sensitive apps on app stores other than the HMA App Store, such as the Amazon or Google Play app stores. In such cases, the community or an organization can run a proxy service that retrieves the APKs of these apps from these stores and create container apps for them accordingly; this service can be free or for a fee. This might require an agreement between the proxy service and the app developers. Also, as discussed in Section 2.11.1, some apps might have optional features that are not recommended or features that are not supported by user-level virtualization techniques. Similarly to the case of the Amazon app store, the proxy service can provide developers with a testing service to test the compatibility of their apps.

2.12 Conclusion

In this chapter, we have systematically investigated the problem of apps fingerprinting other installed apps. We have shown that apps can collect a significant amount of static and runtime information about other apps to fingerprint them. We have also quantified the prevalence of the problem and have shown that many third parties and apps are interested in learning the list of installed apps on people's phones. Moreover, our analysis has shown that there are no existing mechanisms for hiding the presence of a sensitive app from other apps. Therefore, we have proposed HMA, the first solution that addresses this problem. HMA does not require any modifications to the Android OS and it preserves the key functionality of apps. Our thorough evaluation of HMA on a diverse data-set of both free and paid mHealth apps from the Google Play Store has shown that HMA is practical with reasonable operational delays. Moreover, the results of our user study have suggested that HMA is perceived as usable and of interest to the users.

Chapter 3

A Privacy-Preserving Yet Accountable Ride-Hailing Service

In Chapter 2, we showed that smartphones do not provide adequate privacy protection for certain apps. As a result, the use of such apps introduces privacy risks for users. In this chapter and Chapter 4, we look at privacy threats from a different perspective: due to the lack of privacy by design, apps and online service providers often put users' privacy at risk. That is, they often over-collect information that might not be needed for the functionality of the services, and the collected information can be used for purposes that users did not consent to. We will show that it is possible to design systems that gracefully balance the users' privacy and the services' usability. We will begin with the case of ride-hailing services (e.g., Uber and Lyft). Afterwards, in Chapter 4, we will look into the case of activity-tracking services (e.g., RunKeeper).

3.1 Introduction

Ride-hailing services (RHSs), such as Uber and Lyft, enable millions of riders and drivers worldwide to set up rides via their smartphones [8]. Their advantage over traditional taxi services is due to the convenience of their services: ride requests at the touch of a button, fare estimation, automatic payments, and reputation ratings. Moreover, the accountability provided by RHSs is a key feature for riders and drivers, as it makes them feel safer [94, 95]. For instance, in case of a criminal investigation, the RHS provider can provide law-enforcement agencies with the location trace of a particular ride and the identities of the participants.

To offer such services, however, RHSs collect a vast amount of sensitive information, which puts at risk the privacy of riders and drivers. First, for each ride, the location traces and the rider's and driver's identities are known to the service provider (SP). As a result, the SP, or any entity with access to this data, can infer sensitive information about riders' activities such as one-night stands [13], monitor the locations of riders in real time for entertainment [96], track the whereabouts of their ex-lovers [97], look up trip information of celebrities [98], and even mount revenge attacks against journalists critical of such services [99]. In the case of drivers, there are reports of SPs that track drivers to

find if the drivers attended protests [100]. Second, due to the release of drivers' personal identifiable information (PII) early in the ride set-up procedure, an outsider adversary can massively collect drivers' PII [17]. Therefore, there is a strong need to provide privacy and anonymity for both riders and drivers with respect to the SP and each other.

In this chapter, we analyze privacy threats in RHSs and develop a threat taxonomy. We then focus on designing protection mechanisms for defending against two critical threats: location tracking of riders by the SP and harvesting of drivers' personal information by an external adversary. To target these threats, we present ORide, a privacy-preserving RHS that enables the SP to efficiently match riders and drivers without learning either their identities or their locations, while providing accountability to deter misbehavior. ORide provides strong privacy for both riders and drivers, i.e., all users in the system are part of large anonymity sets, even if they are in sparsely populated areas. Even in the extreme case of targeted attacks (i.e., a curious SP wants to know the destination of a ride taken by a specific rider given the time and location of her pick-up event [101]), the location privacy of the rider's destination is still guaranteed. For this purpose, ORide relies on a state-of-the-art somewhat-homomorphic encryption system [102] (SHE), to which we apply optimizations for ciphertext packing and transformed processing [103], hence enabling a notable boost in performance and a reduction in overhead with respect to naive cryptographic solutions.

Accountability and usability are often considered to be as important as privacy in RHSs [94, 95]; this introduces challenges in resolving the uneasy tension between privacy, accountability and usability. To achieve accountable privacy, ORide enables the SP to revoke, when needed, the anonymity of misbehaving riders or drivers. However, the SP does not have full control over this re-identification operation, i.e., it is able to do it only with the support from the affected party. In addition, to preserve the convenience of the service, ORide supports automatic payment through credit cards and enables riders to contact drivers for lost items. ORide also preserves the reputation-rating operations of current RHSs.

The evaluation of ORide by using real data-sets from NYC taxi cabs [104] shows that, even with strong bit-security of 112 bits, ORide introduces acceptable computational and bandwidth overheads for riders, drivers and the SP. For example, for each ride request, a rider needs to download only one ciphertext of size 186 KB with a computational overhead of less than ten milliseconds. ORide also provides large anonymity sets for riders at the cost of acceptable bandwidth requirements for the drivers: For instance, for rides in the boroughs of Queens and Bronx, a ride would have an anonymity set of about 26,000, and the drivers are only required to have a data-connection speed of less than 2 Mbps.

In summary, in this chapter, we make the following contributions:

- *A threat taxonomy for RHSs.* We present the first general privacy analysis of RHSs. By analyzing currently deployed RHSs, we develop a threat taxonomy and identify high-risk threats, particularly the unreported threat of drivers' personal information being harvested.

- *A novel, oblivious, and efficient ride-matching mechanism.* To match riders and drivers without revealing their identities and locations to the SP, ORide includes a novel protocol based on quantum-resistant SHE. We optimize our SHE-based

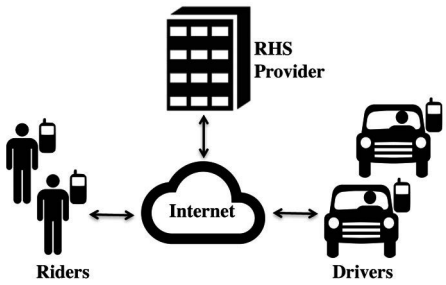


Figure 3.1: Ride-hailing services overview.

protocol to considerably reduce the bandwidth requirements and the processing overhead, compared to a vanilla SHE-based protocol; and we propose an efficient extension to deal with malicious drivers.

- *The design and prototype of ORide.* ORide supports the matching of riders and drivers, different accountability mechanisms, and it reduces the amount of sensitive information revealed to the SP. In particular, ORide supports functionalities that are often also considered as important as privacy, such as credit-card payment, reputation rating, contacting drivers in case of lost items and traceability in case of a criminal activity during a ride.

- *Thorough performance evaluation.* Using real data-sets and robust security parameters (112 bits security), we show that ORide provides strong privacy guarantees for riders and drivers. The computational and network overhead introduced by ORide is also practical for riders, drivers and SP. We also show that ORide has a negligible effect on the accuracy of matching riders and drivers, compared with current RHSs. The source code of our prototype implementation is available at [105].

3.2 Ride-Hailing Services

In general, RHSs involve three parties: riders, drivers, and a service provider (SP) (see Figure 3.1). The SP handles incoming ride-requests and matches riders with available drivers, based primarily on their locations; it also offers key services such as fare estimation and calculation (based on the route of the rides), ride payment and reputation management. In exchange for these services, the SP charges a fee for each completed ride (e.g., Uber charges around 20% of the total fare). Some SPs also sell ride data or location traces to third parties (e.g., for city planning [106] or marketing [107]). To use a RHS, riders and drivers need an account, a GPS-equipped smartphone with the SP's mobile app installed, and an active Internet connection.

To hail a ride, the rider sends a request to the SP by using the mobile app. In some countries, before requesting a ride, the SP can confirm upfront with the riders about the fare, given her desired pick-up and drop-off locations. A ride request includes the rider's identity and the precise pick-up, and (optionally) drop-off locations. The SP selects an available driver, based on her proximity to the pick-up location (while on duty, drivers continuously report their locations to the SP). The SP sends the request, together with the rider's username, reputation and phone number (for ride coordination) to the driver.

If the driver declines the request, the SP sends the request to another available driver. If the driver accepts the request, the SP notifies the rider and sends her back the driver's information such as the driver's real name, photo, phone number, and car plate number. The rider uses this information to decide if she accepts the ride, as well as to coordinate the pick-up event.

As the driver approaches the pick-up location, the SP shares the driver's location and estimated time of arrival with the rider. Once the rider is in the car, the driver notifies the SP that the ride has begun. Both the rider and the driver can cancel a ride at any point. However, in certain cases, the SP can charge them a penalty fee or lower their reputations [108] (e.g., in case of systematic ride cancelation). Unlike the rider, the driver must keep her smartphone switched on (with Internet and GPS connectivity) during the ride, in order to report her location to the SP. Upon reaching the destination, the driver notifies the SP that the ride has ended. The SP automatically charges the rider (e.g., her credit card); the amount charged to the rider (minus the service fees) is credited to the driver's account. Both the driver and rider can check the details of the ride via the mobile app and rate each other. This operation is optional and it can be performed within a predefined period of time that begins right after the end of the ride. The SP uses the reputation information to keep the quality of the service high by detecting and punishing misbehaving parties. For example, riders and drivers with low reputations could be banned temporarily or permanently.

3.3 Privacy and Security Analysis of RHSs

In this section, we present a privacy and integrity analysis of current RHSs and identify the most critical privacy threats in these services.

3.3.1 Adversarial Model

We define four adversaries in current RHSs:

Rider. Active adversary who might attempt to harvest personally identifiable information (PII) from drivers (e.g., for stalking or blackmailing purposes).

Driver. Active adversary who might attempt to collect PII from riders she picks up (e.g., for stalking or blackmailing purposes).

SP. Passive adversary that strives to safeguard its business and maximize its profits. It has incentives to profile riders and drivers and infer sensitive information about them, to either improve its own services or to monetize harvested data (e.g., for advertisement purposes or coercion [99]). However, it also has incentives to prevent certain attacks, such as data harvesting from outsiders, or pervasive fare overcharging, as these attacks threaten the viability of the SP's business over its competitors. Finally, we assume the SP has no incentive to actively *cheat* (e.g., by providing a malicious app to users, or otherwise deviating from an established protocol), if there is a non-negligible chance of the SP being caught in the act. Indeed, the risk of public exposure and reputation loss is a strong economic deterrent against such attacks from the SP.

Outsider. Active adversary who tries to collect and/or steal riders' and drivers' PII, account credentials, and ride data. The malicious outsider might have more resources than a single driver or rider. For instance, it could be a competitor SP, a criminal organization, or a group of regular taxi drivers.

Privacy Threats

Description		Risk-RH
1) R→ D	PII harvesting [109]	Low
2) D→ R	PII harvesting	Low
3) **SP→R**	**location tracking [13, 96, 110]**	**High**
4) SP→ D	location tracking [111, 100]	Medium
5) O→ R	PII harvesting	Medium
6) **O→ D**	**PII harvesting**	**High**
7) **O→ SP**	**PII and ride data breach [112]**	**High**

Integrity Threats

Description		Risk-RH
1) R→ D	fare undercharging [113]	Low
2) R→ SP	reputation cheating	Low
3) R→ SP	incentives abuse [114]	Low
4) **D →R**	**fare overcharging [115]**	**High**
5) D→ D	ride matching cheating [116]	Low
6) D→ SP	reputation cheating	Low
7) D→ SP	fees and incentives cheating [117]	Medium
8) **O→ R,D**	**account theft [118, 119]**	**High**

Table 3.1: Threat taxonomy for RHSs (Risk-RH) based on four possible adversaries: riders (R), drivers (D), service provider (SP), and outsiders (O). The notation $X \rightarrow Y$ means *X attacks Y*.

3.3.2 Threat Taxonomy

We built the first threat taxonomy for RHSs (Table 3.1) with a focus on privacy and integrity threats. Privacy threats affect the personal information of riders and drivers (e.g., PII, ride data) and integrity threats affect the correctness of the services offered by the SP (e.g., fare calculation, ride matching, reputation systems). To identify these threats, we reviewed online sources (e.g., news articles, RHSs websites, forums, blog posts) and experimentally evaluated some of the most popular RHSs, see Section 3.3.3 for more details. To estimate the level of risk associated with each adversary and threat, we followed the OWASP risk-rating methodology where the risk is defined as $Risk = Impact * Likelihood$ [120]. In our risk assessment, we assume that only a small subset of the riders and drivers are malicious. This is a reasonable assumption, given the current success of RHSs.

In general, threats with low risks involve attacks that are not scalable, offer limited rewards, and can be deterred by existing mechanisms such as decreased reputation [121, 108]. Threats with medium risks involve attacks that have relatively higher impact, offer higher rewards, and are more likely to happen. Threats with high risks involve attacks that have the highest impact and reward, have the highest likelihood and reported incidents, and for which current defense mechanisms are insufficient. Appendix A details each threat and justifies each risk level.

From the high-risk threats in RHSs in Table 3.1, privacy threat 7 and integrity threat 8 are not exclusive to RHSs; data breaches and account theft can affect almost any online service. Hence, there are existing mechanisms for reducing these risks. For example, the SP database could use CryptDB [122] to securely store riders' and drivers' data.[1]

[1]Inference attacks against CryptDB [123] do not apply in RHSs as there is no "auxiliary database" available to adversaries.

Therefore, in this work we focus on the high-risk threats that occur exclusively due to the design of current RHSs and for which there are no current solutions.

SP→R Location Tracking. By design, a SP can track, in real time or offline, riders' precise locations during their rides and infer private information from such data. Compared with other forms of public transportation, ride-hailing data can reveal significantly more private information about millions of riders. For example, the SP and other parties with access to this data (e.g., by agreement, coercion or attack) can determine where riders live, work, socialize, where exactly they go [124, 125, 126], even for one-night stands [13]. Furthermore, there are reported incidents of SP's employees abusing such data to track riders for entertainment [96] or revenge [99].

O→D PII Harvesting. A *malicious outsider* could exploit privacy weaknesses in the services offered by the SP to efficiently collect PII of a large number of drivers. A particular weakness that could be exploited for this purpose is that, in current RHSs, the driver's information is revealed to the rider *before* the ride begins in order to coordinate the pick-up event. Hence, an adversary could efficiently collect drivers' PII in a particular area by using fake rider accounts, selecting different pickup locations, requesting and then canceling rides. This attack can be used by shady marketers (e.g., loan sharks) or, even worse, angry taxi-cab drivers trying to physically harm RHSs' drivers [127]. The possibility of this attack was demonstrated when Uber and Lyft employees harvested, from each other, information about thousand of drivers for recruitment purposes [128, 129]. To defend against this attack, the SP could define thresholds on the number of canceled requests that a user can make to the service. If a user passes this threshold, she is banned from the service. However, an adversary could bypass such measures by creating more fake accounts (they are relatively easy to create) or by buying stolen accounts on online black markets [130]. In Section 3.7, we present an effective solution for this unreported high-risk threat.

D→R Fare Overcharging. In case the fare is not calculated upfront by the SP, a *malicious driver* could exploit weaknesses in the fare calculation service offered by the SP to unduly charge riders a higher fare. In most RHSs, drivers use their personal smartphones to report to the SP the distance and duration of each ride for fare calculation. However, unlike the trusted hardware used for fare calculation in regular taxi cabs (i.e., taximeter), smartphones are general-purpose devices and are relatively easy to tamper with. Thus, a malicious driver could modify her smartphone to report longer ride distances to the SP and, as a result, charge a higher fare. Malicious drivers do not need high technical skills to perform such attacks, as they could acquire modified devices [117] or access tools on the Internet. The SP's mobile app could perform some checks to detect such attacks, but there are also ways to avoid detection (see Section 3.3.3 for more detail about the feasibility of this attack). Moreover, the SP could also provide drivers with restricted smartphones; this approach, however, is less scalable and unpopular among drivers.

3.3.3 Assessment of Popular RHSs

In this section, we present our experimental evaluation of popular RHSs. In particular, we assess how well these services deal with the three high-risk and RHS-specific threats described in Section 3.3.2. For this purpose, we installed the rider and driver Android

Provider	SP→R Location tracking	O→D PII harvesting	D→R Fare overcharging
Uber	○	◐	◐
Ola Cab	○	○	◐
Lyft	○	◐	◐
GrabTaxi	○	◐	○
EasyTaxi	○	○	○

Table 3.2: Robustness level of popular RHSs against high-risk threats (Section 3.3.2). Current RHSs provide no protection (○) or only partial protection (◐) against these threats.

mobile app[2] of the selected RHSs and, when possible, made ride requests. In addition, we reviewed the online documentation of the selected RHSs. Table 3.2 presents the results.

For the *SP→R location tracking* threat, we used rider apps to make ride requests to RHSs. From these requests, we learnt the information that the riders have to present to the SPs and the information that the SPs return to the riders. Unsurprisingly, we did not find any evidence that the evaluated SPs provide any sort of privacy-preserving mechanisms (e.g., pseudonyms, obfuscated ride traces or ride summaries instead of full location traces) for protecting the riders from inference attacks. All the evaluated SPs collect the full location trace of each ride, together with the riders' and drivers' identities.

For the *O→D PII harvesting* threat, we determined how much drivers' information is revealed to the rider by the SP when she makes a ride request. First, all the SPs reveal the driver's name, phone number, current location, and car's plate number; some SPs also reveal the driver's photo and car model and picture. Second, we checked how SPs punish riders if they cancel rides too often (e.g., penalties or lower reputation). Only Uber [121], Grabtaxi, and Lyft [108] provide such types of penalties; this can partially deter harvesting attacks. Reported incidents [128, 129, 131] demonstrate that such mechanisms are insufficient.

For the *D→R fare overcharging* threat, we determined, for each SP, whether the mobile app for drivers checks if the mock location API [132] is enabled or if the device is rooted. Only the Uber's driver app checked for both conditions, whereas Lyft only checked for the first condition. Still, these checks could be bypassed with tools that are already available for Android [133] and iOS [134]. Moreover, only Uber and Ola Cab offer the option of providing drivers with restricted smartphones. However, this is optional and country dependent.

3.4 Related Work on Privacy-Preserving RHS Design

Researchers have proposed different privacy-enhancing solutions for ride sharing (i.e., car pooling) services [135, 136, 137, 138, 139] and public transportation ticketing systems [140, 141, 142]. However, little work exists in the area of privacy and security for RHSs, probably due to their relative novelty. According to our literature review, the most relevant work in this area is PrivateRide [17].

PrivateRide is our first attempt to design a solution to enhance location privacy for riders and protect drivers' information from harvesting attacks. PrivateRide gracefully balances the trade-off between privacy guarantees and the convenience of the services.

[2]We assume RHSs apps offer equivalent privacy and security mechanisms on different mobile platforms.

However, it has several limitations that are addressed in this work. First, due to the use of static location cloaking, PrivateRide cannot guarantee the same level of privacy to all riders, because the size of the anonymity set in a particular cloaked area depends on the density of riders in that area. For instance, the anonymity set is smaller for ride requests in areas outside a city center. Also, the tradeoff between the size of a cloaked area and the accuracy of the ride-matching results prevents the use of larger cloaking areas (to achieve larger anonymity sets). Second, PrivateRide does not protect drivers' privacy, also important [100]. Third, PrivateRide provides limited accountability features to deal with relatively common scenarios such as drivers and riders physically attacking each other (i.e., safety concerns) or items being lost during a ride; for many users, such features can be as important as their privacy. Fourth, PrivateRide's usability is reduced with respect to current RHSs because the supported payment mechanism is less convenient (i.e., PrivateRide requires payments with e-cash bought in advance before a ride). Moreover, ride-matching in PrivateRide is suboptimal, because the distance between rider and drivers is estimated using the centers of the cloaked areas, instead of precise locations, resulting in additional waiting time for riders.

3.5 Our System: ORide

Our goal is to design a RHS that provides strong privacy guarantees to both riders and drivers, as well as equivalent usability and accountability compared with current RHSs (e.g., Uber, Lyft, and Easy Taxi). To do so, we assume a system consisting of three parties: riders, drivers and the service provider (SP). We now describe our adversarial and system assumptions.

3.5.1 Adversarial Assumptions

In our model, riders and drivers are active adversaries, and the SP is a passive adversary. We assume that most riders and drivers do not collude with the SP, as drivers are independent contractors rather than SP's employees. The case of a covertly active SP is discussed in Section 3.9.2. In such a case, we assume that the SP does not provide riders and drivers with malicious apps. This is a reasonable assumption, because such attacks can be detected by third parties via reverse-engineering or black-box analyses; the risk of public exposure and reputation loss is a strong deterrent against such attacks.

Given that they have been observed in current RHSs (see Section 3.3), we focus on the following attacks:

- *(A1)* The riders and drivers might attempt to assault each other [143]; in extreme cases, a driver might attempt to kidnap and/or kill a rider, or vice versa [144, 145].
- *(A2)* The SP uses its knowledge about side information about riders and drivers, including their home/work addresses, together with protocol transcripts, to perform *large-scale* inference attacks in order to profile riders' and drivers' activities [13].
- *(A3)* The SP might attempt to carry out *targeted attacks* on specific riders. That is, besides their home/work addresses, the SP knows the precise pick-up location and time of a specific rider and wants to know the drop-off location and time of this ride, or vice versa [98, 99, 101].

- *(A4)* A malicious outsider might attempt to massively collect driver's PII (e.g., drivers' vehicle information).

3.5.2 Design Goals

The goal of ORide is to defend against the attacks listed in Section 3.5.1, and to offer the same level of accountability and usability as current RHSs, as follows.

- Riders and drivers are held accountable for their behaviors during their rides. That is, the SP is able to identify misbehaving riders or drivers when needed, for instance, if one party attacks the other. However, the SP is able to identify the misbehaving party only with support from the affected party (or her trusted contacts, see Section 3.8.)
- The system preserves the convenience and usability properties offered by current RHSs, such as payment through credit cards and reputation rating. In addition, once a rider is matched with a driver, the rider can track the location of the driver when she is driving towards the pick-up location, and they can contact each other to coordinate the pick-up event. The system also enables riders to contact drivers of their past rides to find lost items.

3.5.3 System Assumptions

We assume that the metadata of the network and lower communication layers cannot be used to identify riders and drivers or to link their activities. Such an assumption is reasonable because, in most cases, the smartphones of drivers and riders do not have fixed public IP addresses; they access the Internet via a NAT gateway offered by their cellular provider. If needed, a VPN proxy or Tor could be used to hide network identifiers.

We also assume that, besides localization capabilities, the riders' and drivers' smartphones support peer-to-peer wireless communication, e.g., Bluetooth and WiFi Direct. And for all location-based computations, the apps use a coordinate system such that the Euclidean distances correspond to the great-circle distances, e.g., by using map-projection systems for local areas such as UTM [146] to convert a pair of (latitude, longitude) to planar coordinates (x, y). Moreover, drivers use a navigation app that does not leak their locations to the SP. This can be done by using a third-party navigation/traffic app (e.g., Google Maps, TomTom, Garmin) or pre-fetching the map of their operating areas (e.g., a city) and using the navigation app in off-line mode.

3.5.4 Notation

Throughout the rest of this work, we denote polynomials and scalar values with lowercase letters, variables and rings with uppercase letters, and vectors with boldface letters. $\lfloor . \rceil$ denotes rounding to the nearest integer. A polynomial of degree $(d-1)$ will be interchangeably denoted as $a = \sum_{i=0}^{d-1} a_i X^i$ or in its vector form $\boldsymbol{a}$ when there is no ambiguity. The used symbols and terms are summarized in Table 3.3.

3.5.5 Cryptographic Primitives

In this section, we briefly describe the cryptographic building blocks used in ORide.

Notation	Description
k_s	Ephemeral private key
$\boldsymbol{k}_p$	Ephemeral public key
$cert_X$	Public-key certificate of X
sk_X	Public-key associated with the certificate of X
pk_X	Private-key associated with the certificate of X
loc_X	Planar coordinates of X, $loc_X = (x_X, y_X)$
n	Number of available drivers
d	Degree of the polynomial
dt	Deposit token
r_{dt}	A random number to create a deposit token
z	A geographical zone (e.g., a city)
$sig_X\{m\}$	Message m and signature of X on m
$Bsig_{SP}(m)$	Blind signature of the SP on message m
$sig_{R-D}\{m\}$	Message m is signed by both the Rider and the Driver i.e., $sig_D\{sig_R\{m\}\}$

Table 3.3: Table of notations

Blind Signatures

A blind-signature scheme [147] is a form of digital-signature schemes in which the signer does not know the content of the message that she is signing. This is achieved by enabling the signature requester to 'blind' (i.e., randomize) the message before sending it to the signer. When the signature requester receives the signature on her blinded message, she 'unblinds' it to obtain a valid signature for the original message. The signer can verify the signature of an unblinded message but she is not able to link this message back to the blinded version she signed.

Anonymous Credentials

An anonymous credential (AC) is a cryptographic token with which the credential owner can prove to another party that she satisfies certain properties without revealing her real identity. In ORide, a user is identified when she obtains ACs from the SP. However, when she wants to start an anonymous session, she reveals to the SP only the expiration date and the role specified in the AC (rider or driver), and she proves to the SP that, in a zero-knowledge fashion, she knows the private key associated with the AC. To prove her reputation to a driver, a rider reveals to the driver the reputation score specified in her AC, together with the proof to show that the revealed value is trustworthy. ORide relies on the Anonymous Credentials Light (ACL) [148]. However, note that ACL is a linkable anonymous credential scheme, i.e., a user can only use a credential once to avoid her transactions from being linkable.

Somewhat-Homomorphic Encryption

Somewhat-Homomorphic Encryption (SHE) is a special kind of malleable encryption that allows a certain number of operations (additions and multiplications) to be made over ciphertexts, without the need to decrypt them first. All SHE cryptosystems present semantic security, i.e., it is not (computationally) possible to know if two different encryptions conceal the same plaintext. Therefore, it is possible for a party without the private key (in our case, the SP), to operate on the ciphertexts produced by riders and

drivers, without obtaining any information about the plaintext values. Additionally, we choose one of the most recent and efficient SHE schemes based on ideal lattices, the FV scheme [102]. This scheme relies on the hardness of the Ring Learning with Errors (RLWE) problem [149]. Note that whenever working with cryptosystems based on finite rings, we usually work with integer numbers, hence, from here on, we will assume that all inputs are adequately quantized as integers. Here, we briefly describe the main functions of the FV scheme.

For plaintext elements in a polynomial quotient ring $m \in R_t = \mathbb{Z}_t[X]/(X^d + 1)$ and ciphertext elements in $R_q = \mathbb{Z}_q[X]/(X^d + 1)$, where q and t are positive integers $q > t$ defining the upperbound of the ciphertext and plaintext coefficients, respectively. Let $\Delta = \lfloor q/t \rfloor$ and χ_k, χ_n be two *short* noise random distributions in R_q, the FV encryption of a message $m \in R_t$ with secret key $k_s = s \sim \chi_k$ and public key $\boldsymbol{k}_p = [p_0, p_1] = [(-a \cdot s + e), a] \in R_q^2$, with e drawn from χ_n and a randomly chosen in R_q, generated by FV.GenKeys, results in a vector expressed as

$$\boldsymbol{c} = \mathsf{FV.Enc}(\boldsymbol{k}_p, m) = [p_0 \cdot u + e_1 + \Delta \cdot m,\ p_1 \cdot u + e_2], \tag{3.1}$$

where u is drawn from χ_k, and e_1, e_2 are short random polynomials from the error distribution χ_n. All operations are in R_q.

Decryption of a ciphertext $\boldsymbol{c} = [c_0, c_1]$ works as

$$m = \mathsf{FV.Dec}(k_s, \boldsymbol{c}) = (\lfloor t \cdot [c_0 + c_1 \cdot s \bmod q]/q \rceil) \bmod t.$$

The scheme enables us to seamlessly add (FV.Add), subtract (FV.Sub) and multiply (FV.Mul) two encryptions to obtain the encryption of the added, subtracted, and multiplied plaintexts respectively; multiplications consider the encryptions as polynomials in v: $[c_0, c_1] \rightarrow c_o + c_1 \cdot v$, such that the product between $\boldsymbol{c}$ and $\boldsymbol{c}'$ is evaluated as $[c_0, c_1] \cdot [c'_0, c'_1] \rightarrow c_0 \cdot c'_0 + (c_0 \cdot c'_1 + c_1 \cdot c'_0)v + c_1 \cdot c'_1 \cdot v^2 \rightarrow [c''_0, c''_1, c''_2]$, which results in a ciphertext in R_q^3, with one extra polynomial. It is possible to recover a fresh-like encryption with two polynomials by employing a relinearization primitive, which requires the usage of a matrix (relinearization key) composed of encrypted pieces of the secret key (we refer the reader to [102] for further details).

3.6 Oblivious Ride-Matching Protocol

One of the challenges in privacy-preserving RHSs is how to efficiently match ride requests to ride offers, without revealing the riders' and drivers' locations to each other and to the SP. For this, ORide relies on somewhat-homomorphic encryption (see Section 3.5.5) where the riders and drivers send their encrypted locations to the SP, from which the SP computes the encrypted squared Euclidean distances between them. We detail this in the following sections.

3.6.1 Naive Approach

SHE can be applied to the ride-matching problem in RHSs as follows. When a rider wants to make a ride request, she generates an ephemeral FV public/private key-pair together with a relinearization key. She uses the public key to encrypt her planar coordinates and obtains their encrypted forms. She then informs the SP about the zone of her pick-up location, the public and relinearization keys, and her encrypted planar coordinates.

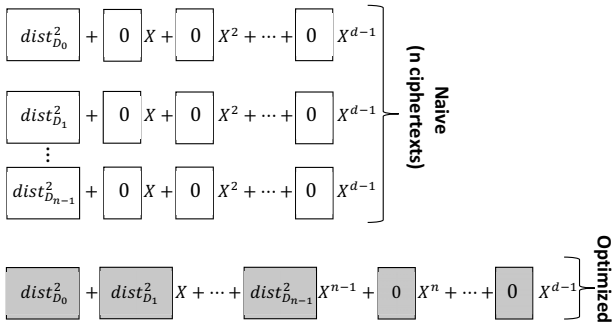


Figure 3.2: Our optimized ride-matching approach enables the SP to send to the rider a single ciphertext containing all the squared Euclidean distances ($dist^2_{D_i}$) between the rider and available drivers as opposed to one ciphertext per driver (naive approach). Note that $dist^2_{D_i} \geq dist^2_{D_j} \Leftrightarrow dist_{D_i} \geq dist_{D_j}$, hence instead of using Euclidean distances for comparisons, we can use squared Euclidean distances.

When this information arrives at the SP, the SP broadcasts the public key to all drivers available in that zone. Each driver uses the public key to encrypt their planar coordinates and sends them to the SP. The SP computes, based on their encrypted coordinates, the encrypted squared-Euclidean-distances between the rider and the drivers, and it returns the encrypted squared-Euclidean-distances to the rider, from which the rider can decrypt and select the best match, e.g., the driver who is the closest to her pick-up location.

However, due to the high ciphertext expansion, a naive use of SHE would incur impractical computational and bandwidth costs for the riders and the SP. Furthermore, for each ride request, the SP would need to separately compute the encrypted squared-Euclidean-distances between the rider and each of the drivers: For n drivers, this would mean n squared-distance calculations between encrypted polynomials of d coefficients each, and n encrypted squared-Euclidean-distances returned to the rider. This would incur an unfeasible overhead in terms of computations for the SP, consequently delaying the ride-matching for the rider and causing a considerable bandwidth overhead at the rider-SP link, e.g., hundreds of MBs if the system has several thousand drivers (see Section 3.11.3).

3.6.2 Optimized Approach

In order to enable the SP to operate on d elements of $\mathbb{Z}_t$ packed as a polynomial in R_t in a *single* ciphertext, such that each encrypted operation affects all the coefficients in parallel (see Fig. 3.2), we propose two optimizations: ciphertext packing and transform processing. When the rider decrypts this ciphertext, she can recover these d values by looking at all the coefficients. From here on, we assume that $d \geq n$, which will usually be the case due to the security bounds on d (see Section 3.10); in other cases, $\lceil n/d \rceil$ encryptions can be used to analogously pack the whole set of squared Euclidean distances.

First, ciphertext packing enables the SP to pack n ciphertext squared-distances into one ciphertext, hence reducing the bandwidth overhead. But this is not enough for our goal. As we show in Section 3.7.4, we use all the n packed encrypted planar coordinates from the drivers independently of each other to homomorphically calculate all the squared-distances in the same encrypted operation, so we need coefficient-wise

homomorphic operations. While polynomial additions and subtractions are naturally coefficient-wise, polynomial multiplication in R_t (and its homomorphic counterpart in R_q) is a *convolution product* of the coefficients. A well-known method for transforming convolution products into coefficient-wise products (and vice-versa) in polynomial rings is the *Number-Theoretic Transform* (NTT) [103], a Fourier transform specialized for finite fields. This transform is commonly used in the ciphertext space to speed up polynomial multiplications that are then implemented as coefficient-wise products. In our case, for the second optimization, in order that products in the encrypted domain be translated into coefficient-wise products in the plaintext domain, we apply an inverse-NTT to plaintexts before encryption and an NTT after decryption. The NTT does not affect additions and subtractions because it is linear. We note that the NTT exists only for certain values of d and t, in particular when t is a prime and d divides $t-1$ (see Section 3.11). To make operations in $\mathbb{Z}_t$ simulate operations in $\mathbb{N}$ on our values, we choose $d = 2^l$ as a power of two and t as a sufficiently large Proth prime (of the form $k2^l + 1$, see [103]) such that all squared Euclidean distances are less than t. As a result, we improve on both the bandwidth and the computation overhead.

Moreover, due to the low degree of the evaluated operations (squared Euclidean distances), we avoid the use of re-linearizations at the SP, which (a) reduces the need to generate and to send the relinearization key from the rider to the SP, (b) reduces the noise inside the encryptions, and (c) enables more efficient operations at the SP, at the cost of one extra polynomial to represent the encrypted squared-Euclidean-distance returned to the rider.

3.7 ORide Protocols

In this section, we present our solution, called ORide (Oblivious Ride). We begin with an overview of the system and then detail ORide operations.

3.7.1 ORide Overview

ORide provides strong location privacy and anonymity for riders and drivers and still guarantees service accountability, secure payment and reputation rating operations. For this purpose, the riders and the drivers must possess *ride prerequisites* (Section 3.7.2), including anonymous credentials (ACs), deposit tokens, and digital certificates issued by the SP. To participate in the system, both riders and drivers create anonymous sessions by *logging in to the service* (Section 3.7.3) with their respective ACs. Drivers periodically report to the SP the geographical zones where they are located. These zones are defined by the SP to balance the network load in the system and the size of the anonymity set of the zones (Section 3.11.4). Note that, in contrast to PrivateRide, expanding the size of a zone in ORide *does not* affect the performance of the ride-matching and fare-calculation operations (Section 3.7.4).

When a rider initiates a ride request, the SP, the rider and drivers are involved in a *ride set-up* procedure (Section 3.7.4) that matches the rider to a driver. In addition, as in current RHSs, the rider and the driver agree on the fare based on the estimated distance and duration of the ride [150, 151]. Some random time after the fare agreement, they terminate their anonymous sessions. When the ride is completed, the driver creates a new anonymous session and notifies the SP that she is available again. Note that

drop-off times and locations are not reported to the SP. Moreover, some time after the ride finishes, e.g., at the end of the day, the rider and driver perform *ride-payment and reputation-rating* operations (Section 3.7.5).

3.7.2 Ride Prerequisites

Digital Certificates. We assume each rider and driver has a digital certificate denoted as $cert_R$ or $cert_D$, issued by the SP at registration time. Each certificate contains a public key and a randomly generated ID. The SP can use this random ID to find the real identity of the certificate holder. Note that the digital certificates are not used by the riders and drivers to log in to the service, and they are not revealed to the SP during a ride. They are used by the riders and drivers to identify each other during the ride, as part of ORide's accountability mechanism (Section 3.8).

Anonymous Credentials. ORide relies on Anonymous Credentials Light (ACL) [148], a linkable anonymous credential system, i.e., a user should use an AC only once to avoid her transactions from being linkable. To use the service anonymously, each user (rider or driver) requests ACs in advance from the SP, using their digital certificate. Hereafter, we denote the anonymous credential for a user X as AC_X, where X is R for riders or D for drivers. Each AC_X contains the average reputation score rep_X, an expiration date exp_X, and the secret key sk_X associated with the public key pub_X in the digital certificate of the AC holder. To differentiate between riders and drivers in the system, an AC also contains a role attribute $role_X$, e.g., $role_X = 1$ if $X = D$, and $role_X = 0$ if $X = R$.

Note that to prevent the SP from de-anonymizing users by correlating the time an AC is issued with the time it is used, or by relying on the AC's expiration date, the user's app could automatically request ACs from the SP at a certain time (e.g., at midnight), and the expiration date is coarse-grained, e.g., all ACs issued in a day expire at the end of that day. The reputation scores cannot be used by the SP to de-anonymize the users, as they are never shown to the SP during the rides. Furthermore, to prevent users from abusing the system, the SP defines a threshold on the number of ACs a rider or driver can acquire per day.

Deposit Token. Each rider is required to possess a *deposit token* and give it to the SP at the beginning of a ride. In case of misbehavior, the token is not returned to the rider. A deposit token, denoted as dt, is worth a fixed amount of money defined by the SP. It is a random number generated by the rider, blindly signed by the SP (by using blind-signature schemes, e.g., [147]) such that the SP is not able to link a token it issued with a token spent by a rider. A rider deposits a token to the SP in the beginning of the ride, and she is issued a new token by the SP after the ride payment is successfully completed. Note that the driver is not required to make a deposit because, during the ride set-up operation, the rider and driver exchange their digital certificates with each other. Consequently, if the driver misbehaves, the SP can identify the driver by collaborating with the rider. We discuss this in more detail in Section 3.7.6.

3.7.3 Login to the Service

To use the service, the rider and the driver need to create anonymous sessions to the SP: to do so, they use their anonymous credentials AC_R and AC_D, respectively.

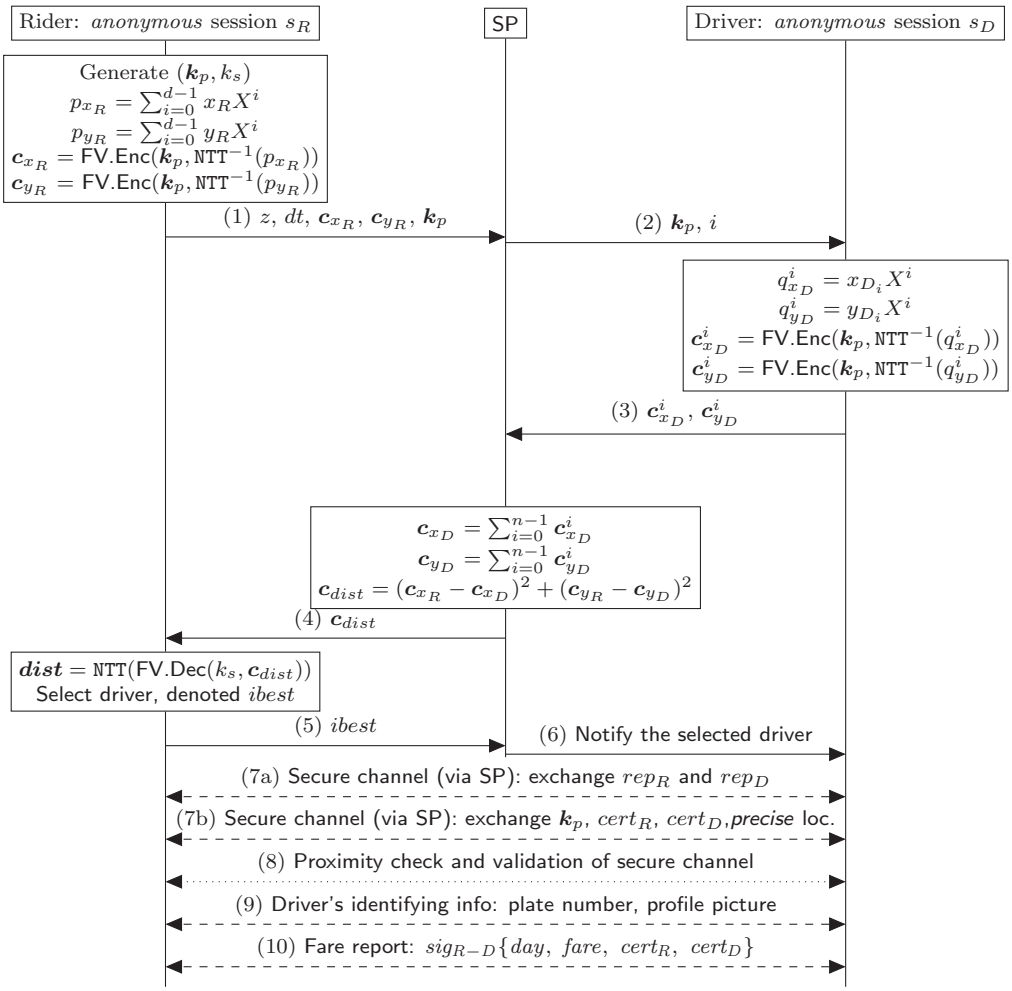


Figure 3.3: ORide ride setup protocol. The dashed arrows represent the secure channel (via the SP), and the dotted arrows represent the proximity channel.

Rider. The rider sends to the SP the rider-role $role_R$ and the expiry date exp_R stated in her AC_R. In addition, she proves to the SP that the claimed values are correct and that, in a zero-knowledge fashion, she knows the secret key sk_R tied to the AC_R.

Driver. Similarly to the rider, by using her AC_D, the driver follows the same aforementioned procedure to anonymously log in to the service.

In order to keep track of that session for coordination, the SP assigns a one-time session ID to each anonymous session. For the sake of simple exposition, hereafter, we exclude this one-time session ID from messages exchanged between the rider/driver and the SP.

3.7.4 Ride Set-Up

When a rider requests a ride, the operations performed by the rider, the drivers and the SP are as follows (see Fig. 3.3).

1. The rider generates an ephemeral FV public/private key pair, denoted as $(\boldsymbol{k}_p, k_s)$. She first computes the polynomial representations of the coordinates $p_{x_R} = \sum_{i=0}^{d-1} x_R X^i$ and $p_{y_R} = \sum_{i=0}^{d-1} y_R X^i$. She then applies the inverse-NTT on the polynomials and uses $\boldsymbol{k}_p$ to encrypt these values: $\boldsymbol{c}_{x_R} = \mathsf{FV.Enc}(\boldsymbol{k}_p, \mathtt{NTT}^{-1}(p_{x_R}))$, and analogously for $\boldsymbol{c}_{y_R}$. She then sends the zone of her pick-up location (denoted as z), deposit token dt, $\boldsymbol{k}_p$, $\boldsymbol{c}_{x_R}$ and $\boldsymbol{c}_{y_R}$ to the SP.

2. The SP checks the validity of the deposit token, i.e., if it has not been used before. If the token is valid, the SP adds it to the list of used tokens. It then sends to each driver in zone z a randomly permuted index $0 \leq i < n$ and the public key $\boldsymbol{k}_p$.

3. The i-th driver encodes her coordinates in the i-th coefficient: $q^i_{x_D} = x_{D_i} X^i$ and $q^i_{y_D} = y_{D_i} X^i$. Similarly to the rider, she applies the inverse-NTT, encrypts these values and sends them to the SP: $\boldsymbol{c}^i_{x_D} = \mathsf{FV.Enc}(\boldsymbol{k}_p, \mathtt{NTT}^{-1}(q^i_{x_D}))$, and analogously for $\boldsymbol{c}^i_{y_D}$.

4. The SP sums all drivers' ciphertexts by using the homomorphic property of the cryptosystem to pack them together: $\boldsymbol{c}_{x_D} = \sum_{i=0}^{n-1} \boldsymbol{c}^i_{x_D}$ and similarly for $\boldsymbol{c}_{y_D}$. It then homomorphically computes the n packed squared-values of the Euclidean distances between the n drivers and the rider in parallel, due to the packing $\boldsymbol{c}_{dist} = (\boldsymbol{c}_{x_R} - \boldsymbol{c}_{x_D})^2 + (\boldsymbol{c}_{y_R} - \boldsymbol{c}_{y_D})^2$, and it sends the result to the rider (see Fig. 3.2).

5. The rider decrypts the ciphertext and applies the NTT to obtain a squared Euclidean distance in each coefficient: $\boldsymbol{dist} = \mathtt{NTT}(\mathsf{FV.Dec}(k_s, \boldsymbol{c}_{dist}))$. Then, she selects the driver with the smallest distance.

6. The SP notifies the selected driver. If she declines the offer, the SP asks the rider to select a different driver; it repeats this operation, until one driver accepts. The SP confirms,to the rider and the driver, that they have been assigned to each other.

7a. The rider and the driver establish a secure channel via the SP, e.g., using the unauthenticated Diffie-Hellman protocol, to exchange data that should not be observed by the SP.[3] From the information used to derive the secret key of the secure channel, the rider and the driver compute a shared secret *pairing PIN*. This *pairing PIN* will be used for the proximity-check operation in Step 8. With this secure channel, the rider and the driver reveal their reputation scores to each other. The trustworthiness of the revealed values is proved by showing that they are indeed the values in the rider's and driver's ACs. If the rider's reputation is too low, the driver can abort the protocol at this step. Likewise, the rider can select another driver, by using the list of decrypted squared-Euclidean-distances she obtained in Step 5.

7b. Via the secure channel, the rider and the driver exchange their precise locations (loc_R and loc_D, respectively). In addition, they exchange their digital certificates ($cert_R$ and $cert_D$) with each other. This provides accountability for the rider and driver (see Section 3.8). Also, the driver can reveal to the rider the public key $\boldsymbol{k}_p$ that she used to encrypt her locations; this helps to detect possible man-in-the-middle attacks at Step 2 of the protocol by the SP.

[3] Detection of possible man-in-the-middle attacks by the SP is done in Step 8. Note that this check is needed only if the SP is an active adversary.

The driver drives from her current location loc_D to the pick-up location loc_R, using an off-line navigation app or a third-party navigation app (such as Google Maps or TomTom). She sends, in real time via the secure channel, her precise locations to the rider, thus the rider can track the movements of the car. Also, at this point, the rider and the driver can call or message each other through their ride-hailing apps, if needed.

8. When the rider and the driver are in proximity, the driver performs a proximity check to verify the *physical presence* of the rider before releasing her identifying information: they use a short-range wireless technology (e.g., Bluetooth or WiFi Direct) to set up a proximity channel using the *pairing PIN*. If the channel is successfully established, the driver can verify that the rider is in her proximity. This helps prevent drivers' PII from being harvested. If this step fails, the driver can decide to abort the protocol. Also, via the proximity channel, the rider and the driver can check whether the secure channel (established at Step 7a) was tampered with by the SP.

9. The driver releases her identifying information to the rider, including her vehicle's license plate number and her profile picture. This information helps the rider to identify the driver and her car and to prevent certain threats, such as fake drivers [152]. Therefore, it is needed when the rider is about to enter the car. That is, the required communication distance between the phones of the rider and the driver is small (e.g., several meters).

10. The rider and the driver create a *fare report*. A fare report is a token generated by the rider and driver; and at the end of the day, the driver deposits it to the SP to get paid (Section 3.7.5). A fare report is created as follows. The rider sends her drop-off location to the driver via the secure channel, they agree on the path and, based on the estimated path, they compute the fare. The rider and driver then use the private key associated with their certificates to sign a message that reports about the day of the ride, the fare and their certificates, i.e., *fare report* $= sig_{R-D}\{day,\ fare,\ cert_R,\ cert_D\}$. Note that this upfront-fare method has been implemented in current RHSs, such as in Uber [151] and in Lyft [150]. Once the driver receives the fare report from the rider, the ride begins. The rider's and driver's apps do not report any information to the SP at this step and during the ride. Also, to prevent the SP from inferring the starting time of the ride based on the interactions between the rider and the driver over the secure channel, the rider and driver can randomly send dummy information to each other through the secure channel. Also, some random time after the fare-report agreement, they terminate their anonymous sessions.

Intuitively, because the distances between the rider and drivers are computed based on their (encrypted) *precise* locations, expanding the size of the zone will not result in negative effects on the performance of the ride-matching and fare-calculation operations. In addition, with ciphertext packing, we reduce by a factor of n the communication between the SP and the rider. However, if the drivers are malicious, they could corrupt the inputs from other drivers. Furthermore, note that in Step 1 of the protocol, any valid rider can generate an ephemeral public/private key pair. Consequently, if the SP

is an active attacker, it could track the locations of the drivers, thus indirectly track the locations of the riders. We discuss solutions to these potential issues in Section 3.9.

3.7.5 Ride Payment and Reputation Rating

When the car arrives at the drop-off location, the driver creates a new anonymous session to the SP. This enables her to receive ride-request broadcasts from the SP. Note that the driver *does not* report to the SP that the ride is completed.

At the end of the day, the driver sends to the SP the fare report $sig_{R-D}\{day,\ fare,\ cert_R,\ cert_D\}$ she received during the ride set-up operation (step 10, Section 3.7.4). The SP checks the correctness of the rider certificate $cert_R$ in the fare report and the correctness of the signature. If they are valid, the SP charges the rider according to her payment method, e.g., credit card. It then subtracts the service fee, and deposits the remainder to the driver. The SP then notifies the rider about the payment and that a new deposit token is available. The rider generates a random number r_{dt}, blinds it to r'_{dt}, and sends r'_{dt} to the SP. The SP signs r'_{dt} (i.e., $dt' = sig_{SP}\{r'_{dt}\}$, and it sends this blind signature to the rider's account. The rider unblinds the signature to obtain the deposit token which she can use for her next ride. Note that this procedure can be done automatically by the rider's app.

Once the payment is successfully completed, the rider and driver can rate the reputation of each other, similarly to current RHSs. They can log in to the service with their real credentials and provide the reputation score for the party whom they rode with.

Note that ORide preserves the payment and reputation-rating operations of the current RHSs. Also, as both the rider and driver are anonymous during the ride, ORide does not require the rider and the driver to hide their identifying information from the SP during the payment and reputation-rating operations. However, it is important to note that, in order to prevent the SP from de-anonymizing the rider and the driver by correlating the time that a fare report is deposited with the drop-off event of the ride, the payment operation should *not* occur immediately after the ride, e.g., the drivers deposit the fare reports to the SP at the end of the day.

3.7.6 Ride Cancelation

As in current RHSs, a rider or a driver can cancel a ride at any time before or during the ride. This is, however, discouraged by the SP, because it can lead to malicious behavior: For example, once a rider and a driver are assigned to each other by the SP, they meet at the pick-up location and start the ride as normal; but, to avoid the service fee, the rider or the driver can send a cancelation notification to the SP. Therefore, similarly to current RHSs, if a rider or a driver cancels a ride a certain amount of time after the ride request, they should be penalized by the SP, for example, their reputation scores are lowered or fees are charged [153].

In ORide, when a rider cancels a ride, the SP can offer her two options: to lose her *deposit token* (i.e., pay a penalty) or to reveal her $cert_R$ and have her reputation score lowered. If a driver cancels a ride, the SP can ask the rider to reveal the $cert_D$, from which the SP can identify and penalize the driver according to its policy.

3.8 Accountability

In this section, we discuss the accountability goals (mentioned in Section 3.5.2) of ORide. This includes audit-trail mechanisms against the attack *A1* in Section 3.5.1 and additional features such as retrieval of lost items, assurance of payment, and integrity of the reputation-rating operation. Attacks *A2*, *A3* and *A4* are discussed in Section 3.10.

(A1) Accountability. ORide enables the rider and the driver to exchange, during the ride set-up procedure, their digital certificates, i.e., $cert_R$ and $cert_D$, respectively, and the fare report. This provides accountability for riders and drivers, i.e., an affected party can report to the SP the digital certificate of the attacker and the fare report, from which the SP can identify the attack in order to charge her a fee, lower her reputation and/or to support legal action. However, the SP is only able to identify the attacker with support from the affected party; likewise, the affected party cannot obtain the real identity of the attacker without support from the SP. This is because the certificates $cert_R$ and $cert_D$ contain only the pseudonyms and only the SP knows the mapping between the pseudonyms and the real identities of the certificate owners.

ORide enables the rider to share with her trusted peers the driver's certificate $cert_D$ and the fare report, via out-of-band channels such as messaging apps, or a plug-in in her rider app. Similarly, during the ride, via out-of-band channels, she can share her GPS trace with her friends using (k, l) threshold secret sharing [154], i.e., each GPS location point is split into l parts so that any k out of l parts reconstruct the original coordinate. The driver can follow the same mechanism. Such information can be shared with law enforcement in case riders or drivers disappear (e.g., kidnapping), as in current services. This is similar to the approach used in personal safety apps, such as Google Trusted Contacts [155].

ORide guarantees assurance of payment. A rider cannot avoid paying the fare of a ride, because the fare report contains her digital certificate $cert_R$ and the day of the ride. As the rider and driver agree on the fare and both sign it before the ride, they cannot subsequently increase or decrease this fare. However, they might collude to underpay the service fee to the SP, by agreeing on a small fare and paying the difference in cash. Yet in this case, ORide offers the same guarantees as current RHSs, because riders can already request a small ride through the application and then pay in cash for a longer ride, once they have met the driver.

Furthermore, the bilateral rating system enables the SP to ban abusive riders and drivers from the service. Neither a rider nor a driver can claim a better reputation for herself, because the proof for attributes in her AC will not be correct with respect to her falsely claimed reputation. They also cannot arbitrarily rate the reputation of each other, because a payment record is needed (the deposit of a *fare report*). As discussed in Section 3.7.6, similarly to current RHSs, ORide enables the SP to hold riders and drivers accountable for ride cancelations.

SP Incentives. From an economic perspective, ride-hailing service SPs would have incentives to deploy ORide because it provides privacy and security for the riders and still preserves their business models (i.e., the SP can still charge a commission for each ride). In order to monetize ride data, the SPs can provide a discount for riders if they reveal (part of) their GPS traces. In addition, privacy and security for RHSs could be required by law and legislation, and ORide shows that it is technically possible to achieve

a strong level of protection. As such, this work lays the foundation for the design of a privacy-preserving and secure RHSs.

Additional Features. Similarly to current RHSs, ORide enables the riders to retrieve lost items (i.e., items forgotten in the car), as drivers' certificates $cert_D$ and car information are provided during the ride set-up procedure. As discussed earlier, the riders can share $cert_D$ with their friends, hence even if the riders lose their phones, they can still retrieve the $cert_D$ from their friends and contact the driver (as in current RHSs). Moreover, due to the secure channel established between the rider and the driver, the rider can still track the driver trajectory while waiting at her pick-up location, or they can contact each other (e.g., messaging or calling).

3.9 Protecting against Malicious Behaviors

In this section, we describe how the protocol presented in Section 3.7 can be extended to defend against malicious drivers and a covertly active SP.

3.9.1 Malicious Drivers: Masking

As mentioned in Section 3.7.4, if a driver behaves maliciously, she could encrypt non-zero values in the slots other than her allotted one, thus corrupting the inputs from other drivers. Our protocol can cope with this malicious behavior by adding one extra step in which the SP homomorphically multiplies each driver ciphertext by a mask $m_i = \texttt{NTT}^{-1}(X^i)$ for the driver's index i (see notations from Section 3.7.4), which preserves only the contents in the allocated slot. However, because the mask does not hold any sensitive information and it is known by the SP, a naive homomorphic multiplication with an encrypted mask would incur an unjustified overhead. Therefore, we propose, instead, a more efficient multiplication operation, denoted $\star$, as follows.

Given a ciphertext $\boldsymbol{c} = [c_0, c_1] \in R_q^2$ corresponding to a plaintext $m \in R_t$, and a mask $m_i \in R_t$, we want to obtain a ciphertext $\boldsymbol{c}' = \boldsymbol{c} \star m_i$ corresponding to the masked plaintext $\mathsf{FV.Dec}(k_s, \boldsymbol{c}) \cdot m_i$. Here, m_i can be thought of as its own noiseless and unscaled encryption (Equation (3.1) on page 43, evaluated for $u, e_1, e_2 = 0$, and no scale Δ), being a vector in R_q^2 with only one non-zero component ($[m_i, 0] \in R_q^2$). Therefore, the product results as

$$\boldsymbol{c} \star m_i = [c_0 \cdot m_i, c_1 \cdot m_i].$$

The $\star$ operation consists of two polynomial multiplications, it avoids encryption of m_i, halves the number of products with respect to an encrypted homomorphic multiplication, and keeps the cipher size from growing after the product, thus considerably improving the performance of this operation.

In any case, this precaution is only needed in case the drivers are malicious; and random checks on their locations can be implemented instead if the drivers are only covertly active (i.e., they refrain from cheating if there is a negligible chance of being caught in the act).

3.9.2 Covertly Active SP

If the SP is an active attacker, it might attempt to perform a man-in-the-middle (MITM) attack at Step 2 of the ride set-up protocol (Section 3.7.4) by replacing the public key

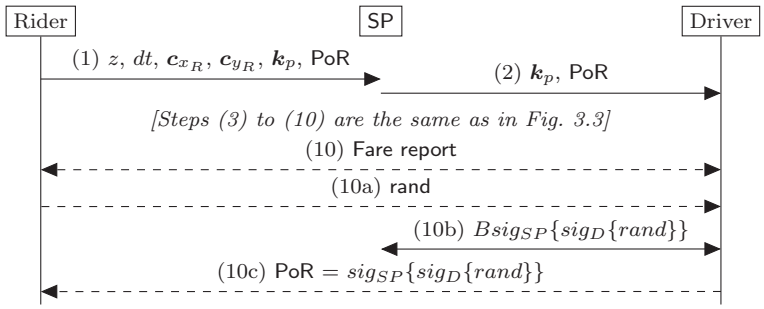


Figure 3.4: Changes introduced to the original ride set-up protocols (Fig. 3.3) to handle *covertly active* SP.

$\boldsymbol{k}_p$. However, this can be detected because the driver can share the key she received with the rider through the secure channel. If the rider detects that this key is different from the one she originally sent to the SP, then she can know that a MITM attack must have happened. The SP might also attempt to tamper with the set-up of the secure channel (Step 7a, Fig. 3.3). However, this can be detected because via the proximity channel, the rider and the driver can compare with each other the inputs that they received from the SP during the set-up protocol.

As mentioned in Section 3.7.4, any valid rider can generate an ephemeral key to make a ride request. As the SP issues credentials for riders and drivers, it can impersonate a rider or a driver in its own system. If the SP *continuously* impersonates a rider, it could learn the drivers' locations from which it could learn the coarse-grained pick-up locations of the riders. In other words, if a rider chooses the driver who is the closest to her pick-up location, the SP would know that she is in the Voronoi cell of her selected driver [156]. In what follows, we will present a mechanism for deterring this attack. However, we note that the attack is not trivial, due to the high dynamics of the system, i.e., drivers can arbitrarily go on-line and off-line anytime. Also, the SP would not have strong incentives to perform this attack, because it would add computational and bandwidth overhead to the service, thus negatively affecting the productivity of the service itself.

To deter this attack, we introduce the notion of *Proof-of-Ride* (PoR), defined and used as explained below. An illustration of the protocol with PoR is shown in Figure 3.4. A PoR is a random number $rand$ generated by the rider, signed by the driver by using the secret key associated with her $cert_D$, and then *blindly* signed by the SP by using blind-signature schemes such as [147], i.e., PoR $= Bsig_{SP}\{sig_D\{rand\}\}$. It is used to prove to the drivers that the rider is real, i.e., she took a ride in the past. When a rider makes a ride request, she has to provide in her ride request a PoR, the $cert_D$ and the random number $rand$ used in the PoR. A PoR can be used only once. For the first ride, PoR $= cert_R$.

To prevent the SP from creating its own $cert_R$ and $cert_D$ in order to create its own fake PoR, the SP has to provide a public bulletin board such as certificate transparency [157]: the SP maintains and publishes a publicly auditable and append-only log of all rider and driver certificates it has issued and revoked. Whenever a driver receives a PoR, she can check whether the rider's certificate $cert_R$ (in the case of the first ride), or the driver's certificate indicated in the PoR, is in the list of certificates published by the SP. In this way, similarly to the cases of companies opening fake user accounts [158],

	Identities	Pick-up loc.	Pick-up time	Drop-off loc.	Drop-off time	Loc. trace	Fare
Current RHSs	Rider, Driver	Precise	Precise	Precise	Precise	Full	Yes
PrivateRide	Driver	Zone	Obfuscated	Zone	Obfuscated	Partial	Yes
ORide	N/A	Zone	Obfuscated	N/A	N/A	N/A	N/A

Table 3.4: Information observed by the SP during ride set-up procedure with respect to different RHS designs. Note that the zone in ORide is larger than the zone in PrivateRide without affecting the ride-matching optimality (see Section 3.11.4). Also note that, the payment operation in ORide reveals some information about the riders, but it cannot be used to break the anonymity of the rides (see Section 3.10).

if the SP internally creates fake accounts, it can be detected by auditing authorities. Also, to prevent a rider from double-spending a PoR, and the SP from reusing a valid PoR to perform the aforementioned active attack, the SP maintains and publishes an append-only logs of PoRs that have been spent, or cancelled (due to ride cancelations).

Note that PoR could create a point of linkability, i.e., the SP is able to know that the rider is in the set of identities indicated in the fare reports deposited by a specific driver. This can be easily prevented by using anonymous-reputation and anonymous-payment systems (e.g., e-cash), as used in the PrivateRide system [17].

3.10 Privacy and Security Analysis

In this section, we present an analysis of ORide to show that it effectively defends against the privacy attacks described in Section 3.5.1. In Table 3.4, we provide a comparison between information observed by the SP during the ride set-up procedures of different RHS designs.

With ORide, the SP cannot de-anonymize a rider or driver through their anonymous logins by using their ACs. This is guaranteed due to the anonymity and unlinkability properties of the ACL anonymous credential system [148]. Additionally, the SP cannot obtain extra information from the riders' and drivers' encrypted locations and their encrypted squared-Euclidean-distances; this is due to the semantic security property of the FV encryption scheme [102]. The information observed by the SP from ORide operations can be put in two databases (DB), as follows.

- Ride DB, in which each entry contains the role and expiration date of the AC, the pick-up zone and obfuscated pick-up time. The role and expiration date are coarse-grained, i.e., all ACs issued on the same day expire at the end of the day they were issued.
- Payment DB, in which each entry contains a rider's ID, a driver's ID, a fare, and the day the fare report is deposited to the SP. Note that this database does not exist if payment is done through e-cash or regular cash, as the fare and payment are done without the SP knowledge.

(A2) Large-Scale Inference Attacks by the SP

To profile riders' and drivers' activities, the SP needs to learn the identities, the locations, and the times associated with their rides.

By using the Payment DB, the SP would know which specific rider took a ride with a specific driver, on which day, and what its fare was. As RHS drivers are often licensed

to operate in a city or state, knowing that a rider took a ride with a specific driver, the SP might be able to know the city where the rider took a ride, but it does not know the specific location in the city. Note that, in most cases, as the city could be inferred from the zones reported by the riders in their ride requests, this is not an additional leakage of information. In addition, knowing the home/work addresses and the fares of the rides, the SP might be able to infer if a rider went from home to work. However, note that even for frequent rides between the home and work of the same rider, the fares would not be the same due to different routes and traffic conditions. Therefore, the inference of rides between the home and work of a rider is error-prone. Moreover, such rides are not sensitive, compared to others, such as one-night stands, going to abortion clinics or political-party meetings. For improved anonymity, anonymous-payment methods, such as e-cash or regular cash, could be used to decouple the riders' identities from the fares, thus preventing the SP from learning about rides between the homes and places of work of the riders.

By using the Ride DB, the SP might be able to guess the identities of the riders, only if the pick-up zone has a limited number of ride activities *and* riders, e.g., a zone where only one rider lives. This case, however, is unlikely to happen in ORide, because the zones are defined in such a way that each zone has *at least* a large minimum number of ride requests per day, while balancing the bandwidth requirements for the drivers. We illustrate this in Section 3.11. Note that the SP would be detected if it lied about the activity densities in the zones, because these densities are public knowledge [159], and the drivers would notice if they received very few ride requests from a certain zone.

In the case where the SP knows that a rider makes ride requests from a specific zone (e.g., the zone that contains her home/work addresses) and it wants to know the pick-up times of these rides, the anonymity set of a ride is the number of rides that occurred on the same day from that zone. As this requires the SP to have precise knowledge about the pick-up zones, the anonymity-set size in this case is the *lower-bound* estimation of the anonymity set for the general case of large-scale profiling attacks by the SP. This lower bound is used in the evaluation of the anonymity set achieved by ORide, presented in Section 3.11.

(A3) Targeted Attacks by the SP

In the case where the SP knows the precise pick-up location and time of a ride, it still cannot know the drop-off location and time of the ride, because, in ORide, the drop-off event is not reported to the SP. Knowing the fare from the Payment DB, the SP might be able to guess whether the target went home or to work, but it could not know about other destinations. However, note that similar to the aforementioned case, the inference of rides between home and work of a rider is error-prone. Also such rides are not very sensitive. Anonymous-payment methods, such as e-cash could be used to prevent these attacks.

(A4) PII- and Location-Harvesting Attacks by Outsiders

In the current form of RHSs, a malicious outsider can easily harvest PII from drivers: it could create a rider account, fake different pick-up locations, make ride requests, obtain drivers' information, and then cancel the ride requests. This is possible, because the drivers' information is revealed to the riders right after the riders are matched with the

drivers. With ORide, the driver can easily check the proximity of the rider by using a short-range wireless communication channel before sending her PII to the rider. Thus, to collect PII from drivers, a malicious outsider not only needs to make a ride request but also to be physically close to the pick-up location reported. As a result, ORide prevents large-scale PII-harvesting attacks.

A malicious outsider might attempt to triangulate drivers, to obtain a snapshot of the locations of all drivers in a zone: It could make three fake ride requests from different locations at the same time to obtain the distances, and cancels these requests immediately. ORide mitigates this attack by applying two measures: (1) requiring a deposit token from each rider per request, thus making the attack more financially expensive and enabling the SP to identify riders who make many requests and cancelations (as discussed in Section 3.7.6), and (2) permuting the list of drivers' indices for each ride request (Step 2 in Section 3.7.4). Also, if the threat of such an attack is high, the SP can define a low threshold on the number of ACs each rider account can obtain per day.

3.11 Evaluation

In this section, we evaluate our protocols by using a real data-set of taxi rides. We first evaluate the performance of the ride-matching operation in terms of computational and bandwidth requirements for the riders and drivers. We then evaluate the effect of Euclidean distances on the optimality of ride-matching operations.

3.11.1 Data Sets

Our data-set consists of over 1.1 billion taxi rides in New York from January 2009 to June 2015 [104]. We extracted data for the month of October in 2013, one of the busiest months in the data set, which resulted in a subset of over 15 million rides. In this subset, the average duration of the rides is 13 minutes. The GPS traces of the rides are not given; however, the precise pick-up and drop-off locations and times, and the pseudo-IDs of the taxi drivers associated with the rides are provided. In addition, the data set provides mapping between latitude/longitude coordinates to NYC census tracts (CTs), neighborhood tabulation areas (NTAs) and boroughs in NYC.

We make the following assumptions. First, the drop-off location of a driver is her waiting location for new ride requests. Second, a ride-request event is a pick-up event (consisting of a pick-up location and pick-up time) in our data set. Third, for each ride-request event, the set of drivers available for that request consists of drivers who have at least one drop-off event in the last 30 minutes since the ride-request timestamp. The 30-minute interval is chosen, because the data-set shows that 99^{th} percentile of the time gap between the drop-off event of a driver and her next pick-up event is approximately 30 minutes.

3.11.2 Implementation Details

Our ORide prototype features the main cryptographic operations for the ride matching in the ride set-up procedure (Section 3.7.4). To measure the cryptographic overhead of ride-matching operations, we implemented a proof-of-concept ORide in C++, by relying on the NFLlib library [160]. In our experiments, the SP, the rider, and the driver are located on the same computer, hence network delays are not considered. However, the network

delay would not impose a considerable overhead, because a ride-matching operation requires only only one round-trip message between the rider and the SP, and one round-trip message between the SP and each driver. Also, the amount of data exchanged between the rider and the SP, and the SP and the drivers, is small (as discussed in Section 3.11.4). Note that, similarly to current RHSs, the SP can implement a timeout for responses from the drivers such that the latency is reasonable for the service. Due to the dependency requirements of the NFLlib, it is not trivial to port the implementation to mobile platforms. However, to make our experiments close to the performance of smartphones, in all of our evaluations, we *did not* use SSE or AVX optimizations for Intel processors. The source code is made available at [105].

3.11.3 Per-Ride Overhead

In this section, we describe our experimental setup and present the bandwidth and computational overhead per ride request for a rider and a driver.

We used ORide's prototype to estimate the overhead added for ride-matching operations in three settings: *(S1)* the naive SHE approach (Section 3.6.1) without using re-linearizations at the SP, *(S2)* ciphertext-packing optimizations and honest-but-curious drivers (i.e., drivers follow the protocols correctly) (Section 3.7.4), and *(S3)* ciphertext-packing optimizations and malicious drivers (Section 3.9.1).

Experimental Setup. To measure the performance of our system, we used a computer (Intel i5-4200U CPU, 2.6 GHz, 6 GB RAM) with Debian Jessie (Linux kernel 3.16). The security parameters used in our experiments are tuned to achieve an equivalent bit-security of more than 112 bits, therefore exceeding current NIST standards for 2016-2030 [161]. With this security target, and a plaintext size of 20 bits, the needed polynomial dimension is $d = 4096$, with coefficients of size 124 bits (each polynomial has a size of 62 KB). These parameters guarantee both 112-bits of security and correct operations for homomorphically adding up to 4096 encrypted locations in the same ciphertext and calculating the corresponding squared Euclidean distances. We refer the reader to Section 6 in [102] for more details on the choice of cryptographic parameters for FV. It is worth noting that we have considered pessimistic bounds in order to cope with recently published attacks that reevaluate the security of lattice-based cryptosystems [162]. Also note that, with 20-bit plaintext space, a geographical area of size 60 km^2, such as the borough of Manhattan in NYC, would be quantized into a fine-grained grid of resolution approximately 10 m $\times$ 10 m.[4]

Assuming that a rider makes a ride request to the SP and that there are 4096 drivers available for the request ($n = 4096$), with the aforementioned security parameters, the bandwidth requirements and computational overhead per ride request, for a rider and a driver, are shown in Table 3.5 and Table 3.6, and explained below.

[4]The calculation is done as follows. Assume a geographical area of size $s \times s$ and a plaintext space of b bits to represent the squared Euclidean distances between points in the area. The area can be quantized into a grid with cells of size $s/2^{(b-1)/2} \times s/2^{(b-1)/2}$, with the explanation as follows. Assuming the area is discretized into a grid of $v \times v$ cells, the largest possible squared Euclidean distance between any two points on the grid is $2 \times v^2$, and this has to be at most 2^b. Therefore, $v \leq 2^{(b-1)/2}$. In other words, each edge of size s can be discretized into v points, and the distance between any pair of two consecutive points is $s/2^{(b-1)/2}$. Therefore, the area can be represented by a grid with cells of size $s/2^{(b-1)/2} \times s/2^{(b-1)/2}$.

Setting	Rider		Driver	
Algorithm	Upload (KB)	Download (KB)	Download (KB)	Upload (KB)
S1	372	761856	124	248
S2	372	186	124	248
S3	372	186	124	248

Table 3.5: Per-ride bandwidth requirements of ORide, with $d = 4096$, $\log_2(q) = 124$, and there are 4096 drivers available for a ride request ($n = 4096$). Compared to the naive SHE approach *S1*, optimized approaches (**S2** and *S3*) significantly reduce the bandwidth requirements for the riders

Setting	Rider			Driver		SP	
Algorithm	Gen. keys (ms)	Encrypt (ms)	Decrypt (ms)	Load key (ms)	Encrypt (ms)	Load key (ms)	Compute Dist. (ms)
S1	1.51±0.06	2.6±0.2	7823.4±573.4	0.53±0.01	2.6±0.2	0.53±0.01	113868.8±6553
S2	1.51±0.06	2.6±0.2	2.2±0.1	0.53±0.01	2.6±0.2	0.53±0.01	208.9±4
S3	1.51±0.06	2.6±0.2	2.2±0.1	0.53±0.01	2.6±0.2	0.53±0.01	745.5±24.5

Table 3.6: Per-ride computational overhead of ORide (without AVX/SSE optimizations), for $d = 4096$, $\log_2(q) = 124$, and there are 4096 drivers available for a request. Statistics ($avg \pm std.dev.$) were computed from 1000 experiments. Compared to the naive SHE approach (*S1*), optimized approaches (*S2* and *S3*) significantly reduces the computation time for the SP and the decryption time for the riders.

- *Bandwidth overhead for a rider:* In all three settings, for each ride request, a rider sends to the SP a public key and two ciphertexts for her encrypted planar coordinates. This totals 6 polynomials, a payload size of 372 KB.

  Regarding the number of squared-distance ciphertexts a rider receives from the SP, in setting *S1*, it is equal to n, i.e., the number of responding drivers. In settings *S2* and *S3*, it is significantly reduced to $\lceil n/d \rceil$, due to ciphertext packing. A ciphertext of the squared Euclidean distance, when avoiding relinearizations (see Section 3.7.4), consists of 3 polynomials, thus having a total size of 186 KB. Assuming 4096 drivers respond to a ride request, setting *S1* would require the SP to send 4096 ciphertexts (744 MB) to the rider, whereas *S2* would require only one ciphertext (186 KB).

- *Bandwidth overhead for a driver:* In all three settings, for each request: (1) on the downlink, the SP forwards to each driver a public key, 2 polynomials of size 124 KB, and (2) on the uplink, each driver sends back to the SP her encrypted planar coordinates, totaling 4 polynomials of size 248 KB.

- *Computational overhead:* As shown in Table 3.6, for both riders and drivers, in all three settings, the computational overhead introduced by key generation and encryption operations are small, i.e., 1.5 ms and 2.6 ms, respectively. Due to masking, setting *S3* introduces a small computational overhead for the SP in homomorphic squared Euclidean-distance computation, compared to setting *S2* (745 ms vs. 208.9 ms). However, noticeably, due to ciphertext packing, settings *S2* and *S3* significantly reduce the computational cost for the SP (208.9 and 745 ms compared to 113.9 s required by *S1*). It also significantly reduces the decryption overhead for the rider, from 7823 ms in setting *S1* to 2.2 ms in settings *S2* and *S3*.

  Note that the results for the rider and driver are optimistic, as we used a lap-

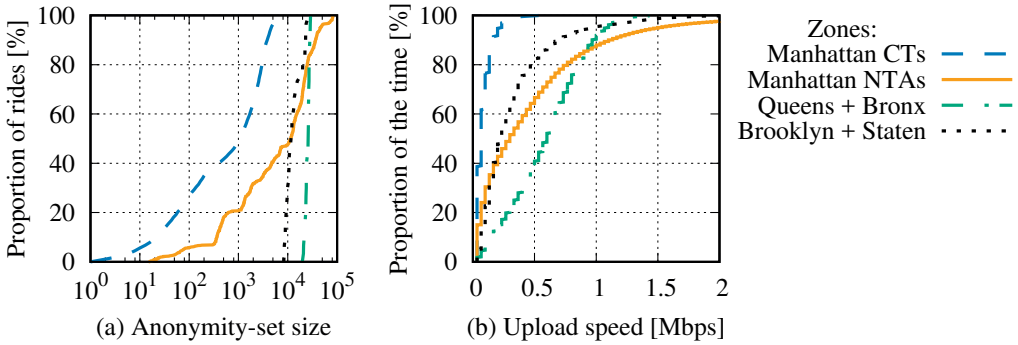


Figure 3.5: System performance. (a) Anonymity-set size, (b) Upload speed requirement for the drivers. Our results show that ORide is scalable while providing good anonymity-sets for riders.

top instead of a smartphone (however, as stated before, CPU optimizations were disabled to reduce the difference). Although such comparisons are not straightforward, we can make a rough estimation of the expected performance of ORide in smartphones. For instance, by comparing the performance scores of top multicore CPUs in smartphones [163] with top multicore CPUs in desktops [164], we can see that the difference is less than an order of magnitude. Assuming such a difference, then we can see that the computational overheads for key generation, encryption, and decryption are still acceptable in smartphones, around 15 ms, 26 ms, and 22 ms, respectively. The overhead is still acceptable even if we consider two orders of magnitude difference, as the total time to hail a ride is in the order of minutes [165].

3.11.4 Riders' Anonymity and Drivers' Bandwidth Requirements

In this section, we present the trade-off between the ride anonymity set vs. bandwidth requirements for the riders and drivers, by using the real data-set presented in Section 3.11.1. Due to the high demand of taxi rides in Manhattan with respect to the lower level of activity in other boroughs in NYC (from our data-set, Manhattan accounts for 90% of ride requests), we define two zone settings as follows.

- Setting one (Z1): Manhattan is divided into census tracts (CTs). Each CT is one zone. The boroughs of Queens and Bronx are merged into one zone, and the boroughs of Brooklyn and Staten Island are merged into one zone.

- Setting two (Z2): Manhattan is divided into neighborhood tabulation areas (NTAs). Each NTA is one zone. Similarly to setting one, the boroughs of Queens and Bronx are merged into one zone, and the boroughs of Brooklyn and Staten Island are merged into one zone.

Estimation of the Anonymity Set. As explained in Section 3.10, the number of rides in a day from a zone is a *lower-bound* estimation of the anonymity set for a ride. Fig. 3.5a shows the experimental cumulative distribution function (CDF) of the lower-bound anonymity-set size. It can be observed that, for Manhattan with the zone granularity of census tracts, 81.7% of the rides have an anonymity set of size *at least* 50, and

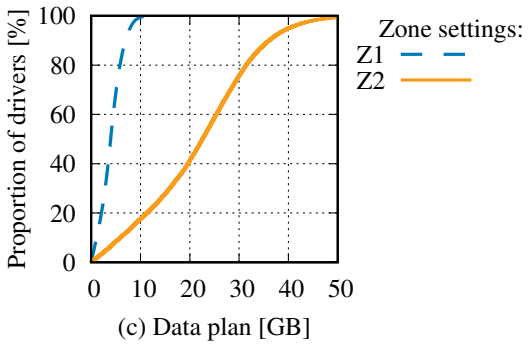


Figure 3.6: Monthly data-plan requirement for the drivers.

for a zone consisting of Queens and Bronx, all of the rides have an anonymity-set size of *at least* 26,000.

Bandwidth Requirements for Riders. The bandwidth requirements for a rider, per ride request, depends on the number of available drivers. Our experiments show that for both zone settings, for all ride requests, the number of available drivers is less than 3,500. This means that with the security parameter chosen (as presented in Section 3.11.3) and when proposed optimized packing approaches are used, a rider needs to download only one ciphertext, i.e., 186 KB, which is negligible.

Bandwidth Requirements for Drivers. Fig. 3.5b shows the CDF of the upload speed required for the drivers; the upload speed is computed by multiplying the number of requests a driver receives per second with the size of the ciphertexts she has to upload per request. Note that the required downlink speed is half of the uplink speed, because the downlink payload is half the size of the uplink payload (Section 3.11.3). It shows that for Manhattan with the zone granularity of census tracts, the required upload speed is less than 0.5 Mbps; and for other zones, the required upload speed is less than 2 Mbps, which is provided by 3G or 3.5G networks.

Monthly Data-Plan Required for Drivers. Fig. 3.6 shows the CDF of a data plan required for the drivers for two aforementioned zone settings; this is calculated by multiplying the total number of requests a driver would receive during her waiting time with the uplink and downlink payloads per request. The result shows that with the zone setting Z1, a driver needs *at most* 10 GB of data per month, and with the zone setting Z2, 60% of drivers need less than 25 GB of data per month. This requirement is reasonable: For example, in the U. S. , an unlimited data plan typically offers 20-26 GB of high-speed data for less than $100 [166]. In addition, the drivers can reduce their data-plan consumption by using free WiFi networks, such as LinkNYC [167].

The requirements on bandwidth for the drivers and the anonymity-set sizes for riders enables the SP to define the zones that balance the trade-off between the two aforementioned requirements. For example, for areas that have a high density of ride activities such as Manhattan, the SP could discretize the borough into zones of CTs or NTAs, or combinations of CTs and NTAs. Note that, as shown earlier, at the granularity level of CTs (Z1), the anonymity set provided by ORide for the case of a very strong adversarial SP is already large. In special cases, such as concerts and sport events, the SP can split a crowded zone into sub-zones, in order to find a balance for the aforementioned trade-off. For areas that have fewer ride activities, such as other boroughs in NYC or other cities,

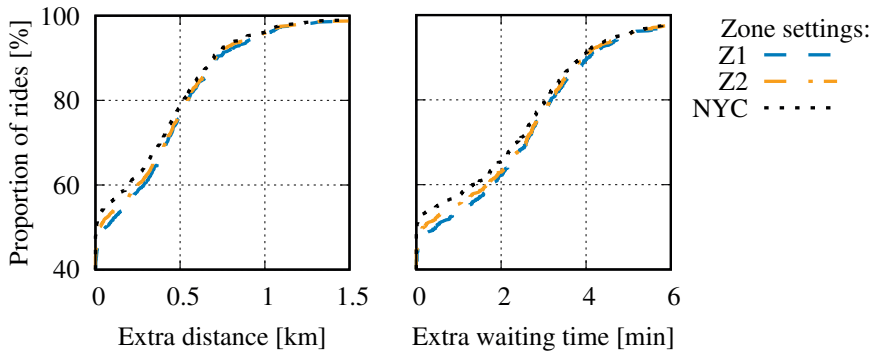


Figure 3.7: Effect of Euclidean distance on the extra distances for the drivers (left) and on the waiting time for the riders (right) with respect to different zone settings. Our results show that the overhead added by ORide is reasonable.

an entire borough or city can be a zone. For example, a zone consisting of the boroughs of Queens and Bronx would guarantee an anonymity-set size of at least 26,000 for a ride, while requiring the drivers to have an Internet connection of only at most 2 Mbps. It is worth noting that the zones used in our experiments were chosen based on the existing zones predefined in our NYC dataset; however, the SPs can adjust the zones such that each individual rider receives approximately the same level of privacy guarantee (all the zones have approximately the same anonymity-set size). This can be calculated, for example, based on the knowledge of the SPs about the taxi demand in the past and the future events (e.g., concerts or sport events) in an area.

3.11.5 Effect on Ride Matching

Ideally, to minimize the extra costs for both the drivers (e.g., gas and driving time to pickup) and the riders (e.g., waiting times at pick-up locations), the ride-matching algorithm should take into account the road networks and real-time traffic conditions. Due to the limited operations supported by SHE, ORide uses a simpler matching metric, i.e., Euclidean distances between the riders' and drivers' projected locations. Due to the bandwidth constraints, the ORide ride-matching algorithm matches a rider to drivers in the same zone, hence suboptimality, e.g., if a rider is close to the border of a zone, the closest driver might be in one of the neighboring zones.

Fig. 3.7 shows the CDFs of the extra costs due to the suboptimality of ORide, compared to the ideal solution, with respect to three different zone settings: Z1, Z2, and the entire city of New York (NYC). The experiment was done on a set of 1,000 randomly selected ride requests. For the ideal matching, we used the Google Maps Distance Matrix APIs [168] to compute the times and distances between a pick-up request and the available drivers (see assumptions in Section 3.11.1). To reduce the number of requests made to the Google APIs[5], from the set of available drivers, we selected 100 drivers who were closest to the pick-up location as the potential candidates for the ideal matching.

It can be observed that the median extra costs are small: when Z1 is used, in more than 45% of the cases, the driver selected by ORide and the ideal solution is the same, and, in nearly 80% of the cases, the extra driving distance is less than 0.5 km. In addition, the

[5]The number of requests per day is limited.

size of the zone has only negligible effects on the optimality of the matching algorithm: If the set of all the drivers available in NYC was used for the ORide matching algorithm, compared to the ideal solution, 78.7% of the cases would have an extra distance of less than 0.5 km, compared to 76.2 % and 76.8 % of the cases when Z1 and Z2 were used, respectively.

3.12 Conclusion

In this chapter, we have proposed ORide, a practical solution that efficiently matches riders and drivers in a privacy-preserving way while still offering key RHS features such as easy payment, reputation scores, accountability, and retrieval of lost items. ORide enables the SP to choose a balanced trade-off between anonymity sets for riders vs. bandwidth requirements for the drivers. For example, for a lower-bound anonymity-set size of 26,000 for rides from the boroughs of Queens and Bronx, drivers only need to have an Internet connection of at most 2 Mbps. The trade-off enables the SP to define the zones such that all users in the system are guaranteed large anonymity sets, even if they are in sparsely populated residential areas with sparse ride activities (by expanding the zones). We have also shown that, even in the extreme case of targeted attacks, i.e., a curious SP wants to know the destination of a rider given the time and location of a rider's pick-up event, the location privacy of the rider's destination is still guaranteed.

Chapter 4

Cheat-Proof and Private Summaries for Location-Based Activities

In Chapter 3, we presented a privacy-preserving system for ride-hailing services. In these services, in order to provide privacy for users, we focused on how to provide anonymous ride matching between riders and drivers, and on how to still be able to provide accountability and key ride-hailing operations (secure payment and reputation rating). In this chapter, we will present a privacy-preserving design for the context of activity-tracking services. Such services enable people to record and upload information about their location-based activities (e.g., the routes of their activities). They also enable users to share information and compete with their friends on activity-based social networks and, in some cases, to obtain discounts on their health insurance premiums by proving they conduct regular fitness activities. Our goal is to design a system that enables users to prove to service providers that they have truly achieved certain activity summaries (e.g., they have run 10 km), while preserving their location privacy (i.e., they do not have to reveal their location traces to the service providers).

4.1 Introduction

More and more people rely on activity-tracking services and quantified-self devices to monitor, manage and to encourage themselves to do physical activities. Mobile apps, such as Endomondo, Garmin Connect, RunKeeper, Runtastic and Strava, and wearable devices, such as Fitbit, Nike+ Fuelband and Jawbone UP, enable users to keep track of their performance while running, hiking or cycling. This information is collected using location-based services (LBSs) and embedded sensors in smartphones and wearable devices. Due to the popularity of these apps, top mobile operating systems now include APIs that facilitate the gathering and sharing of fitness and health data across multiple apps and devices (e.g., HealthKit for iOS and Google Fit for Android). A key feature of these services is to enable users to access summaries of their activities and performance statistics and to share this information with other users and service providers on online social networks. For instance, users can share the total distance covered, the cumulative elevation gain and the path taken during their activities. For this purpose, activity-

tracking apps collect and send users' location and fitness data, possibly while they pursue their activities, to service providers.

In exchange for their data, users are offered various incentives. For example, users can receive discounts, coupons or even cash [169, 170, 171, 172, 173], awards at competitions [174, 175] or simply points to improve their social reputation. In addition, many companies, including big names such as British Petroleum (BP), Bank of America and Autodesk, are giving activity-tracking devices to their employees to encourage healthier lifestyles and, as a result, improve productivity and lower corporate insurance costs [176, 177, 178]. Similarly, health insurance companies such as United Health, Kaiser Foundation Group, Humana Group, Aetna and Chrétienne Sociale Suisse (CSS) have created programs to include activity-tracking devices into their policies, i.e., consumers are rewarded by the insurers with lower rates based on their activity summaries [177, 179, 173, 180]. Also, activity-tracking service providers, such as Fitbit, have partnered with organizations to enable them to continuously monitor their employees' health and wellness conditions [181].

Although activity-tracking and sharing services are gaining popularity, there are two important issues that could hinder their wide-scale adoption and viability. First, the privacy concerns associated with the data collected by these apps. In particular, users' location data, which is revealed to service providers, can be used to infer private information about them, such as their home/work locations [182, 183], activity preferences [184], interests [185], social networks [186], or even locations of classified military bases [14]. Moreover, the current privacy protection mechanisms provided by activity-tracking apps, such as the Endpoint Privacy Zones (EPZs) supported by Strava, are shown to be insufficient to protect the users' privacy [187]. Second, the concerns about the integrity of the data reported by users. As the value of incentives increases, users could be tempted to cheat when reporting their performance [188], which would endanger the viability of the system for the service provider and its affiliates, as well as its attractiveness to the users. For example, location cheating can be achieved by making mobile devices report erroneous location information [189, 4], or by spoofing the GPS/Wi-Fi signals used for geo-location users [190, 191, 192]. Moreover, some tools enable users to manipulate activity data to lie about their performance [193, 194, 195].

To assess the awareness and concerns of users of activity-tracking services regarding opportunities to cheat and privacy issues, we conducted a user survey of 50 participants. Our survey participants are active RunKeeper users who we recruited on the Amazon Mechanical Turk platform. In the survey questionnaire, we first informed the participants about existing opportunities to cheat and the privacy issues of fitness-tracking services such as RunKeeper; we then polled them about their awareness and their concerns (see Section 4.6.1 and Appendix B, including the full transcript of the questionnaire). Regarding opportunities to cheat, we found that all the participants were unaware of them and 48% declared themselves very or extremely concerned about the cheating possibility. We also found that 90% of the participants were unaware of privacy issues and 82% declared themselves very or extremely concerned about privacy. These results highlight the need to raise awareness and the need for technical solutions to build cheat-proof and private activity-tracking services.

A straightforward solution to these issues consists in enforcing the use of either secure and/or privacy-preserving location proofs for users [190, 196, 197, 198], where their location could be either (1) trusted and known (as it is the case for activity trackers) or (2)

untrusted and known (but useless for obtaining rewards). In fact, solutions guaranteeing (1) would benefit the service provider by ensuring that cheating is infeasible, whereas solutions satisfying (2) would protect users' location privacy but would provide locations that are too coarse-grained for computing meaningful summaries.

In this chapter, we propose SecureRun, a novel infrastructure-based approach that provides guarantees both in terms of the prevention of cheating and location privacy for the users *vis-à-vis* the service provider, while allowing the service provider to compute accurate summaries and statistics of users' activities, such as the total distance covered during an activity. SecureRun relies on existing wireless access-point (AP) networks and alleviates the need for a costly deployment of a dedicated ad-hoc infrastructure. Instead, it could rely on strategic partnerships between social network providers and access-point network operators. SecureRun consists of two phases: First, users obtain secure and privacy-preserving proofs of performance during their activities, by relying on a lightweight message exchange protocol between their mobile device and the Wi-Fi access points encountered while pursuing the activity. Second, the service provider computes an accurate summary of a user's activity, such as the total distance covered between two time instants or the elevation gain, without learning any additional information about the user's actual location. SecureRun produces, in a privacy-preserving way, a secure and accurate lower bound of the actual distance covered by a user while she performs an activity. Finally, it is able to take advantage of the co-existence of multiple access-point operators to improve the accuracy/privacy trade-off. Unlike the initial version of our system presented in [20], in order to maximize the accuracy of the activity summaries produced, SecureRun computes the optimal set of access points to communicate with. To the best of our knowledge, this is the first work to address privacy and cheating issues in the computation of activity summaries.

We evaluate our solution on a large data-set of real user-activities, collected from the Garmin connect [199] social network in the regions of Brussels (Belgium), London (UK) and Paris (France). For these regions, we also extract the actual locations of a network of deployed Wi-Fi APs operated by FON [200]. Moreover, to evaluate the benefits of having multiple operators in a given area, we extract the locations of a second access-point network in the urban area of Paris. The experimental results show that our solution can gracefully balance accuracy and privacy. SecureRun achieves good accuracy, up to a median accuracy of more than 80%. This constitutes a significant improvement compared to the performance of the initial version of the system reported in [20]: On average, we achieved an absolute improvement of 5-10% in the median accuracy. In our survey, we also asked the participants about their level of satisfaction regarding activity summaries for different values of the summary accuracy (i.e., "Assuming that you have run 10 miles, what would your level of satisfaction be if SecureRun issues a certificate of 6 mi? 7 mi? 8 mi? 9 mi?): We found that for an accuracy of 80% (i.e., SecureRun issues a certificate of 8 mi when the user runs for 10 mi), which roughly corresponds to the median performance of SecureRun, the "high" and "very high" levels of satisfaction account for 42% of the participants (74% with the "medium" level of satisfaction). We also conduct a sensitivity analysis to evaluate the effect of the distribution of the APs on the performance of SecureRun.

4.2 Related Work

Cheating on activity-based social networks is becoming a serious problem. For example, He et al. [190] show that users can easily override Foursquare's GPS verification mechanisms by modifying the values returned by the calls to the geo-location API of smartphones. Similarly, Polakis et al. [201] use a black-box approach to uncover the mechanisms used by Foursquare and Facebook Places to detect location attacks and propose several ways to circumvent them. Moreover, work from Carbunar and Potharaju [188] analyze data from Foursquare and Gowalla and find that incentives to cheat exist because people actively check-in and collect rewards. Thus, it is necessary to carefully balance incentives with a more effective verification of users' location claims. In this regard, Zhang et al. [191] show that fake check-ins lead not only to monetary losses for the venues that offer special deals on location-based check-ins but also to the degradation of the quality of service provided by recommendation systems that rely on users' location information. Carbunar et al. [198] also show that there is tension between privacy and correctness in location-based services, where users are unable to prove that they have satisfied badge conditions, without revealing the time and location of their check-ins.

Meanwhile, to defend against cheating, researchers have also proposed several mechanisms that offer secure verification of location information. From a broad perspective, such mechanisms can be grouped in three categories: infrastructure-independent, infrastructure-dependent and hybrid mechanisms. In the *infrastructure-independent* approach, a user obtains location evidence from her neighbors by using short-range communication technologies, such as Bluetooth [202, 203, 204]. Specifically, Talasila et al. [202] propose a location authentication protocol where a set of users help verify each others' location claims. The protocol operates by keeping a centralized authority that, based on users' spatio-temporal correlations, decides whether such claims are authentic or not. Similarly, Zhu et al. [203] propose a system where mutually co-located users rely on Bluetooth communications to generate their location claims that are then sent to a centralized location verifier. In addition to the security and privacy guarantees presented in [202], Zhu et al. [203] enable individual users to evaluate their own location privacy and decide whether to accept location proof requests by other users. Jadliwala et al. [205] provide a formal analysis of the conditions needed in an ad-hoc network to enable any distance-based localization protocols in wireless networks. Similar approaches were explored in mobile sensor networks [206, 207].

More in line with our work, the *infrastructure-dependent* studies assume the presence of a centrally-operated set of access points (AP) that produce and verify location claims. For instance, to ensure the presence of a user in a given region, the AP can require her to execute together a nonce-based challenge-response protocol, with constraints on the maximum round-trip delay of the messages exchanged between the user and the AP [208], or any distance bounding protocol [209, 210, 211, 212], which enables the AP to check the minimum distance between itself and the user. In particular, Capkun and Hubaux [211] propose a verifiable multilateration protocol that can be used to securely position nodes in a wireless network. Once the secure localization phase is done, the user can obtain a location proof to certify that the user is at a specific geographical location [208]. Alternatively, Luo and Hengartner [197] show that a user can obtain location proofs with different precision levels and then select one to disclose to the service provider, depending on her privacy preferences.

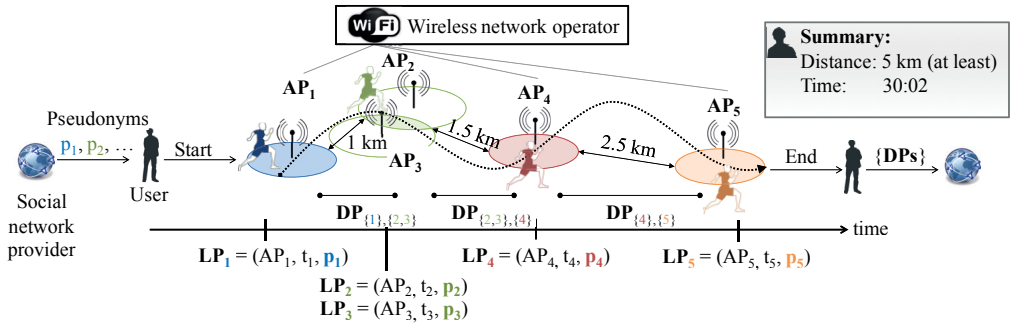


Figure 4.1: Overview of SecureRun's system architecture. A user first obtains a set of pseudonyms $\{p_1, \ldots\}$ from the social network provider. Then, while performing an activity along the dotted trajectory, she sporadically requests location proofs (LP) at times t_i, using pseudonyms p_i, to the APs encountered along the trajectory. By using the LPs, the APs compute, and deliver to the user, distance proofs for the different time intervals. The user finally sends the distance proofs to the social network provider that combines them and publishes the summary on her profile.

Hybrid approaches rely on both landmarks (e.g., WiFi, cellular base stations) and short-range communications between users (e.g., Bluetooth) to obtain location evidences. For instance, Uchiyama et al. [213] describe an opportunistic localization algorithm for positioning mobile users in urban areas. The area of presence of a user is calculated based on a map of obstacles (i.e., the area where there is no radio signal), the last presence areas of the node itself and the nodes it encountered. Similarly, Koo et al. [214] present a hybrid system that relies on landmarks located at corners or intersections of streets and short-range communication between users. The system considers the users' routes (i.e., a sequence of segments connecting successive landmarks), the average moving speed in the segment and the collaboration between mobile nodes to locate the users.

Along the line of protecting users' privacy in activity-tracking services, Hassan et al. [187] show that the privacy measures provided by popularly used systems, e.g., Strava, are insufficient. SecureRun relies on an infrastructure of wireless access points to provide secure distance proofs, in line with the infrastructure-dependent models discussed above. However, it is the first work, to the best of our knowledge, that provides secure distance proofs and that addresses the challenge of provable activity summaries while protecting the users' privacy.

4.3 System Architecture

In this section, we describe the different entities involved in our system: a user, one or more Wi-Fi network operator and a service provider (e.g., a social network). Figure 4.1 depicts the system considered and a sketch of SecureRun. We also describe the adversarial model in this scenario. Moreover, we discuss and analyze the incentives of the various involved entities and the adoption of SecureRun in Section 4.6. For the sake of readability, notations used throughout the chapter are summarized in Table 4.1.

Notation	Description
AP	Access Point
LP	Location Proof
DP/EP	Distance/Elevation Proof
R	Communication range of APs
ΔT	Duration of silence periods
$\mathrm{GK}_{\mathrm{pub}}$	The public group key. It is known by everyone.
$\mathrm{GK}_{\mathrm{priv}}$	The private group key. It is known only by (all) the APs.
$\mathrm{LP}_{i,j}$	The j-th LP collected at sampling time t_i. It has format $\{P_i, t_{i,j}, (x_{i,j}, y_{i,j})\}$.
P_i	The pseudonym which is used at sampling time t_i.
$t_{i,j}$	The time at which the j-th LP is collected at sampling time t_i.
$(x_{i,j}, y_{i,j})$	The coordinate of the AP from which the j-th LP is collected at sampling time t_i.
C_i	The set of APs from which the LPs are collected at sampling time t_i.
A_i	The time-aligned intersection of the communication disks of the APs from which the LPs are collected at sampling time t_i.

Table 4.1: Summary of notations.

4.3.1 Users

We assume that some users pursue location-based activities, where they move in a given geographical region, and that they want to obtain statistics or summaries of their activities. These users are equipped with GPS- and WiFi-enabled devices and have sporadic Internet connectivity (at least at some point in time before and after the activity). Thus, they can locate themselves and communicate with nearby Wi-Fi access-points. We assume a unit-disk model for Wi-Fi communications, in which a user and an AP can communicate only if the distance between them is less than a given radius R, which is constant across all users and all APs. Note that we do not assume that users can communicate with the APs as soon as the distance is less than R (we only assume that *if* they can communicate with an AP, *then* the distance between the user's device and the AP is less than R). As such, this is a relaxed version of the unit-disk model. Note also that this assumption can always be enforced by choosing a high value of R. In particular, we assume that users cannot violate this model by, for example, increasing the transmission power of their devices. We assume that users can obtain random identifiers (or pseudonyms) from the online service provider, and that they can use such pseudonyms to protect their privacy while pursuing their activities. We assume that users do not hand their pseudonyms to other users (this can be enforced by embedding sensitive or critical information about the users in their pseudonyms, such as tokens that enable the users to reset their passwords). Finally, we assume direct Wi-Fi connections to have much smaller communication delays than cellular Internet connections, thus enabling us to prevent proxy/relay attacks [190] by using delay-based challenge-response mechanisms.

In order to brag or obtain rewards, users might be tempted to unduly increase their performance by deviating from the protocol. To do so, such users could, for instance, report locations that are different from their actual locations, forge messages, or reuse messages they, or their friends, obtained in the past.

4.3.2 Wi-Fi AP Network Operator

We assume the existence of one or multiple Wi-Fi network operators, and that each operator controls a set of fixed Wi-Fi APs deployed in the regions where the users pursue their activities. ISP-controlled Wi-Fi community networks such as British Telecom Wi-Fi, LinkNYC (or a federation of such networks, such as FON) constitute typical candidates for deploying and running the APs used by SecureRun, because they own and control the routers that they provide to their subscribers (e.g., the ISPs can transparently remotely update the firmware of their subscribers' routers). Each AP is aware of its geographic position and of its communication radius. We assume that all the APs have synchronized clocks (by using e.g., the Network Time Protocol (NTP) [215]), and that they are able to compute public-key cryptographic operations. In particular, we assume that all the APs from a same network operator share a public/private group key pair $(\mathrm{GK}_{\mathrm{pub}}, \mathrm{GK}_{\mathrm{priv}})$, where $\mathrm{GK}_{\mathrm{pub}}$ is known by the users and the service provider, and $\mathrm{GK}_{\mathrm{priv}}$ is only known to the network operator and to its APs. Finally, we assume that the APs cannot uniquely identify mobile devices based on the characteristics of their wireless-network adapters (i.e., wireless fingerprinting techniques based on clock skews or physical layer signatures [216, 217, 218]).

Some access-point operators might be interested in tracking the users' locations, based on the information obtained by all of their APs.[1] We assume them to be *semi-honest* or *honest-but-curious*, meaning that they do not deviate from the protocol specified in our solution, but that they analyze the information they collect while executing the protocol.

4.3.3 Social Network Provider

We assume that there is a social-network provider that offers activity summaries and sharing services to its registered users. The provider is able to generate sets of pseudonyms for its users, by using a suitable public-key encryption scheme. Moreover, it is able to verify the authenticity of messages signed with the network operators' group keys (by using their public group keys). Like the network operators, the social-network provider might be interested in the users' locations[1] and is assumed to be *honest-but-curious*. Finally, we assume that the different entities do not collude with each other.

4.4 Our System: SecureRun

We build an activity-tracking system that (1) guarantees the authenticity of the user's activity data with respect to cheating users who try to unduly increase their performance and (2) protects the users' location-privacy with respect to curious network operators and service providers that try to track them.

In this section, we present SecureRun, our solution for secure and privacy-preserving activity summaries. First, we give a high-level overview of SecureRun and define the main operations it involves. Then, we provide a detailed description of each of the aforementioned operations. Figure 4.1 shows an overview of SecureRun and of its operations.

[1] The information available to the adversaries is made explicit in the description of the protocol (Section 4.4) and summarized in the security analysis (Section 4.7).

4.4.1 Overview

From a high-level perspective, SecureRun operates as follows. As a user pursues her location-based activity, she moves and communicates (through her smartphone) with the wireless access points located along her route (and in her communication range) to obtain *location proofs* (LP). A location proof is a digitally signed message, delivered by an access point, that certifies that the user is, at a given time t, in a given range of an access point that is located at a given position (x, y).[2] The times/positions at which users request such location proofs are determined by a *sampling algorithm*.

A user employs different pseudonyms (provided to her beforehand by the service provider) when communicating with the access points. The different location proofs obtained by a user (from different access points) in a short interval of time are *aligned in time* and *combined* into a more precise location proof by using intersection techniques. To obtain an *activity proof*, a user provides pairs of consecutive precise location proofs to an access point; more specifically, she obtains a *distance proof* (DP) and/or an *elevation proof* (EP). The activity proofs that the user obtains do not provide any location information, as they do not include information about where the activity was pursued but only about the distance or elevation. Such proofs are digitally signed messages that certify that a user achieved (at least) one given performance during a given time span, e.g., that she ran at least 1 km between 3:02pm and 3:08pm on March 19th. Finally, a user sends all the activity proofs she collects, while pursuing her activity, to the service provider that performs the adequate verifications; if the information is correct, the provider combines the proofs into an activity summary that it publishes on the user's profile.

In terms of privacy, the use of pseudonyms protects users' locations with respect to the access-point operators; the lack of location information in activity proofs provides protection with respect to the social-network provider. Finally, the use of digital signatures and pseudonyms, combined with the fact that the activity proofs represent the lower bounds of the user's actual performance, provide security properties with respect to dishonest users.

4.4.2 Location Proofs

At each sampling time t_i (determined by the sampling algorithm described below), a user begins to collect location proofs from the access points in her communication range. To do so, she periodically broadcasts (during a short time interval starting at time t_i) location-proof requests that contain one of her pseudonyms P. Note that a different pseudonym is used for each sampling time. All the access points in her communication range send back messages that contain the pseudonym P, a timestamp t (i.e., the time at which the request is processed by the access point) and their coordinates (x, y), digitally signed with the private group key $\mathrm{GK}_{\mathrm{priv}}$, specifically a location proof $\mathrm{LP} = \mathrm{sig}_{\mathrm{GK}_{\mathrm{priv}}}\{P, t, (x, y)\}$. We denote by $\mathrm{LP}_{i,j} = (P_i, t_{i,j}, (x_{i,j}, y_{i,j}))$ the j-th location proof collected at sampling time t_i (note that we omit the signature for the sake of readability). As the communication and processing delays differ from one access point to another, the location proofs collected from different access points at a same sampling time have different timestamps. Under

[2]Throughout the chapter, we use an equi-rectangular projection to map the latitude and longitude of the considered locations to a Cartesian coordinate system, in which the Euclidean distance is a good approximation of the Haversine distance.

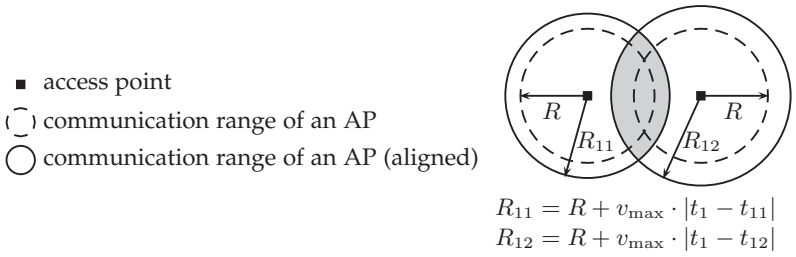


Figure 4.2: Time alignment of location proofs LP_{11} and LP_{12} at time t_1. At t_1, the user was in the gray area defined by the intersection of the two solid disks.

the relaxed unit-disk communication model (with radius R), such a location proof certifies that, at time t, the user is at a distance of at most R to the access point that issues the location proof. In other words, it certifies that the user is in a disk of radius R, centered at the point of coordinate (x, y). We denote such a disk by $\mathcal{C}((x, y), R)$. In addition, we denote by C the entire set of APs located along the user's route and by C_i the set of APs defined by the location proofs collected by the user at time t_i. The sensitivity of our solution with respect to the density and distribution of the access points is presented in Section 4.5.3.

4.4.3 Activity Proofs

To obtain an activity proof (i.e., a distance proof or an elevation proof), a user sends to any access point (whenever she needs it) the location proofs collected at two consecutive sampling times t_i and t_{i+1}. The contacted AP first combines the location proofs (those collected at each of the two sampling times) into more precise location proofs, by aligning them in time and intersecting them. As these location proofs have different timestamps, the first step of the combination consists in aligning the different location proofs as follows. Assuming the speed at which users move is upper-bounded by a constant $v_{\max}$ defined by the system, the fact that a user is at a distance at most d to an access point at time t means that, at time t', the user is at a distance of at most $d + v_{\max} \cdot |t - t'|$ to this access point. The second step of the combination simply consists in computing the intersection of the aligned location proofs. Note that only the locations proofs with a timestamp in $[t_i, t_i + \delta t]$ are combined. The access point determines a geographical area A_i where the user was at time t_i from the following expression.

$$A_i = \bigcap_{j \in \mathcal{C}_i} \mathcal{C}((x_{i,j}, y_{i,j}), R + v_{\max} \cdot |t_i - t_{i,j}|) \tag{4.1}$$

The AP repeats the same operation for the location proofs obtained at sample time $i + 1$. Figure 4.2 shows an example of a location-proof combination: Assuming that at sampling time t_1, a user obtains two LPs: $\mathrm{LP}_{11} = (P_1, t_{11}, (x_{11}, y_{11}))$ and $\mathrm{LP}_{12} = (P_1, t_{12}, (x_{12}, y_{12}))$ from two APs.

The activity proofs are computed from a lower bound of a user's performance. As for distance proofs, knowing that a user was in an area A_i at time t_i and in an area A_{i+1} at time t_{i+1}, the distance d_i between A_i and A_{i+1} (i.e., the minimum of the distances between any point in A_i and any point in A_{i+1}) constitutes a lower bound of the distance covered by a user during the time interval $[t_i, t_{i+1}]$. More specifically, using the Euclidean

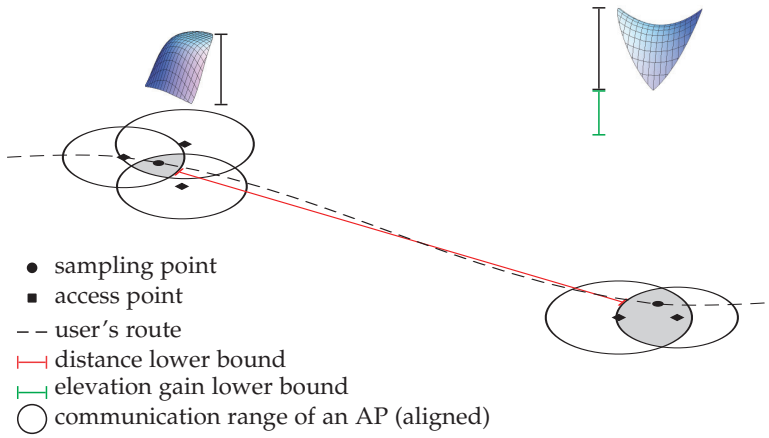


Figure 4.3: Computation of distance and elevation proofs. The shaded areas correspond to the intersections of the location proofs obtained at the same sampling time. The 3D plots correspond to the elevation profiles of the shaded areas, based on which the lower-bound of the elevation gains are computed.

distance, we have

$$d_i = \min_{\substack{(x,\,y)\,\in\,A_i \\ (x',\,y')\,\in\,A_{i+1}}} \sqrt{(x-x')^2+(y-y')^2} \tag{4.2}$$

A tight approximation of d_i can be obtained by using a non-linear optimization toolbox such as IPOPT [219].

With respect to the elevation proofs, the following expression gives a lower bound of the cumulative elevation gain[3] achieved by a user during the time interval $[t_i, t_{i+1}]$.

$$e_i = \min_{\substack{(x,\,y)\,\in\,A_i \\ (x',\,y')\,\in\,A_{i+1}}} \left(\max\left(0, z(x',y') - z(x,y)\right)\right) \tag{4.3}$$

where $z(\,\cdot\,,\,\cdot\,)$ denotes the elevation of the point of coordinate (x, y). Note that the "max" operator is used here in order to account for only *positive* elevation gains. Unlike for the lower bound of the covered distance, we compute the lower bound of the elevation gain analytically: $e_i = \max\left(0, \min_{(x,y)\in A_{i+1}} z(x,y) - \max_{(x,y)\in A_i} z(x,y)\right)$. Figure 4.3 illustrates the stages of the generation of activity proofs in the case of the covered distance/elevation gain. Finally, the access point generates an activity proof $\mathrm{sig}_{\mathrm{GK}_{\mathrm{priv}}}\{d_i, e_i, [t_i, t_{i+1}], \{P_i, P_{i+1}\}\}$ and sends it back.

Note that, in Equation 4.1, we consider only the set C_i and do not take into account the fact that the user was *not* in the regions defined by the access points in the set $C \backslash C_i$ (hereafter, we call this *negative information*). Intuitively, if negative information is considered, the region that the user is inside at time t_i could be redefined as the intersection of A_i (as defined in Equation 4.1) and the complements of the regions defined by the APs in $C \backslash C_i$. Therefore, the negative information would provide a tighter estimate of the area the user was in at time t_i. This would provide better accuracy for the system, as the refined region is included in A_i, but it would also enable the user to cheat by selectively reporting only a subset of the collected LPs (i.e., omitting some of the collected LPs to unduly increase the resulting distance proofs). For this reason, in our evaluation of

[3]Note that the elevation loss can be computed by following the same line of reasoning.

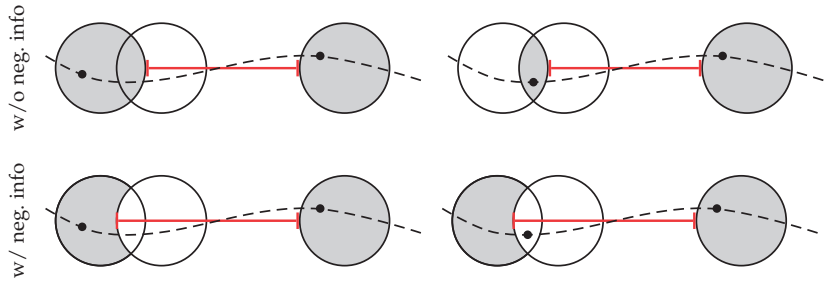


Figure 4.4: The lower-bound distance with and without considering negative information. The figures on the left show the case that using negative information improves the tightness of the lower-bound distance. The figures on the right show that using negative information can enable the users to unduly increase their lower-bound distance. The location samples of the user are shown with a dot.

SecureRun, we *do not* consider the negative information when calculating the lower-bound distance. These situations are illustrated in Figure 4.4: It can be observed on the left-most figures that, when the user is not in the intersection of the communication ranges of two or more APs, considering negative information (bottom) increases the accuracy of the distance proof compared to the base case (top). It can also be observed on the right-most figures that, when the user is in the intersection, by omitting to report one of the LP she collects (bottom), she can unduly obtain a larger distance proof (potentially higher than the actual distance) compared to the base case (top).

4.4.4 Activity Summary

To publish an activity summary on her profile, the user uploads her collected activity proofs to the social-network service provider. In turn, the provider checks that (1) the signatures of the activity proofs are valid (using the public group keys of the access points), that (2) all the pseudonyms that appear in the activity proofs indeed belong to the same user, and that (3) time intervals of the activity proofs do not overlap (otherwise the distance covered in the time overlap would be counted twice, hence violating the lower-bound property of the summary). If this is the case, the social-network provider simply sums the distances (or the elevation gains, respectively) from the activity proofs and adds the resulting summary to the user's profile.

If the user's mobile device has an active wireless Internet connection (e.g., 3/4G), the user can upload on-the-fly the activity proofs she collects, while she pursues the activity. By doing so, it enables real-time online tracking of the user's activity by her friends, such as the live-tracking feature offered by the Runtastic and Endomondo mobile apps.

4.4.5 Sampling Algorithms

We now describe SecureRun's sampling algorithm. The sampling algorithm determines the sampling times/positions at which the user requests location proofs from the access points in her communication range. The general objective of the sampling algorithm is to achieve high accuracy (i.e., tight lower-bounds in the activity proofs) and a high level of privacy.

We distinguish between two cases: the case where a user knows beforehand the path she is about to take, namely *planned sampling*, and the case where she does not, namely

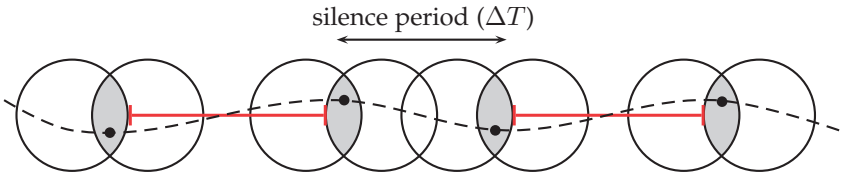


Figure 4.5: Silence period. By implementing a silence period between every pair of successive distance proofs (i.e., not requesting a distance proof for this period), a user reduces the risk of her distance proofs being linked by the access-point, hence protecting her privacy.

unplanned sampling. In both cases, the sampling algorithm knows the locations of the access points. Planned sampling corresponds to the quite common situation where a user records the set of her preferred paths and of her past activities. Such a feature is commonly implemented in activity-tracking apps (including Garmin's) in order to enable users to *compete* against their own previous performance. For instance, the activity-tracking app indicates to the user whether she is late or in advance, compared to her best performance. With planned sampling, the sampling points are determined before the user starts the activity with the full knowledge of the path and of AP locations, thus yielding potentially better results. We now describe both variants of the algorithm, considering at first the case of a single access-point operator and, subsequently, multiple operators.

We focus our description on the case of distance proofs.[4] The planned and unplanned versions of the algorithm share a common design rationale: (1) limit the discrepancies between the actual path and the lower-bounds, by requesting location proofs where the direction of the path changes significantly; and (2) enforce a silence period after requesting certain location proofs, in order to achieve unlinkability of successive activity proofs.

Silence Periods. To highlight the importance of silence periods, consider a user who collects three location proofs at three successive sampling times (with pseudonyms P_1, P_2 and P_3). If she requests a distance proof for the time interval between the first two location proofs and another distance proof for the time interval between the last two, the access-point operator can link the three location proofs (as it knows that P_1 and P_2 belong to the same user and so do P_2 and P_3) and thus track the user, despite the use of pseudonyms. To circumvent this issue, a user requests an additional location proof some time after she requests the second location proof, leaving her with four location proofs. The time between the second and the third (i.e., the additional) location proofs is called a *silence period*. Finally, the user requests distance proofs only for the time intervals between the first and the second, and between the third and the forth location proofs. The distance covered between the second and the third location proofs is not counted in the user's activity summary. The user repeats this process throughout her activity, as depicted in Figure 4.5. The duration ΔT of the silence period[5] is a parameter of the system that enables users to balance their accuracy of the activity summaries and their privacy: Short silence periods yield high-accuracy activity summaries (as the distances covered during the silence periods, which are not counted in the activity summary, are

[4]The reasoning is the same for elevation proofs.

[5]In practice, the length of the silence period is a random variable of mean ΔT (e.g., drawn for the uniform distribution on $[0.5\Delta T, 1.5\Delta T]$) in order to prevent an access-point operator from linking two distance proofs based on the time elapsed between them.

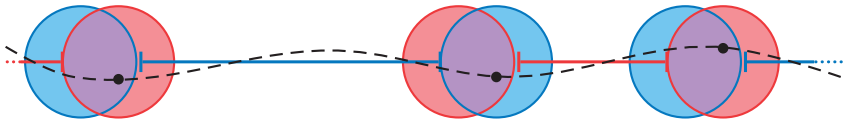


Figure 4.6: Case of multiple access-point operators (Operator 1 in red and Operator 2 in blue). At every sampling point, a user requests location proofs from both operators. Then, she requests distance proofs, alternatively from different operators, to reduce the risk of linking the distance proofs she collects without reducing the accuracy of her activity summary (unlike with silence periods).

small) but provide low privacy guarantees (as the access-point operators can link with high confidence two successive activity proofs, because the time interval between them is short). Conversely, long silence periods yield low-accuracy activity summaries and provide high privacy guaranties.

Multiple Access-Point Operators. In the case where multiple access-point operators are involved, the silence periods are not always needed: By requesting successive distance proofs from different operators (assumed to not collude with each other), a user does not need to wait for ΔT seconds (i.e., implement a silence period) to reduce the risks of linking her distance proofs. To protect her privacy, a user should not need to request two successive distance proofs from the same operator, unless she implements a silence period. With two operators, a user can alternatively request distance proofs from each of the two operators, as illustrated in Figure 4.6.

We now describe the planned and unplanned sampling algorithms.

Planned Sampling. In the planned case, we formalize the sampling problem at discrete time: The path of a user is represented as a sequence of time/position samples, ordered by increasing times $\{(\tau_i, p_i)\}_{i=1..T}$. Such a discrete formalism matches the conventional format of activity traces (e.g., Garmin represents the paths of activities as sequences of time/positions samples). In such a formalism, the sampling problem consists in selecting sampling times (i.e., , a subset of $\mathcal{T} = \{\tau_i\}_{i=1..T}$) at which a user needs to collect location proofs from the neighboring access points so that the sum of the resulting activity proofs is maximized. Using the notations from Equations 4.1 and 4.2, the sum of the distance proofs obtained by collecting locations proofs at times $\tau_{i_1}, \ldots, \tau_{i_K} \in \mathcal{T}$, with a single operator and without silence periods, is

$$\sum_{k=1}^{K-1} d\left(A_{i_k}, A_{i_{k+1}}\right), \tag{4.4}$$

and the optimal set of sampling points is the subset of $\mathcal{T}$ that maximizes this quantity.[6] This optimization problem is equivalent to a maximum-weight path problem in a directed graph G, where the vertices of G are the time samples (i.e., $\mathcal{T}$), and where the edges of G connect each time sample τ_i to all the subsequent time samples τ_j $(j > i)$, with a weight equal to the value of the distance proof obtained by combining the location proofs obtained at time τ_i, with those obtained at time τ_j (i.e., $d(A_i, A_j)$). A sample graph is shown in Figure 4.7, together with an example of a set of sampling points (in red). As G is a directed acyclic graph, the maximum-weight path problem can be solved in linear

[6] The exact same reasoning applies to elevation-gain proofs. The distance function $d()$ that appears in Equation 4.4 should be replaced with the elevation-gain function $e()$ from Equation 4.3.

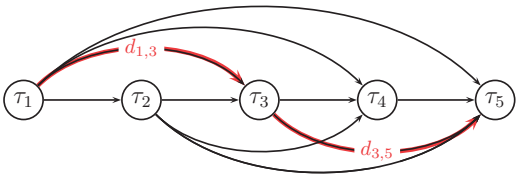


Figure 4.7: Graph construction for the maximum-weight path formulation of the sampling problem (one operator, no silence periods). The thick, red path shows an example of a set of sampling points: The user collects location proofs at times τ_1, τ_3, and τ_5. The total of the obtained distance proofs is $d_{1,3} + d_{3,5}$ ($d_{i,j}$ is the short for $d(A_i, A_j)$).

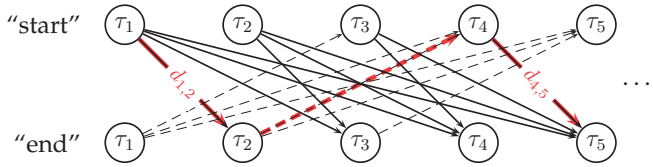


Figure 4.8: Graph construction for the maximum-weight path formulation of the sampling problem (one operator, with silence periods). Solid edges have weights equal to the corresponding distance proofs ($d_{i,j}$ is the short for $d(A_i, A_j)$). Dashed edges have zero weights. The thick, red path shows an example of a set of sampling points: The user starts a distance proof at time τ_1 (by collecting location proofs), which she ends at time τ_2 thus collecting a proof of value $d_{1,2}$; she then starts a new distance proof at time τ_4 (thus observing a silence period) which she ends at time τ_5. The total weight of this path is $d_{1,2} + d_{4,5}$.

time with respect to the number of edges, that is $\mathcal{O}(T^2)$. Unlike the heuristic proposed in [20], this formalization enables SecureRun to find the optimal set of sampling points, resulting in an absolute improvement of 5-10 points in the median accuracy of SecureRun (see Section 4.5).

In the case of silence periods, unlike in the base case, we must distinguish between the samples where a user collects a location proof to *start* a distance proof and those where she collects a location proof to *end* a distance proof. The case of a single operator with silence periods is illustrated in Figure 4.8. We build a graph with two vertices per time sample that corresponds to both situations ("start" and "end"). The "start" vertex corresponding to τ_i is connected to all the "end" vertices corresponding to τ_j ($j > i$), which means that a distance proof started at time τ_i can be ended at any time τ_j ($j > i$). The weight of such edges (depicted with solid lines in the figure) is the value of the distance proof obtained, i.e., $d(A_i, A_j)$. To start a new distance proof, the users must observe a silence period. Implementing silence periods means that a user cannot start a new distance proof before ΔT time units after she ended her last distance proof. In other words, assuming that the time between two time samples is 10 seconds and that $\Delta T = 15$, if a user ends a distance proof at τ_i, she can start a new distance proof at τ_{i+2} at the earliest. Therefore, we connect the "end" vertex corresponding to τ_i to all the "end" vertices corresponding to τ_j ($j > i + 1$). The weight of such edges (depicted with dashed lines) is zero.

In the case of multiple operators, we build a graph with two vertices per sample ("start"/"end"), *for each operator*. A distance proof started with a location proof of a given operator can be ended only with a location proof from the same operator, therefore the "start" vertices are connected only to the "end" vertices of the *same* operator. The "end" vertices, however, are connected to the start vertices of all the operators as a user

can start a new distance proof with any of the operators. The construction of these edges follows the same rationale as above, except that a user does not need a silence period when beginning a distance proof with an operator different than that of the previous distance proof.

Note that, in practice, a user normally does not follow the exact same path as she previously did. Therefore, the algorithm determines sampling points based on the previously recorded path, and the user requests location proofs when she reaches the *vicinity* of a pre-determined sampling point (e.g., within 20 m).

Unplanned Sampling. In the unplanned version, only the current and past positions of the user are known to the algorithm. A users first collects location proofs at the starting point of her activity (e.g., when she presses the "start" button on her mobile device). As the user pursues her activity, the algorithm periodically determines whether location proofs should be requested. To do so, the algorithm compares the actual distance covered since the last sampling point with the straight-line distance between the last sampling point and the current position. If the difference between the two distances is higher than a threshold, the algorithm triggers the collection of location proofs. To limit the rate at which location proofs are collected, we impose a minimal distance between two sampling points. Algorithm 1 presents the pseudo-code version of the unplanned sampling algorithm, where $d(\,\cdot\,,\,\cdot\,)$ denotes the distance between two locations.

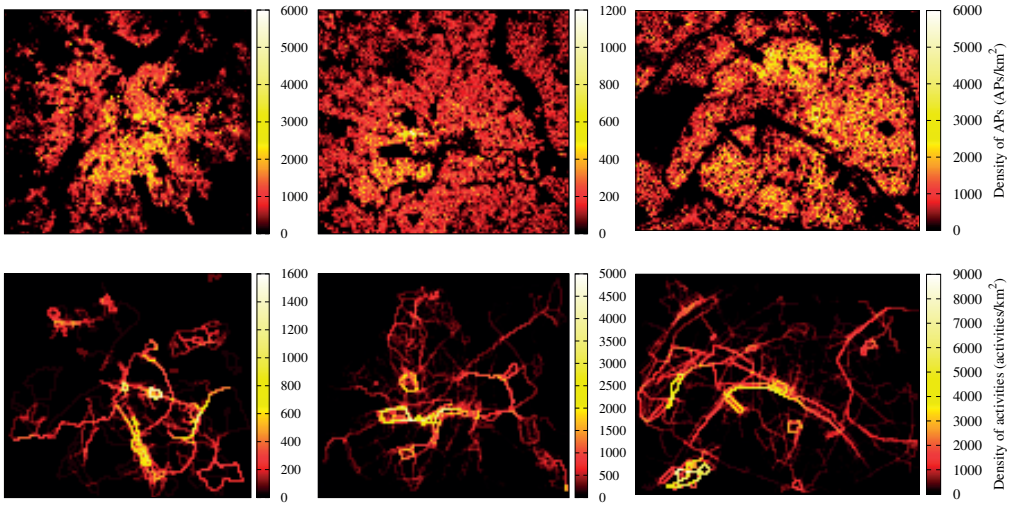
Algorithm 1 Unplanned sampling algorithm.

Input: MIN_LP ▷ Minimum distance between two LPs
MAX_LP ▷ Maximum distance between two LPs
MAX_ERR ▷ Maximum error

1: $S \leftarrow [\,]$ ▷ List of past locations since last sampling
2: **while** true **do**
3: $p_c \leftarrow$ current location
4: $S \leftarrow S + [p_c]$
5: $p_l \leftarrow S[1]$
6:
7: **if** $d(p_l, p_c) <$ MIN_LP **then**
8: **next**
9:
10: $e \leftarrow \left(\sum_{k=1}^{|S|} d(S[k], S[k+1]) \right) - d(p_l, p_c)$
11:
12: **if** $d(p_l, p_c) >$ MAX_LP **or** $e >$ MAX_ERR **then**
13: **sample()**
14: $S \leftarrow [p_c]$

4.4.6 Summary

In this section, we have presented SecureRun, a solution for providing secure and privacy-preserving activity summaries, and we have described in detail the different operations this involves. The inaccuracy of the activity summaries, defined as the difference between the lower bounds and the actual values, produced by SecureRun are due to the fact that (1) the distances covered inside the areas $\{A_i\}$ and the distances covered during the silence periods are not counted, and (2) the paths taken by the users between two areas are approximated with a straight line. We will report on the evaluation of the accuracy of SecureRun in the next section. The security and privacy properties of SecureRun are



(a) Brussels (b) London (c) Paris

Figure 4.9: Heat-maps of the densities of FON access points (top) and of Garmin activities (bottom) in (a) Brussels, (b) London, and (c) Paris.

provided by the use of pseudonyms and cryptographic techniques, by the aggregation and sanitization of data (with respect to location information), and by the silence periods. We discuss this in more detail in Section 4.7.

4.5 Evaluation

We evaluate SecureRun's performance on real traces of users' activities from Garmin Connect [199], pursued in cities where wireless access-point networks are deployed by the FON operator [200] (and possibly Free [220]). We consider scenarios where mobile users, equipped with Wi-Fi-enabled devices, report the total elevation gain and distance covered during their location-based activities (e.g., running). We focus our evaluation on three geographical areas that correspond to the cities of Brussels, London and Paris. We also discuss the practical aspects of the implementation of SecureRun on wireless routers.

4.5.1 Data Sets

In order to evaluate SecureRun, we collected data sets of access-point locations and activities, and we relied on the Google Elevation API. All the data we collected was publicly available and the data collection was in accordance with the policies of the ethical committee at EPFL (HREC).

Wi-Fi Access Points. In late 2013, we collected the geographic coordinates of the Wi-Fi access points from the FON community network in the region of Brussels, London and Paris. FON is a large community network with more than 14 million hotspots (12 million at the time of the data collection) worldwide; most of them are located in western Europe. FON achieves very high coverage in urban areas (up to 2,500 AP/km^2) through strategic partnerships with local ISPs (e.g., Belgacom, British Telecom, SFR): The routers of the ISPs' subscribers, provided by the partner ISP, act as FON hotspots. As ISPs hold

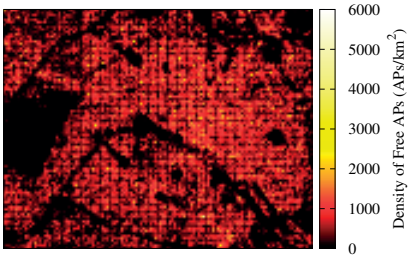


Figure 4.10: Heat-map of the density of Free access points in Paris.

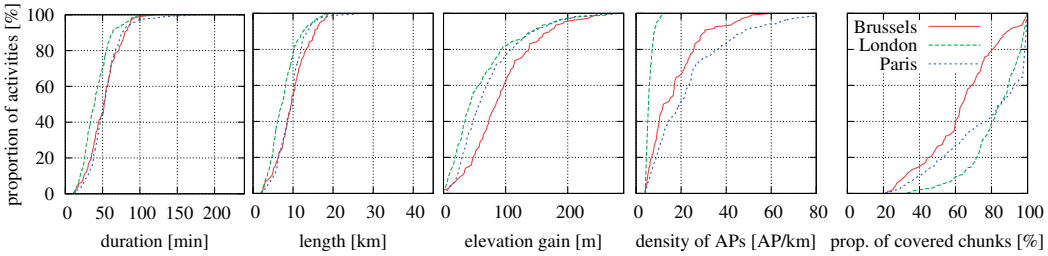
total control over the routers of their subscribers (through automatic firmware updates), they could easily implement and deploy SecureRun. Overall, we obtained the locations of 92,280 unique APs in Brussels, 39,776 unique APs in London, and 87,521 unique APs in Paris. In order to evaluate SecureRun with multiple access-point network operators (used jointly as described in the previous section), we also collected the geographic coordinates of the Wi-Fi access points from the Free community network. Free is a major French national ISP that offers community network features based on the routers of its subscribers. We obtained the locations of 60,280 unique APs from Free in Paris, which correspond to a density of 445±381 AP/km^2. Figure 4.9 (top) and Figure 4.10 depict, in the form of heatmaps, the densities of FON access points in Brussels, London and Paris and the densities of Free access points in Paris respectively. We can observe that the density of access points is low in regions that correspond to rivers, cemeteries, parks, highways and railways; this is due to the community nature of the FON network.

Activities. In early 2014, we collected activity information from Garmin Connect, an online service where users can upload and share information about their location-based activities, including the type of activity (e.g., running, biking) and the path of the activity (under the form of time-coordinates samples). We collected *running* activities and we computed, for each of them, the duration, the length and the cumulative elevation gain of the path, the inter-sample times, and the density of APs along the path (i.e., the number of APs met along the path, assuming a unit-disk communication model with a radius $R = 25$ meters, normalized by the length of the path). For each activity, we divided its path into chunks of 500 m, and we determined for each chunk whether it is covered by at least one access point (i.e., it intersects with the communication range of at least one access point). This metric is crucial for SecureRun to work, as having a high proportion of covered chunks ensures that users will be able to collect location proofs, and thus distance proofs. To exclude clear outliers or activities that are not covered by a minimal number of access points from our data set, we filtered out activities that (1) last less than 10 minutes or more than 4 hours, or (2) are shorter than 2 km or longer than 45 km, or (3) have a gap of more than 10 minutes between two samples, or (4) have less than 4 AP/km along their paths, or (6) have less than 20% of covered chunks. In the remainder of the chapter, we consider only the activities that pass the aforementioned filters (i.e., the *filtered data-sets*). Table 4.2 summarizes the filters applied to our raw data-set.

Figure 4.11 shows the distributions (experimental CDFs) of the main characteristics of the activities used in our evaluation, and Figure 4.9 (bottom) depicts the heat-maps of the densities of activities (i.e., the number of distinct activities that cross a given area

Property	Filter
Duration	<10 min or > 4 h
Length	<2 km or > 45 km
Elevation gain	> 50 m
(only for elevation gain summaries)	
Inter-sample times	> 10 min
Density of AP along activities	< 4 AP/km
Proportion of covered chunks	< 20%

Table 4.2: Summary of the filters applied to our activity data-set.



(a) (b) (c) (d) (e)

Figure 4.11: Experimental CDF of the (a) duration, (b) length, (c) elevation gain (d) density of FON AP (along the activity) and (e) proportion of chunks covered by FON APs, among the activities from the Garmin data-set.

	Brussels	London	Paris
Number of AP	92,280	39,776	87,521
Number of activities	107	294	437
Density of AP (AP/km^2)	401±569	109±96.6	646±686
Density of AP along path (AP/km)	17.1±12.0	5.99±1.67	23.8±18.6
Proportion of covered chunks (%)	63.9±20.0	83.0±15.0	77.7±23.5

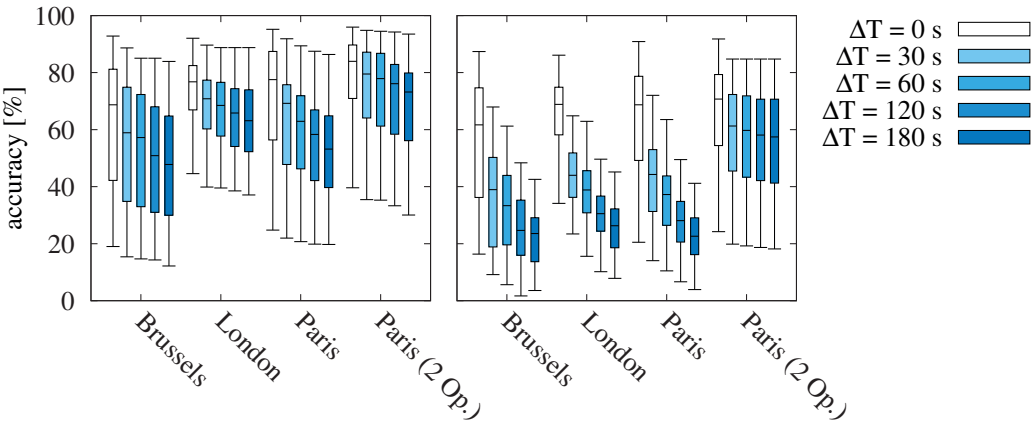
Table 4.3: Summary of the statistics of the filtered data-sets (FON and Garmin Connect) used in the evaluation (mean and standard deviation).

of the map). It can be observed that many activities take place in parks, where the density of access points is relatively low. In the filtered data set, we observe a median inter-sample time of 3-4 seconds (which correspond to 7-11 meters). Table 4.3 gives some statistics on our (filtered) data sets of access points and activities (i.e., FON and Garmin Connect). It can be observed that the density of access points is lower in London but they are more uniformly spread, especially along activities (as illustrated by the relatively small standard deviation compared to Brussels and Paris).

Elevation. We evaluate SecureRun for the case of elevation only in the region of Paris, and we filter out the activities with bogus elevation data or with an elevation gain lower than 50 m. In order to determine the minimum and maximum elevation in a given region, typically the intersection of communication disks centered at the AP locations, we rely on the Google Elevation API and sample the elevation of random points in the regions of interest. We make sure that every region contains at least 20 samples. We obtained a total of 701,793 such elevation samples.

Parameter	Description	Value	Unit
v_{max}	Maximum speed	3	m/s
R	Communication range of APs	25	m
MAX_LP	Maximum distance between two LPs	500	m
MIN_LP	Minimum distance between two LPs	50	m
MAX_ERR	Maximum relative error	10	m
δt	Threshold on the time difference between aligned LPs	5	s

Table 4.4: Parameters used in the simulation.



(a) planned sampling (b) unplanned sampling

Figure 4.12: Accuracy of the distance summaries, with the (a) planned and (b) unplanned sampling algorithms, for different values of the duration of the silence periods, with the FON network (+ Free for Paris 2 operators).

4.5.2 Methodology

We implement SecureRun in a Java simulator and evaluate its performance for the activities from the Garmin Connect data-set, with the access-point networks from the FON and Free data-sets (under the relaxed unit-disk communication model with a radius of 25 meters). The parameters used in our simulation are shown in Table 4.4. For each activity, we simulate the execution of SecureRun in different scenarios: with one or multiple access-point network operators, with the planned/unplanned sampling algorithm, and for different values of the parameters (such as the duration ΔT of the silence periods). For each such setting, we compute the corresponding activity summary. We measure the SecureRun's performance in terms of the *accuracy* of an activity summary: the ratio between the distance (respectively elevation) in the summary and the actual distance (respectively elevation) covered by the user during her activity.

4.5.3 Results for Distance Summaries

First, we look at SecureRun's absolute performance in different settings. Figure 4.12 shows a box-plot representation (first quartile, median, third quartile, and outliers) of SecureRun's accuracy in the (a) planned and (b) unplanned cases, in the cities of Brussels, London and Paris, for different durations of the silence periods. In the case of Paris, we also evaluate SecureRun with two access-point operators (Free and FON).

Overall, SecureRun achieves good performance: up to a median accuracy of 78%

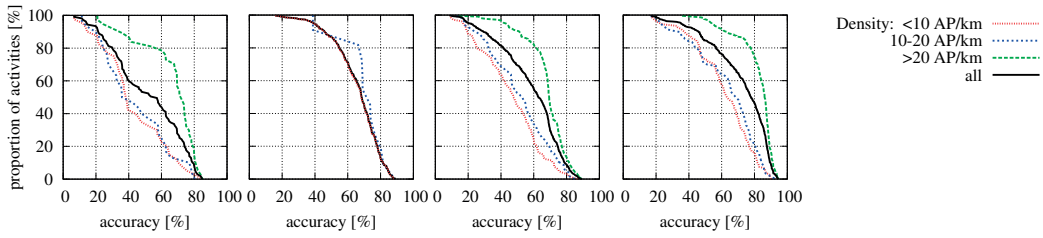
(Paris, FON, planned sampling, $\Delta T = 0$). This value drops to 69% when unplanned sampling is used. It can be observed that, as expected, the planned sampling algorithm yields consistently better results than the unplanned algorithm, and that the accuracy decreases with the duration of the silence period. In the case of two operators (in Paris), it can be observed that the accuracy is substantially better (84%) compared to the scenario with a single operator, when the duration of the silence period is set to 0. This is because a user can optimize the lengths of her distance proofs between the two operators. Moreover, the (negative) effect of the duration of the silence periods on the accuracy is substantially lower in the case of two operators (73% for the case of two operators vs. 53% for the case of a single operator, with planned sampling and ΔT=180 s). This is because silence periods are less frequently needed in such a scenario, only when a user requests a distance proof from an operator and cannot find any access points belonging to the other operator for the subsequent distance proof. Finally, the performances are quite similar across the different cities, with a slight advantage for London, as it has a higher proportion of covered chunks. This confirms our intuition and suggests that SecureRun's performance increases with the proportion of covered chunks. Compared to the initial version of SecureRun [221], we observed an absolute improvement of 5-10 points in the median accuracy. The improvement is lower for situations where the accuracy is already good, typically when no silence periods are used (except in the case of two operators). This means that the optimal sampling algorithm is most beneficial for adjusting the silence periods (except in the case of two operators in which they are less needed) and for choosing a best operator when there are several of them. Note that, as the number of Wi-Fi access points and their communication ranges are likely to increase in the coming years, SecureRun's performance is expected to increase. Moreover, with the emergence of activity-tracking services, the deployment of networks of wireless access points (including in parks) dedicated to activity tracking could become a thriving business, thus further increase the performance and applicability of SecureRun.

To further study the sensitivity of SecureRun to the density and the distribution of the access points (as captured by the number of AP/km and the proportion of covered chunks, respectively), we split the activities into three buckets, based on the values of these two metrics, and we plot the experimental cumulative density functions of the accuracy in each of these buckets. Activities with a low density of AP and/or a low proportion of covered chunks typically correspond to those located in parks; thus they do not really match our target context, i.e., urban areas. In the case of two operators, we only consider the values of the metrics with respect to the FON network.

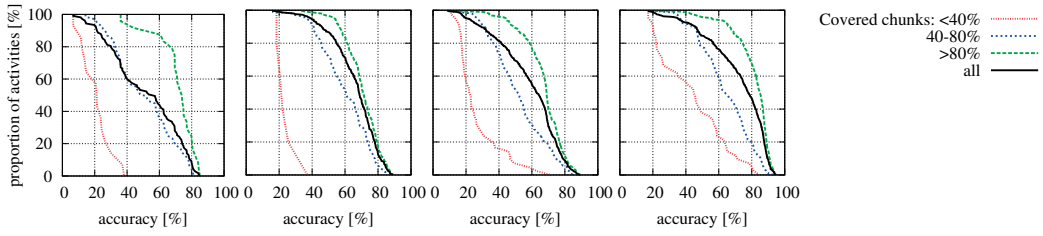
The results are depicted in Figure 4.13, with planned sampling and $\Delta T = 60$ s. It can be observed that the performance is substantially better for high densities and for high proportions of covered chunks, as compared to the low counterparts. In Brussels for instance, the median accuracy goes up to 74% for activities with high densities, whereas it is only 57% for all the activities. Note that even for some activities with a high density, the accuracy can be quite low (i.e., <25%). We investigated this issue by manually inspecting the paths of these activities and we report on the results in Section 4.5.5.

4.5.4 Results for Elevation-Gain Summaries

Figure 4.14 shows a box-plot representation of the accuracy of our solution for the case of elevation gain in Paris, with the planned sampling algorithm, for different durations of



(a) Brussels (FON) (b) London (FON) (c) Paris (FON) (d) Paris (2 Op.)



(a) Brussels (FON) (b) London (FON) (c) Paris (FON) (d) Paris (2 Op.)

Figure 4.13: Sensitivity analysis of the accuracy, with respect to the density of access point along the activities (top) and to the proportion of covered chunks (bottom). The plots represent complementary cumulative distribution functions (ccdf). The planned sampling algorithm was used, with silence periods of $\Delta T = 60$ s. Note that in London, all activities have a density $\leq$20 AP/km.

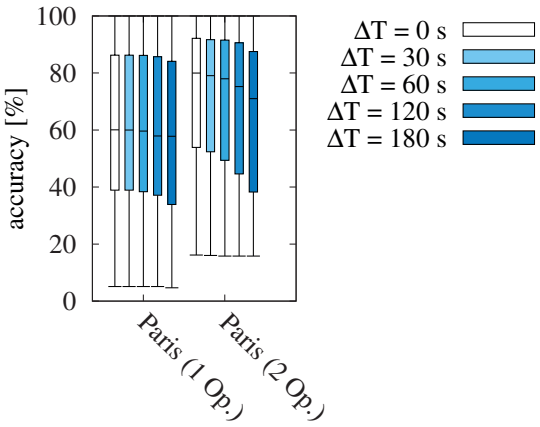


Figure 4.14: Accuracy of the elevation gains proofs in Paris, with the planned sampling algorithms, for different values of the duration of the silence periods, with the FON network (+ Free for 2 operators).

the silence periods. Overall, SecureRun achieves reasonable performance: up to a median accuracy of 60 % with one operator and silence periods of 60 seconds. This number goes up to 78 % when two operators are used.

4.5.5 Investigation of the Corner Cases

In this section, we report on our manual inspection of the paths of the activities for which SecureRun provides low-accuracy summaries despite the high density of access points along the path and the high proportion of covered chunks. During our investigation, we

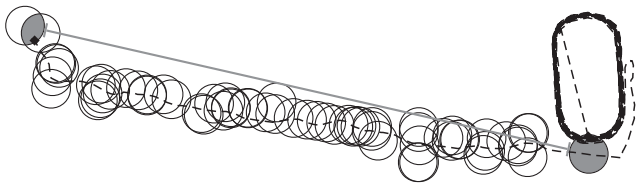


Figure 4.15: Example of an activity for which the proportion of covered chunks is greater than 80% and the precision is smaller than 25% (planned sampling, $\Delta T = 60$ s). The path is shown as a dashed line and the circles denote the communication ranges of the APs. The shaded areas represent the combined location proofs obtained at the sampling points.

found one typical case of such situations, which we show in Figure 4.15 and explain below. The general pattern is when a fraction of the path is very densely covered by access points but the rest is not; hence the average density over the whole path remains relatively high. More specifically, the user typically runs in a periodic fashion (e.g., around a stadium or back and forth on a street) on the part that is not well covered by access points, but this part is still covered by one or several APs (very close to each other) in a single location; hence the user cannot obtain any distance proofs (all the location proofs come from the same set of access points located close to each other) and almost all the chunks of the path are covered.

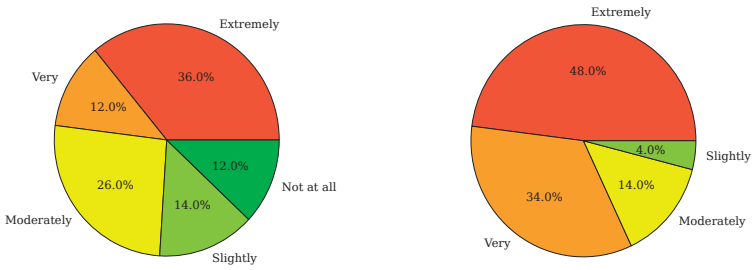
It can be observed in the sample case that the user first runs to a stadium through a residential area and then runs a dozen of times inside the stadium on the 400-meter track. Because the stadium is covered by a single AP, all the chunks of the activity are covered, but it is not sufficient to increase the accuracy as all LPs are obtained from the same AP.

4.5.6 Practical Aspects of SecureRun

For SecureRun to have low latency, it should be implemented at a low layer, typically at the MAC layer. Moreover, the computations involved in the AP-part of the protocol (e.g., cryptography, optimization) should be lightweight.

To better understand the technical challenges of the implementation of SecureRun on a router and to assess its feasibility, we developed a proof-of-concept prototype on a router (alix2d2 @ 500 MHz, 256 MB of RAM) running OpenWrt by using the Click framework[7] for network operations and OpenSSL for cryptographic operations. To do so, we defined a new frame at the MAC layer of the Wi-Fi protocol. In our experiments, we observed a delay of 9 milliseconds for the two-way message exchange between a user's mobile device and an AP to obtain an LP (for an ECDSA signature algorithm with a 224-bit key). The router managed to handle 12 clients that simultaneously send one LP request per second. As for the message exchange to obtain DPs, we estimated the additional running time incurred by the computation of a DP with IPOPT at 150 milliseconds (i.e., we ran it on a laptop and made a projection for a router based on the CPU speed). Note that as IPOPT's optimization algorithm is iterative, we can easily balance the running time and the accuracy of the computation. Note also that the computation of the DPs could be advantageously offloaded to a cloud server maintained by the AP network operator. Finally, to put the aforementioned delays in perspective, note that, in our dataset, a user

[7] http://www.read.cs.ucla.edu/click



(a) authenticity concerns (b) privacy concerns

Figure 4.16: Survey participants' concerns regarding (a) the authenticity of the activity data shared by their friends and (b) the privacy implications of the activity data they share.

remains in the connection range of an AP for about 10.7 seconds on average (i.e., the total delay for a message exchange between the user and the AP for a LP/DP is less than 2% of the in-range time).

As for the battery consumption overhead induced by SecureRun on the mobile device, SecureRun essentially consists of two main operations: the sampling algorithm and LP/DP collection. The unplanned sampling algorithm is lightweight and the planned sampling algorithm can be run offline (and only once per route). The LP/DP collection process is sporadic and involves cryptographic operations, the overhead of which is negligible compared to those performed by activity-tracking applications (e.g., continuous geolocation, HTTPS communication over 3G/4G).

4.6 Adoption of SecureRun

In order for SecureRun to be adopted, it must be beneficial to all the stakeholders: the users, the Wi-Fi AP network operator, and the social-network provider. In this section, we discuss and analyze the incentives for the stakeholders.

4.6.1 Users

In order to assess the interest, as well as the expectations, of users in SecureRun, we conducted a user survey in early 2015. Our survey involved 50 participants recruited through the Amazon Mechanical Turk platform. The online survey was composed of 11 questions covering general demographics, awareness and concerns about opportunities to cheat and privacy issues in activity-tracking systems, and satisfaction and expectations regarding the accuracy of cheat-proof and private summaries (see Appendix B for the full transcript of the survey questionnaire). The survey took approximately two minutes to complete; we paid each participant US $2. In order to be allowed to participate in our survey, participants were required to have an active RunKeeper account[8] and a minimum Human Intelligence Task (HIT) approval rate of 95% with at least 100 past approved HITs. We obtained a diverse and balanced sample of participants: 44% of male and 56%

[8] An active account is an account with at least 20 running activities, recorded with the mobile apps, with a duration of at least 10 minutes and a distance of at least 2 miles over the time period spanning from 2013-01-01 to 2014-12-01. In order for us to check this, the participants had to grant us access to their RunKeeper account through the API

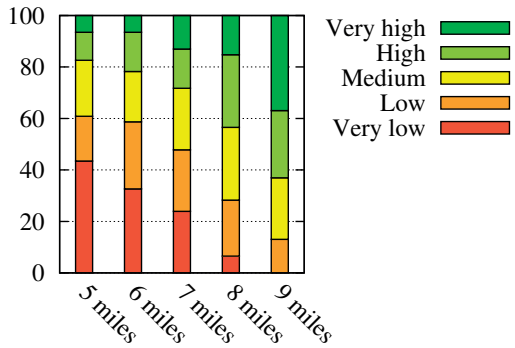


Figure 4.17: Level of satisfaction of the survey participants for various values of the accuracy of activity summaries.

of female, diverse primary areas of employments, ages ranging from 20 to 47 years old (avg.: 31, std.: 5.9). Most participants used a single activity-tracking service.

We polled the participants about their awareness of applications that enable users to cheat (i.e., claim performances they did not actually achieve) and about their concerns regarding the authenticity of the performance reported by their friends. We depicted the results in Figure 4.16. None of the participants were aware of the existence of such applications; however, a large proportion of the participants (48%) were very or extremely concerned about them. Similarly, we polled the participants about their awareness of the privacy implications of activity-tracking services (i.e., the ability to infer home/work locations, demographic information, and medical conditions, and the fact that activity data can be sold to third parties such as insurance companies[9]) and about their concerns regarding these issues. Only 10% of the participants were aware of the existence of such privacy issues; a large majority of the participants (82%) were very or extremely concerned about it. These results clearly make the case for SecureRun.

Finally, we briefly introduced SecureRun to the participants and we polled them about their satisfaction (on a 5-point Likert scale) if, for a 10-mile run, SecureRun provided them with a certificate of 5, 6, 7, 8 or 9 miles. We show the results in Figure 4.17. It can be observed that for an accuracy of 80% or more (which is the case for SecureRun in representative scenarios), the participants' opinions were mostly positive.

4.6.2 Wi-Fi AP-Network Operators and Service Providers

Social network providers[10] offering activity-tracking services have incentives to deploy SecureRun as it increases their attractiveness to the users and certifies the data provided by their users. In order to deploy SecureRun, the social-network provider needs the help of one or multiple Wi-Fi AP-network operators. The social-network provider could pay the AP-network operator for the certified activities, in a B2B fashion, either directly (e.g., in currency) or through advertisement. For instance, we envision a business model in which the activities that are certified by distance proofs provided by the FON network and uploaded on Garmin Connect or RunKeeper are displayed with an icon "Certified by FON", thus providing FON with ad space on the activity-based social network website.

[9] http://www.dailymail.co.uk/news/article-2409486/

[10] The same applies to health insurance companies that provide their customers with activity-tracking apps.

The AP-network operator could also charge the distance proofs to the users (e.g., via a specific subscription). Finally, with the emergence of activity-tracking services and inexpensive low-power Bluetooth beacons (e.g., iBeacons), wireless networks dedicated to activity-tracking could be deployed (e.g., in parks).

4.7 Security and Privacy Analysis

In this section, we discuss the security and privacy guarantees provided by SecureRun, by considering the adversarial model from Section 4.3 (i.e., three possible adversaries: the users, the service provider and the AP operator(s)).

4.7.1 Cheat-Proof Guarantees

Adversary: User. Malicious users can attempt to cheat by forging or modifying location or activity proofs. To prevent these types of attacks, SecureRun relies on digital signatures to protect the integrity of the proofs created by the APs. Moreover, users cannot double-count some of the distances they cover, because the service provider validates that the time intervals on each activity proof do not overlap before summing them up. Note that as the service provider does not permit users to declare activities that overlap in time, a user cannot use twice a collected distance proof. Users could also try to claim other users' location or activity proofs as their own. However, the service provider can detect such cheating attempts by checking that the pseudonyms included in the activity proofs belong to the user claiming them. A malicious user could also ask other users to collect proofs on her behalf by sharing her pseudonyms. To discourage such attempts, pseudonyms include sensitive information such as tokens to reset the user's password or modify certain service settings.[11] Also note that pseudonyms are sent encrypted to the APs with their group public key to prevent eavesdropping by other parties and that our threat model assumes honest-but-curious APs (i.e., they do not abuse the information on the pseudonyms).

SecureRun also requires users to be in the communication range with the APs to obtain valid location proofs. Users can try to bypass this restriction by launching proxy attacks, in which two or more users collude to obtain valid location proofs. Still, such attacks can be limited by introducing constraints on the execution time of the protocol (i.e., distance-bounding mechanisms). For instance, the AP operator could impose a communication delay on the Wi-Fi interface that is smaller than the one achieved by connecting through the cellular network.

Finally, the activity proofs obtained are, by design, lower-bounds of the performance achieved by the users. Thus, independently of the way users obtain and combine their location proofs, the reported summary will always be lower than the actual performance.

4.7.2 Privacy Guarantees

Adversary: Service Provider. In our mechanism, the service provider has access only to activity proofs and pseudonyms of the users. As activity proofs do not contain any location information, the service provider cannot link the activity proofs to the actual

[11] A similar approach was proposed in systems such as PKIs [222] and anonymous credentials [223] to ensure *non-transferability*.

locations. In a region covered by APs, given the number of access points that issued location proofs to users, a distance (more precisely, its lower bound) can be attributed to many possible trajectories between any two sets of APs, hence rendering unfeasible an accurate inference of the actual locations and trajectory. Moreover, as the distance also depends on the time difference between the location proofs, attributing a single distance to a given trajectory is even more challenging.

Adversary: AP Operator(s). SecureRun is also designed to prevent curious AP operator(s) from tracking users' locations, particularly by linking location proofs to reconstruct users' trajectories. For this purpose, SecureRun relies on both randomized pseudonyms (generated by the service provider) and silence periods. On every sampling point, a new pseudonym is used, therefore it is difficult for AP operator(s) to link users' proofs and activities. For single-operator scenarios, the use of silence periods reduces the chances of an AP operator linking users' proofs during an activity, as described in Section 4.4.5. Note that, even if no silence periods are used, the operator can track a user only during her activity, without being able to link different activities over time. Thus, this prevents the AP operator from inferring patterns from activity trajectories over time. In addition, unlike the service provider, the operators have no personal information about the users (e.g., their names). Also note that, because of the need for physical proximity to interact with the APs, the privacy features of SecureRun are offerred on a best-effort basis. We do not provide formal privacy, but it still provides significant improvements compared to the current solutions of activity-tracking services.

Finally, quantifying the location-privacy of users when pseudonyms and silence periods are employed is a typical mix-zone [224]/mix-network problem; a mix-zone allows users to change their pseudonyms and mix with each other, without being observed. Note, however, that in SecureRun the pseudonym-change strategy optimizes the sum of the distance proofs, whereas for traditional mix-zones it optimizes the users' privacy. In such situations, the location privacy of a user depends on the other users, as shown in [225, 226, 227], where the higher the number of users is, the better their privacy is.

4.8 Conclusion

Activity-based social networks have become increasingly popular over the last few years. In their current form, such systems rely on the users' mobile devices to report the users' real locations while they pursue their activities. This provides neither security guarantees against cheaters, nor privacy protection against curious social-network providers, thus potentially discouraging their adoption.

In this chapter, we have proposed SecureRun, a system for providing secure and private proofs of location-based activities. SecureRun relies on the existing wireless access-point networks deployed in urban areas (at the cost of only a software upgrade, hence alleviating the need for deploying ad-hoc infrastructures), and it provides protection for both users and service providers. Our experimental evaluation, conducted using real data-sets of deployed wireless access-points and actual users' outdoor activities, shows that SecureRun achieves a good accuracy (up to 79%) when estimating a lower-bound of the distance that users cover during their activities, and it provides privacy and security properties. From a practical perspective, we envision our scheme to be of interest for strategic partnerships between social-network providers and access point network operators. In addition, our proof-of-concept implementation of SecureRun on a router has shown that it can be

deployed in practice. As such, this work constitutes a first step towards the design of secure and private activity-based social networks.

Chapter 5

Conclusion

In this thesis, we have performed threat analyses for popular mobile apps and services, and have designed privacy protection mechanisms for them by using state-of-the-art privacy enhancing technologies. We have shown that it is possible to enable users to benefit from added privacy, while enjoying the rich functionality and convenience of mobile apps and services.

In Chapter 2, we have shown that Android allows apps, even those with zero permission, to fingerprint and identify other installed apps. This is problematic for mHealth apps because their presence might already reveal the medical conditions of their users. We have systematically analyzed and shown a diverse set of information leaks that nosy apps can exploit (through the Linux-layer interface and the Java API framework) to identify other apps. Also, our static and dynamic analysis, conducted on a large number of apps from the Google Play Store, has shown that apps are interested in learning about the presence of other installed apps. Our solution, HideMyApp, effectively hides the presence of sensitive apps, does not require any modifications to the Android operating system, and preserves the key functionality of apps. The thorough evaluation of HideMyApp on a diverse set of apps from the Google Play Store suggested that it is practical and usable. This is also confirmed by the results of our user study.

In Chapter 3, we have analyzed the privacy threats in ride-hailing services. We have also proposed ORide, a practical solution that provides location privacy for riders and drivers with respect to the service providers, and privacy for the drivers with respect to malicious outsiders. By employing and optimizing the state-of-the-art somewhat-homomorphic encryption system for the scenario of ride-hailing services, ORide efficiently and effectively supports the anonymous matching of riders and drivers. The location privacy of the rider's destination is still guaranteed, even in the extreme case of targeted attacks, i.e., a curious service provider wants to know the destination of a rider given the time and location of a rider's pick-up event. In addition, ORide still offers key features of ride-hailing services, such as easy payment, reputation scores, accountability, and retrieval of lost items.

In Chapter 4, we have proposed SecureRun, a system for providing secure and private proofs of location-based activities. By relying on existing wireless access-point networks, SecureRun provides privacy protection for the users with respect to service providers, while enabling the service providers to compute accurate summaries and statistics of

users' activities. Our experimental evaluation, conducted using real data-sets of deployed wireless access-points and actual users' outdoor activities, shows that SecureRun achieves a good accuracy when estimating a lower-bound of the distance and elevation gain that users covered during their activities. In addition, as shown by the results of our user study, SecureRun is perceived to be usable and of high interest to users.

In conclusion, this thesis has shown that it is technically possible to provide privacy for mobile apps and services, and to balance the trade-off between privacy and usability. For each privacy-preserving solution presented in the thesis, we have shown the entire development process by spanning from the analysis of possible threats, the definition of the privacy and security requirements, to the design of a potential privacy-preserving system and its evaluation and usability. Analyzing security and privacy threats in the smartphone ecosystem, and correspondingly addressing them, is a challenging and fascinating endeavor, and the author hopes that the contributions presented in this thesis will inspire more future work on the preservation of the users' privacy.

Future Work

Despite the concrete solutions provided in this thesis, as the smartphone ecosystem continues to grow, its attack surfaces will also augment, and the prevention of possible privacy risks is an "arms race" between the defenders and the attackers. As technology continues to quickly advance, many novel use-cases will emerge. For example, self-driving cars, which rely on a plethora of sensors and data in order to ensure safe rides and also provide users with convenient features, will also enable the car providers to have a detailed look into users' habits and activities. A number of ride-hailing services has considered using self-driving cars instead of human-driving cars [228]. In such a case, the privacy-preserving design presented in this thesis needs to be adjusted to a different threat model, due to the collusion between the self-driving systems of the cars and the service providers. Technological advances will also help increase the accuracy of measurements provided by quantified-self devices. These devices will enable users to record and share not only their location-based activities but also other sensitive personal data, such as their heart rate and galvanic skin response. The Internet of Things (IoT) technologies will also expose users' habits to the service providers, hence privacy risks. Therefore, protecting user privacy requires continuous efforts from the research community to understand the technologies and to design privacy-enhancing solutions for them.

In addition to research efforts, on the engineering side, it is important to educate app developers about possible security and privacy threats caused by smartphones to their apps, and about tools that they could use to mitigate the problems. The status quo is that even if operating-system providers provide app developers with security/privacy guidelines or tools, the developers might not follow the guidelines or use the tools properly. For example, even with sensitive apps such as mHealth apps, developers still stored unencrypted data on SD cards, sent data over the Internet with insecure protocols (e.g., HTTP) or misconfigured HTTPS [229]. In addition, to bring privacy-preserving systems from the academic world to the real world, we should raise the awareness of the general public about possible privacy risks caused by their mobile apps and services. Consequently, they can demand better privacy guarantees. Only then will we enjoy an equitable society, where technology advances and privacy co-exist.

Appendices

Appendix A

RHS Threat-Taxonomy Details

In Table A.1, we present a more detailed description of the main threats against current RHSs (Section 3.3.2 of Chapter 3), and the key reasons for their associated risk level.

Privacy Threats

Description	Risk	Risk Level Explanation
1) A malicious rider collects PII from drivers assigned to her.	Low	Medium impact: semi-scalable attack due to system design, limited reputation damage for the SP. Low likelihood: only few riders might attempt such attacks due to the reward, few reported incidents [109]
2) A malicious driver collects PII from riders she is assigned to.	Low	Low impact: non-scalable attack due to system design, limited reputation damage for the SP. Low likelihood: only few drivers might attempt such attacks due to the reward, no reported incidents.
3) A curious SP is able to track, sometimes in real-time, riders' precise location during their rides and infer private information from the data observed.	High	High impact: large scale inference attacks, significant reputation damage for the SP. High likelihood: ride data is collected by the SP by design, significant financial gain, reported incidents of misuse of this data [13, 96, 110].
4) A curious SP is able to track, sometimes in real-time, drivers' precise location while on service and infer private information from the data observed.	Medium	Medium impact: large scale inference attacks, drivers' ride data reveal less private information (because drivers' locations are determined by riders' requests), reputation damage for the SP. Medium likelihood: ride data is collected by the SP by design, privacy-inference analysis is more difficult, less reward than riders data.
5) A malicious outsider can collect riders' PII in large scale by exploiting weaknesses on the services offered by the SP.	Medium	High impact: large scale attack, significant reputation damage to SP. Low likelihood: considerable reward, difficult to exploit because SP's services exposed little rider information, requires higher attack skills, no reported incidents.
6) A malicious outsider can collect drivers' PII in large scale by exploiting weaknesses on the services offered by the SP.	High	High impact: scalable due to system design (i.e., an outsider can create many accounts, fake locations, make ride requests and immediately cancel the ride requests when the drivers' information is returned), possible privacy violation of thousands of individuals, harvested data could be used to physically attack drivers. High likelihood: valuable information for some adversaries, system design facilitates the attack, current security mechanisms provided limited protection, reported related incidents [128, 129].
7) Malicious outsider gains (partial or total) unauthorized access to the SP's rider and/or driver registration databases.	High	High impact: significant reputation damage to the SP, possible privacy violation for millions of individuals. High likelihood: significant reward for the adversary, very common attack against online services, reported incidents [112].

Integrity Threats

Description	Risk	Risk Explanation
1) A malicious rider exploits a weakness on the system to pay lower fares.	Low	Low impact: non-scalable, limited financial loss, affects a small number of drivers. Medium likelihood: riders have limited input on fare calculation operations, limited financial gain, reported incidents [113], attacks are easy to detect.
2) A malicious rider exploits weaknesses on the system to improperly increase her reputation (e.g., to hide misbehavior).	Low	Low impact: non-scalable, limited reputation and financial cost to the SP. Low likelihood: rider reputation does not have high value, riders have little control over reputation operations, it may be easier to create a new rider account.
3) A malicious rider improperly accumulate incentives offered by the system for financial gain.	Low	Low impact: limited financial and reputation cost to the SP, incentives are location-dependent, SP can invalidate incentives. Medium likelihood: few incidents reported [114], interesting rewards for riders, abuse is easy to detect .
4) A malicious driver exploits weaknesses on the system to improperly charge riders a higher fare and/or fees.	High	High impact: large-scale attack if tools are made available, considerable reputation damage for the SP. Medium likelihood: drivers provide the inputs for fare calculation, smartphones are relatively easy to tamper with, direct financial gain, some related incidents has been reported [115, 116].
5) A malicious driver fakes her location or availability to increase her chances of being assigned to a ride.	Low	Low impact: limited scalability (only useful in some scenarios), limited financial impact to other drivers. Medium likelihood: attack is easy to execute, available tools, direct financial reward, reported incidents [114].
6) A malicious driver exploits weaknesses in the system to improperly increase her reputation.	Low	Medium impact: scalable if tools are made available, considerable reputation and financial loses for SP (e.g., , bad drivers are not banned). Low likelihood: difficult to exploit as drivers have read-only access to reputation information, significant motivation, no reported attacks.
7) A malicious driver exploits weaknesses on the system to improperly avoid fees or obtain incentives from the SP.	Medium	Medium impact: scalable if tools are made available, significant financial loses for the SP, incentives are location-dependent. Medium likelihood: significant financial motivation, reported incidents [117], some attacks are easy to detect.
8) A malicious outsider steals riders' and drivers' accounts by attacking the SP drivers or riders.	High	High impact: scalable attack, significant financial impact to riders and SP, significant reputation damage to the SP. High likelihood: common attack against online services, reported incidents [118], considerable reward.

Table A.1: Extended description and explanation of risk level for the main threats of RHSs presented in Table 3.1 (Section 3.3.2).

Appendix B

SecureRun Survey Details

In this appendix, we give the complete transcript of our survey questionnaire used in the evaluation of our SecureRun system (Chapter 4). Long lists of options have been truncated for the sake of conciseness. Our online questionnaire was designed with the LimeSurvey system and interfaced with the HealthGraph API in order to access the participants' RunKeeper account data. We used this data only for screening purposes.

Demographics

1. What is your gender?
   - ◯ Male
   - ◯ Female
2. How old are you? ☐
3. What is your primary area of employment?
   - ◯ Retired
   - ◯ [. . .]
   - ◯ Transportation
   - ◯ Other: ☐
4. Besides RunKeeper, which of the following fitness applications are you a member of?
   - ☐ Strava
   - ☐ Runtastic
   - ☐ GarminConnect
   - ☐ Moves
   - ☐ Endomondo
   - ☐ MapMyFitness
   - ☐ Other: ☐

Fitness data sharing on online social networks

5. How often do you share your fitness activities with your friends?
   - ◯ Always
   - ◯ It depends
   - ◯ Never
6. When do you share your location-based fitness activities with your friends? [shown only if the answer to the previous question is "It depends"]
   - ☐ When I take a new path
   - ☐ When I break a record
   - ☐ When I want to compete with myself or with my friends
   - ☐ Other: ☐
7. Applications such as digitalEPO.com and Fake Track enable users to claim a performance that they did not actually achieve. Were you aware of this fact?
   - ◯ Yes
   - ◯ No
8. Knowing this fact, how important to you is the authenticity of the fitness activities your friends share?
   - ◯ Extremely
   - ◯ Very
   - ◯ Moderately
   - ◯ Slightly
   - ◯ Not at all
9. Sensitive information can be inferred from the data you upload on RunKeeper (e.g., home/work locations, medical conditions). Moreover, it has been shown that some popular fitness applications pass personal details about their users to insurance companies, e.g., to set premiums (Click here for more details*). Were you aware of this fact?
   - ◯ Yes
   - ◯ No
10. Knowing this fact, how important to you are the privacy implications of the data you upload on RunKeeper?
    - ◯ Extremely
    - ◯ Very
    - ◯ Moderately
    - ◯ Slightly
    - ◯ Not at all

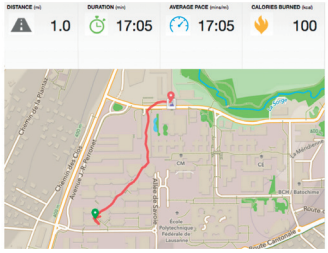
*http://www.dailymail.co.uk/news/article-2409486/

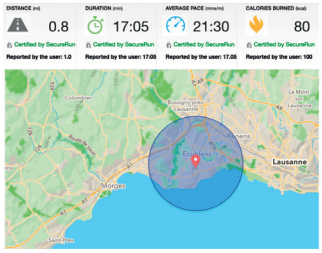
Figure B.1: Transcript of our survey questionnaire (1/2).

Secure and private activity summaries

11. We designed a system, named SecureRun, that provides you with two main features:
    - It protects your privacy. Specifically, the GPS traces of your activities is known only to you. You share only the summaries of your performance (e.g., the covered distance), and the coarse-grained information about the region where you perform your activities.
    - It guarantees the authenticity of a fraction of the performance you report. For example, if you run 10 miles and report it, SecureRun can certify (based on cryptographic techniques) that you indeed ran at least 8 miles out of the 10 miles you ran.

    We illustrate the differences between RunKeeper and SecureRun in the images below.





Assuming that you have run 10 miles, please choose your levels of satisfaction for the different values for which you would receive certification from SecureRun.

	Very low	Low	Medium	High	Very high
5 miles	◯	◯	◯	◯	◯
6 miles	◯	◯	◯	◯	◯
7 miles	◯	◯	◯	◯	◯
8 miles	◯	◯	◯	◯	◯
9 miles	◯	◯	◯	◯	◯

Figure B.2: Transcript of our survey questionnaire (2/2).

Bibliography

[1] "Number of smartphone users worldwide from 2014 to 2020 (in billions)," https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/, last visited: Sep. 2018. [cited at p. 1]

[2] "Store Stats," https://42matters.com/stats, last visited: Jul. 2018. [cited at p. 1]

[3] "Report: Smartphone owners are using 9 apps per day, 30 per month," https://techcrunch.com/2017/05/04/report-smartphone-owners-are-using-9-apps-per-day-30-per-month/, last visited: Sep. 2018. [cited at p. 1, 8, 21]

[4] M. Naveed, X. Zhou, S. Demetriou, X. Wang, and C. A. Gunter, "Inside job: Understanding and mitigating the threat of external device misbonding on Android," in *Proc. of IEEE NDSS*. [cited at p. 2, 10, 64]

[5] "Amid Egypt's anti-gay crackdown, gay dating apps send tips to stop entrapment," https://www.reuters.com/article/us-egypt-gay-apps/amid-egypts-anti-gay-crackdown-gay-dating-apps-send-tips-to-stop-entrapment-idUSKBN1CS0Z5, last visited: Sep. 2018. [cited at p. 2]

[6] X. Zhou, S. Demetriou, D. He, M. Naveed, X. Pan, X. Wang, C. A. Gunter, and K. Nahrstedt, "Identity, location, disease and more: Inferring your secrets from android public resources," in *Proc. of ACM CCS*, 2013. [cited at p. 2, 8, 10]

[7] "116 Amazing Social Media Statistics and Facts)," https://www.brandwatch.com/blog/96-amazing-social-media-statistics-and-facts/, last visited: Sep. 2018. [cited at p. 2]

[8] "97 Amazing Uber Statistics, Demographics and Fact)," https://expandedramblings.com/index.php/uber-statistics/, last visited: Sep. 2018. [cited at p. 2, 33]

[9] "Despite privacy outrage, AccuWeather still shares precise location data with ad firms," https://www.zdnet.com/article/accuweather-still-shares-precise-location-with-advertisers-tests-reveal/, last visited: Sep. 2018. [cited at p. 2]

[10] "Facebook has been collecting call history and SMS data from Android devices," https://www.theverge.com/2018/3/25/17160944/facebook-call-history-sms-data-collection-android, last visited: Sep. 2018. [cited at p. 2]

[11] "Facebook still has not fixed a potentially illegal problem with its advertising platform," https://mashable.com/2017/11/21/facebook-still-discrmination-pro-publica/#G01byQh03Pq7, last visited: Sep. 2018. [cited at p. 2]

[12] "China Moves To Create The Perfect Totalitarian Dystopia," https://www.dailykos.com/stories/2018/2/3/1738528/-China-Moves-To-Create-The-Perfect-Totalitarian-Dystopia, last visited: Sep. 2018. [cited at p. 3]

[13] http://www.oregonlive.com/today/index.ssf/2014/11/sex_the_single_girl_and_ubers.html. Last visited: Sep. 2018. [cited at p. 3, 33, 37, 38, 40, 96]

[14] "U.S. soldiers are revealing sensitive and dangerous information by jogging," http://wapo.st/2BDFrA4, last visited: Sep. 2018. [cited at p. 3, 64]

[15] "2018 reform of EU data protection rules," https://ec.europa.eu/commission/priorities/justice-and-fundamental-rights/data-protection/2018-reform-eu-data-protection-rules_en, last visited: Sep. 2018. [cited at p. 3]

[16] T. V. A. Pham, I. I. Dacosta Petrocelli, E. Losiouk, J. Stephan, K. Huguenin, and J.-P. Hubaux, "HideMyApp: Hiding the presence of sensitive apps on Android," 2018, under submission. [cited at p. 5]

[17] A. Pham, I. Dacosta, B. Jacot-Guillarmod, K. Huguenin, T. Hajar, F. Tramèr, V. Gligor, and J.-P. Hubaux, "PrivateRide: A Privacy-Enhanced Ride-Hailing Service," in *Proc. of PoPETs*, 2017. [cited at p. 5, 34, 39, 54]

[18] T. V. A. Pham, I. I. Dacosta Petrocelli, G. F. M. Endignoux, J. R. Troncoso-Pastoriza, K. Huguenin, and J.-P. Hubaux, "Oride: A privacy-preserving yet accountable ride-hailing service," in *Proc. of USENIX Security Symposium*, 2017. [cited at p. 5]

[19] A. Pham, K. Huguenin, I. Bilogrevic, I. Dacosta, and J.-P. Hubaux, "SecureRun: Cheat-proof and private summaries for location-based activities," *IEEE Trans. on Mobile Computing*, vol. 15, no. 8, pp. 2109–2123, 2016. [cited at p. 5]

[20] A. Pham, K. Huguenin, I. Bilogrevic, and J.-P. Hubaux, "Secure and private proofs for location-based activity summaries in urban areas," in *Proc. of ACM UbiComp*, 2014. [cited at p. 5, 65, 76]

[21] M. Aitken and J. Lyle, "Patient adoption of mhealth: use, evidence and remaining barriers to mainstream acceptance," *IMS Institute for Healthcare Informatics*, 2015. [cited at p. 7]

[22] "The mHealth apps market is getting crowded." https://research2guidance.com/mhealth-app-market-getting-crowded-259000-mhealth-apps-now/, last visited: Sep. 2018. [cited at p. 7]

[23] D. Kotz, C. A. Gunter, S. Kumar, and J. P. Weiner, "Privacy and security in mobile health: A research agenda," *Computer*, vol. 49, no. 6, June 2016. [cited at p. 7]

[24] S. Seneviratne, A. Seneviratne, P. Mohapatra, and A. Mahanti, "Predicting user traits from a snapshot of apps installed on a smartphone," *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 18, no. 2, Jun. 2014. [cited at p. 7, 10]

[25] ——, "Your installed apps reveal your gender and more!" *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 18, no. 3, 2015. [cited at p. 7, 10]

[26] S. Demetriou, W. Merrill, W. Yang, A. Zhang, and C. A. Gunter, "Free for all! assessing user data exposure to advertising libraries on android," in *Proc. of NDSS*, 2016. [cited at p. 7, 8, 10, 14]

[27] J. P. Achara, G. Acs, and C. Castelluccia, "On the unicity of smartphone applications," in *Proc. of ACM WPES*, 2015. [cited at p. 7, 10]

[28] "Twitter scanning users' other apps to help deliver 'tailored content' ," https://www.theguardian.com/technology/2014/nov/27/twitter-scanning-other-apps-tailored-content, last visited: Sep. 2018. [cited at p. 8]

[29] "About Twitter's app graph," https://help.twitter.com/en/safety-and-security/app-graph. [cited at p. 8]

[30] M. C. Grace, W. Zhou, X. Jiang, and A.-R. Sadeghi, "Unsafe exposure analysis of mobile in-app advertisements," in *Proc. of ACM WiSec*, 2012. [cited at p. 8, 10]

[31] "Android Security 2016 Year In Review," https://source.android.com/security/reports/Google_Android_Security_2016_Report_Final.pdf, last visited: Sep. 2018. [cited at p. 8, 18]

[32] "Android Security 2015 Year In Review," https://source.android.com/security/reports/Google_Android_Security_2015_Report_Final.pdf, last visited: Sep. 2018. [cited at p. 8, 18]

[33] "Android Developer Policy Center," https://play.google.com/about/developer-content-policy-print/, last visited: Sep. 2018. [cited at p. 8]

[34] A. M. McDonald, R. W. Reeder, P. G. Kelley, and L. F. Cranor, "A Comparative Study of Online Privacy Policies and Formats," in *Privacy Enhancing Technologies*, 2009. [cited at p. 8, 31]

[35] "Additional protections by Safe Browsing for Android users," https://security.googleblog.com/2017/12/additional-protections-by-safe-browsing.html, last visited: Sep. 2018. [cited at p. 8]

[36] J. Sunshine, S. Egelman, H. Almuhimedi, N. Atri, and L. F. Cranor, "Crying wolf: An empirical study of ssl warning effectiveness," in *Proc. of USENIX Security Symposium*, 2009. [cited at p. 8]

[37] A. P. Felt, E. Ha, S. Egelman, A. Haney, E. Chin, and D. Wagner, "Android permissions: User attention, comprehension, and behavior," in *Proc. of SOUPS*, 2012. [cited at p. 8]

[38] A. Bianchi, Y. Fratantonio, C. Kruegel, and G. Vigna, "Njas: Sandboxing unmodified applications in non-rooted devices running stock android," in *Proc. of SPSM*, 2015. [cited at p. 9, 10]

[39] M. Backes, S. Bugiel, C. Hammer, O. Schranz, and P. von Styp-Rekowsky, "Boxify: Full-fledged app sandboxing for stock android." in *Proc. of USENIX Security Symposium*, 2015. [cited at p. 9, 10, 27]

[40] S. Jaebaek, K. Daehyeok, C. Donghyun, S. Insik, and K. Taesoo, "FLEXDROID: enforcing in-app privilege separation in android," in *Proc. of NDSS*, 2016. [cited at p. 10, 29]

[41] M. Sun and G. Tan, "Nativeguard: Protecting android applications from third-party native libraries," in *Proc. of ACM WiSec*, 2014. [cited at p. 10]

[42] E. Malmi and I. Weber, "You are what apps you use: Demographic prediction based on user's apps," in *Proc. of ICWSM*, 2016. [cited at p. 10]

[43] S. Dai, A. Tongaonkar, X. Wang, A. Nucci, and D. Song, "Networkprofiler: Towards automatic fingerprinting of android apps," in *Proc. of IEEE INFOCOM*, 2013. [cited at p. 10]

[44] Q. Xu, Y. Liao, S. Miskovic, Z. M. Mao, M. Baldi, A. Nucci, and T. Andrews, "Automatic generation of mobile app signatures from traffic observations," in *Proc. of IEEE INFOCOM*, 2015. [cited at p. 10, 12]

[45] V. F. Taylor, R. Spolaor, M. Conti, and I. Martinovic, "Appscanner: Automatic fingerprinting of smartphone apps from encrypted network traffic," in *Proc. of IEEE EuroS&P*, 2016. [cited at p. 10]

[46] ——, "Robust smartphone app identification via encrypted network traffic analysis," *IEEE Trans. on Information Forensics and Security*, vol. 13, no. 1, pp. 63–78, Jan 2018. [cited at p. 10, 12]

[47] G. G. Gulyás, G. Acs, and C. Castelluccia, "Near-optimal fingerprinting with constraints," *Proc. of PoPETs*, 2016. [cited at p. 10]

[48] J. Huang, O. Schranz, S. Bugiel, and M. Backes, "The art of app compartmentalization: Compiler-based library privilege separation on stock android," in *Proc. of ACM CCS*, 2017. [cited at p. 10]

[49] "Permissions Overview," https://developer.android.com/guide/topics/permissions/overview, last visited: Sep. 2018. [cited at p. 11]

[50] "Android Debug Bridge (adb)," https://developer.android.com/studio/command-line/adb.html, last visited: Sep. 2018. [cited at p. 11, 28]

[51] V. F. Taylor, R. Spolaor, M. Conti, and I. Martinovic, "Appscanner: Automatic fingerprinting of smartphone apps from encrypted network traffic," in *Proc. of EuroS&P*. IEEE, 2016, pp. 439–454. [cited at p. 12]

[52] Q. A. Chen, Z. Qian, and Z. M. Mao, "Peeking into Your App without Actually Seeing It: UI State Inference and Novel Android Attacks." in *Proc. of USENIX Security Symposium*, 2014, pp. 1037–1052. [cited at p. 13]

[53] S. Jana and V. Shmatikov, "Memento: Learning secrets from process footprints," in *Proc. of S&P Symposium*. IEEE, 2012, pp. 143–157. [cited at p. 13]

[54] Y. Chen, X. Jin, J. Sun, R. Zhang, and Y. Zhang, "Powerful: Mobile app fingerprinting via power analysis," in *Proc. of IEEE INFOCOM*. IEEE, 2017, pp. 1–9. [cited at p. 13]

[55] C.-C. Lin, H. Li, X.-y. Zhou, and X. Wang, "Screenmilker: How to milk your android screen for secrets." in *Proc. of NDSS*, 2014. [cited at p. 13, 19]

[56] "Privacy, Security, and Deception," https://play.google.com/about/privacy-security-deception/personal-sensitive/, last visited: Sep. 2018. [cited at p. 14, 16]

[57] "A tool for reverse engineering Android apk files," https://ibotpeaches.github.io/Apktool/, last visited: Sep. 2018. [cited at p. 14]

[58] http://www.zdnet.com/article/accuweather-caught-sending-geo-location-data-even-when-denied-access/, 2017, last visited: Sep. 2017. [cited at p. 14]

[59] "XPrivacy," https://github.com/M66B/XPrivacy, last visited: Sep. 2018. [cited at p. 15]

[60] "Solitaire: Super Challenges," https://play.google.com/store/apps/details?id=com.cardgame.solitaire.full, last visited: Sep. 2018. [cited at p. 16]

[61] "DH Texas Poker - Texas Hold'em," https://play.google.com/store/apps/details?id=com.droidhen.game.poker, last visited: Sep. 2018. [cited at p. 16]

[62] "MX Player," https://play.google.com/store/apps/details?id=com.mxtech.videoplayer.ad, last visited: Sep. 2018. [cited at p. 17]

[63] "Sweet Selfie - selfie camera, beauty cam, photo edit," https://play.google.com/store/apps/details?id=com.cam001.selfie, last visited: Sep. 2018. [cited at p. 17]

[64] "InstaSize Editor: Photo Filters and Collage Maker," https://play.google.com/store/apps/details?id=com.jsdev.instasize, last visited: Sep. 2018. [cited at p. 17]

[65] "Angry Birds," https://play.google.com/store/apps/details?id=com.rovio.angrybirds, last visited: Sep. 2018. [cited at p. 17]

[66] "Neon Motocross," https://play.google.com/store/apps/details?id=com.motomex.neonmotocross, last visited: Sep. 2018. [cited at p. 17]

[67] "Supporting Multiple Users — Android Open Source Project," https://source.android.com/devices/tech/admin/multi-user, last visited: Sep. 2018. [cited at p. 17]

[68] "Put Android to work," https://www.android.com/enterprise/employees/, last visited: Sep. 2018. [cited at p. 18]

[69] "Android Instant Apps," https://developer.android.com/topic/instant-apps/index.html, last visited: Sep. 2018. [cited at p. 18]

[70] "Android Instant Apps API reference," https://developer.android.com/topic/instant-apps/reference.html#instantapps.InstantApps, last visited: Sep. 2018. [cited at p. 18]

[71] "Help protect against harmful apps with Google Play Protect," https://support.google.com/accounts/answer/2812853?hl=en, last visited: Sep. 2018. [cited at p. 18, 26]

[72] "Protecting against Security Threats with SafetyNet," https://developer.android.com/training/safetynet/index.html, last visited: Sep. 2018. [cited at p. 18]

[73] "Samsung Knox," https://www.samsungknox.com/en, last visited: Sep. 2018. [cited at p. 18]

[74] "Nova Launcher," https://play.google.com/store/apps/details?id=com.teslacoilsw.launcher&hl=en, last visited: Sep. 2018. [cited at p. 19]

[75] "Parallel Space - Multiple accounts and Two face," https://play.google.com/store/apps/details?id=com.lbe.parallel.intl&hl=en, last visited: Sep. 2018. [cited at p. 19]

[76] "Hide App, Private Dating, Safe Chat - PrivacyHider," https://play.google.com/store/apps/details?id=com.trigtech.privateme&hl=en, last visited: Sep. 2018. [cited at p. 19]

[77] "Private Zone - Safe Vault," https://play.google.com/store/apps/details?id=com.leo.appmaster, last visited: Sep. 2018. [cited at p. 19]

[78] "Amazon App Store," https://www.amazon.com/mobile-apps/b?ie=UTF8&node=2350149011, last visited: Sep. 2018. [cited at p. 19]

[79] "F-Droid," https://f-droid.org/en/, last visited: Sep. 2018. [cited at p. 19]

[80] "Publish Your App," https://developer.android.com/studio/publish/index.html#publishing-unknown, last visited: Sep. 2018. [cited at p. 19, 26]

[81] "DroidPlugin," https://github.com/DroidPluginTeam/DroidPlugin, last visited: Sep. 2018. [cited at p. 22, 27]

[82] "The Design and Implementation of the Tor Browser," https://www.torproject.org/projects/torbrowser/design/, last visited: Sep. 2018. [cited at p. 25]

[83] D. C. Nguyen, D. Wermke, Y. Acar, M. Backes, C. Weir, and S. Fahl, "A stitch in time: Supporting android developers in writingsecure code," in *Proc. of ACM CCS*. ACM, 2017, pp. 1065–1077. [cited at p. 25, 32]

[84] E. Fernandes, J. Paupore, A. Rahmati, D. Simionato, M. Conti, and A. Prakash, "Flowfence: Practical data protection for emerging iot application frameworks." in *Proc. of USENIX Security Symposium*, 2016, pp. 531–548. [cited at p. 25, 32]

[85] "Beurer HealthManager," https://play.google.com/store/apps/details?id=com.beurer.connect.healthmanager, last visited: Sep. 2018. [cited at p. 26]

[86] "Cancer.Net Mobile," https://play.google.com/store/apps/details?id=com.fueled.cancernet, last visited: Sep. 2018. [cited at p. 26]

[87] "Launch-Time Performance," https://developer.android.com/topic/performance/launch-time.html, last visited: Sep. 2018. [cited at p. 28, 29]

[88] https://github.com/commonsguy/cw-omnibus/tree/master/Activities/Lifecycle, last visited: Sep. 2018. [cited at p. 28]

[89] X. Wang, K. Sun, Y. Wang, and J. Jing, "Deepdroid: Dynamically enforcing enterprise policy on android devices." in *Proc. of NDSS*, 2015. [cited at p. 29]

[90] "SQLite via SQLiteOpenHelper," https://github.com/commonsguy/cw-omnibus/tree/master/Database/ConstantsROWID, last visited: Sep. 2018. [cited at p. 30]

[91] N. K. Malhotra, S. S. Kim, and J. Agarwal, "Internet users' information privacy concerns (iuipc): The construct, the scale, and a causal model," *Information systems research*, vol. 15, no. 4, pp. 336–355, 2004. [cited at p. 30]

[92] https://www.dropbox.com/sh/lo273jtx6jkbf1c/AAB1BtkBmBuNVOV13OAwDu-ha?dl=1, last visited: Sep. 2018. [cited at p. 31]

[93] https://www.washingtonpost.com/news/worldviews/wp/2014/09/12/could-using-gay-dating-app-grindr-get-you-arrested-in-egypt/?noredirect=on&utm_term=.470e8bc8f41c, last visited: Sep. 2018. [cited at p. 32]

[94] http://uk.businessinsider.com/despite-its-problems-uber-is-still-the-safest-way-to-order-a-taxi-2014-12?r=US&IR=T, last visited: Sep. 2018. [cited at p. 33, 34]

[95] http://www.dailydot.com/via/uber-lyft-safety-background-checks/, last visited: Sep. 2018. [cited at p. 33, 34]

[96] http://www.forbes.com/sites/kashmirhill/2014/10/03/god-view-uber-allegedly-stalked-users-for-party-goers-viewing-pleasure/, last visited: Sep. 2018. [cited at p. 33, 37, 38, 96]

[97] http://thenextweb.com/insider/2016/12/13/uber-tracks-customers-long-after-their-ride-is-over/. Last visited: Sep. 2018. [cited at p. 33]

[98] http://www.ibtimes.co.uk/former-uber-employee-reveals-drivers-used-tracking-technology-stalk-celebrities-politicians-1596263, last visited: Sep. 2018. [cited at p. 33, 40]

[99] http://www.usatoday.com/story/tech/2014/11/19/uber-privacy-tracking/19285481/, last visited: Sep. 2018. [cited at p. 33, 36, 38, 40]

[100] http://www.theverge.com/2015/6/14/8778111/uber-threatens-to-fire-drivers-attending-protests-in-china, last visited: Jan. 2017. [cited at p. 34, 37, 40]

[101] https://research.neustar.biz/2014/09/15/riding-with-the-stars-passenger-privacy-in-the-nyc-taxicab-dataset/, last visited: Sep. 2017. [cited at p. 34, 40]

[102] J. Fan and F. Vercauteren, "Somewhat Practical Fully Homomorphic Encryption," Cryptology ePrint Archive, Report 2012/144, 2012, http://eprint.iacr.org/2012/144. [cited at p. 34, 43, 54, 57]

[103] A. Pedrouzo-Ulloa, J. R. Troncoso-Pastoriza, and F. Perez-Gonzalez, "Number theoretic transforms for secure signal processing," *Trans. on Information Forensics and Security*, 2017. [cited at p. 34, 45]

[104] https://github.com/toddwschneider/nyc-taxi-data. Last visited: Sep. 2018. [cited at p. 34, 56]

[105] http://oride.epfl.ch. [cited at p. 35, 57]

[106] www.bostonglobe.com/business/2015/01/13/uber-share-ridership-data-with-boston/4Klo40KZREtQ7jkoaZjoNN/story.html, last visited: Sep. 2018. [cited at p. 35]

[107] http://www.forbes.com/sites/ronhirson/2015/03/23/uber-the-big-data-company/, last visited: Sep. 2018. [cited at p. 35]

[108] http://thehub.lyft.com/blog/2014/10/15/cancelations-join-acceptance-rate-equation, last visited: Sep. 2018. [cited at p. 36, 37, 39]

[109] http://www.dailydot.com/technology/uber-female-driver-harassment/, last visited: Sep. 2018. [cited at p. 37, 96]

[110] https://gigaom.com/2014/11/21/if-youre-worried-about-uber-and-privacy-dont-forget-lyft-and-sidecar/, last visited: Sep. 2018. [cited at p. 37, 96]

[111] http://www.newsweek.com/uber-taxi-e-hailing-riding-app-travis-kalanick-emil-michael-josh-mohrer-uber-285642, last visited: Sep. 2018. [cited at p. 37]

[112] http://www.reuters.com/article/uber-tech-lyft-probe-exclusive-idUSKBN0U12FH20151219, last visited: Sep. 2018. [cited at p. 37, 96]

[113] http://jetsettershomestead.boardingarea.com/2015/01/08/ways-passengers-can-cheat-uber/, last visited: Sep. 2018. [cited at p. 37, 96]

[114] http://www.businessinsider.com/blake-jareds-50000-uber-credit-free-rides-for-life-2014-4, last visited: Sep. 2018. [cited at p. 37, 96]

[115] http://wspa.com/2016/01/18/uber-driver-off-the-job-after-he-charged-for-fake-puke-2/, last visited: Sep. 2018. [cited at p. 37, 96]

[116] http://sfist.com/2014/07/30/uber_still_illegally_working_sfo_al.php, last visited: Sep. 2018. [cited at p. 37, 96]

[117] http://www.bloomberg.com/news/articles/2015-06-28/one-driver-explains-how-he-is-helping-to-rip-off-uber-in-china, last visited: Sep. 2018. [cited at p. 37, 38, 96]

[118] http://fortune.com/2015/03/30/uber-stolen-account-credentials-alphabay/, last visited: Sep. 2018. [cited at p. 37, 96]

[119] http://www.theguardian.com/technology/2015/may/23/us-investigates-phantom-cab-rides-on-british-uber-accounts, last visited: Sep. 2018. [cited at p. 37]

[120] www.owasp.org/index.php/OWASP_Risk_Rating_Methodology, last visited: Sep. 2018. [cited at p. 37]

[121] http://therideshareguy.com/uber-deactivated-a-bunch-of-drivers-as-an-intimidation-tactic/, last visited: Sep. 2018. [cited at p. 37, 39]

[122] R. A. Popa, C. M. S. Redfield, N. Zeldovich, and H. Balakrishnan, "Cryptdb: Protecting confidentiality with encrypted query processing," in *Proc. of ACM SOSP*, 2011. [cited at p. 37]

[123] M. Naveed, S. Kamara, and C. V. Wright, "Inference attacks on property-preserving encrypted databases," in *Proc. of ACM CCS*, 2015. [cited at p. 37]

[124] http://toddwschneider.com/posts/analyzing-1-1-billion-nyc-taxi-and-uber-trips-with-a-vengeance/, last visited: Sep. 2018. [cited at p. 38]

[125] http://research.neustar.biz/2014/09/15/riding-with-the-stars-passenger-privacy-in-the-nyc-taxicab-dataset, last visited: Sep. 2018. [cited at p. 38]

[126] http://newsroom.uber.com/2014/09/inferring-uber-rider-destinations/, last visited: Sep. 2018. [cited at p. 38]

[127] http://www.cnet.com/news/taxi-dispute-gets-physical-in-france-with-attack-on-uber-car/, last visited: Sep. 2018. [cited at p. 38]

[128] http://www.thewire.com/technology/2014/08/uber-accused-of-booking-thousands-of-fake-rides-with-rival-lyft/375936/, last visited: Sep. 2018. [cited at p. 38, 39, 96]

[129] http://money.cnn.com/2014/08/11/technology/uber-fake-ride-requests-lyft/, last visited: Sep. 2018. [cited at p. 38, 39, 96]

[130] https://news.yahoo.com/warning-uber-account-might-sale-black-market-164144470.html, last visited: Sep. 2018. [cited at p. 38]

[131] http://www.bbc.com/news/business-35888352, last visited: Sep. 2018. [cited at p. 39]

[132] http://developer.android.com/guide/topics/location/strategies.html#MockData, last visited: Sep. 2018. [cited at p. 39]

[133] http://www.droidviews.com/how-to-hide-root-access-from-apps-that-detect-root-on-android/, last visited: Sep. 2018. [cited at p. 39]

[134] http://onhax.net/disable-jailbreak-detection, last visited: Sep. 2018. [cited at p. 39]

[135] U. M. Aïvodji, S. Gambs, M.-J. Huguet, and M.-O. Killijian, "Meeting points in ridesharing: A privacy-preserving approach," *Transportation Research Part C: Emerging Technologies*, vol. 72, pp. 239–253, 2016. [cited at p. 39]

[136] C. Dai, X. Yuan, and C. Wang, "Privacy-preserving ridesharing recommendation in geosocial networks," in *Proc. of CSoNet*, 2016. [cited at p. 39]

[137] P. Goel, L. Kulik, and K. Ramamohanarao, "Privacy-aware dynamic ride sharing," *Trans. on Spatial Algorithms and Systems*, 2016. [cited at p. 39]

[138] D. Sánchez, S. Martínez, and J. Domingo-Ferrer, "Co-utile P2P ridesharing via decentralization and reputation management," *Transportation Research Part C: Emerging Technologies*, 2016. [cited at p. 39]

[139] P. Goel, L. Kulik, and K. Ramamohanarao, "Optimal pick up point selection for effective ride sharing," *IEEE Trans. on Big Data*, 2016. [cited at p. 39]

[140] A. P. Isern-Deyà, A. Vives-Guasch, M. Mut-Puigserver, M. Payeras-Capellà, and J. Castellà-Roca, "A secure automatic fare collection system for time-based or distance-based services with revocable anonymity for users," *The Computer Journal*, 2013. [cited at p. 39]

[141] M. Milutinovic, K. Decroix, V. Naessens, and B. De Decker, "Privacy-preserving public transport ticketing system," in *Proc. of ACM CODASPY*, 2015. [cited at p. 39]

[142] G. Arfaoui, J.-F. Lalande, J. Traoré, N. Desmoulins, P. Berthomé, and S. Gharout, "A Practical Set-Membership Proof for Privacy-Preserving NFC Mobile Ticketing," *Proc. of PoPETs*, 2015. [cited at p. 39]

[143] http://www.wcnc.com/news/crime/uber-driver-attacked-rider-over-politics-man-says/339458660, last visited: Sep. 2018. [cited at p. 40]

[144] http://www.orlandosentinel.com/news/breaking-news/os-lyft-driver-arrest-sex-battery-20160929-story.html, last visited: Sep. 2018. [cited at p. 40]

[145] http://wfla.com/2016/12/27/uber-and-lyft-drivers-worry-about-passenger-attacks/, last visited: Sep. 2018. [cited at p. 40]

[146] https://www.uwgb.edu/dutchs/FieldMethods/UTMSystem.htm, last visited: Sep. 2018. [cited at p. 41]

[147] D. Chaum, "Blind signatures for untraceable payments," in *Proc. of the 3rd Cryptology Conference*, 1983. [cited at p. 42, 46, 53]

[148] F. Baldimtsi and A. Lysyanskaya, "Anonymous credentials light," in *Proc. of ACM CCS*, 2013. [cited at p. 42, 46, 54]

[149] V. Lyubashevsky, C. Peikert, and O. Regev, "On Ideal Lattices and Learning with Errors Over Rings," Cryptology ePrint Archive, Report 2012/230, 2012, http://eprint.iacr.org/2012/230. [cited at p. 43]

[150] https://techcrunch.com/2016/11/29/just-like-uber-lyft-launches-upfront-fares/, last visited: Sep. 2018. [cited at p. 45, 49]

[151] https://newsroom.uber.com/philippines/new-upfront-fares-on-uberx/, last visited: Sep. 2018. [cited at p. 45, 49]

[152] http://www.usatoday.com/story/tech/columnist/stevenpetrow/2016/10/12/fake-uber-drivers-dont-become-next-victim/91903508/, last visited: Sep. 2018. [cited at p. 49]

[153] https://newsroom.uber.com/updated-cancellation-policy/. Last visited: Sep. 2018. [cited at p. 50]

[154] P. Feldman, "A practical scheme for non-interactive verifiable secret sharing," in *Proc. of FOCS*, 1987. [cited at p. 51]

[155] https://support.google.com/trustedcontacts/?hl=en. Last visited: Sep. 2018. [cited at p. 51]

[156] http://mathworld.wolfram.com/VoronoiDiagram.html. [cited at p. 53]

[157] B. Laurie, A. Langley, and E. Kasper, "Certificate transparency," https://tools.ietf.org/html/rfc6962, 2013. [cited at p. 53]

[158] https://www.bloomberg.com/view/articles/2016-09-09/wells-fargo-opened-a-couple-million-fake-accounts, last visited: Sep. 2018. [cited at p. 53]

[159] http://toddwschneider.com/posts/analyzing-1-1-billion-nyc-taxi-and-uber-trips-with-a-vengeance/, last visited: Sep. 2018. [cited at p. 55]

[160] C. A. Melchor, J. Barrier, S. Guelton, A. Guinet, M. Killijian, and T. Lepoint, "NFLlib: NTT-Based Fast Lattice Library," in *Proc. of the RSA conference - The Cryptographers' Track*, 2016. [cited at p. 56]

[161] "Recommendation for Key Management, Part 1: General, SP 800-57 Part 1 Rev. 4," Online: http://dx.doi.org/10.6028/NIST.SP.800-57pt1r4, January 2016, last visited: Sep. 2018. [cited at p. 57]

[162] M. R. Albrecht, "On dual lattice attacks against small-secret LWE and parameter choices in HElib and SEAL," Cryptology ePrint Archive, Report 2017/047, 2017, http://eprint.iacr.org/2017/047. [cited at p. 57]

[163] http://browser.primatelabs.com/android-benchmarks, last visited: Sep. 2018. [cited at p. 59]

[164] http://browser.primatelabs.com/processor-benchmarks, last visited: Sep. 2018. [cited at p. 59]

[165] www.forbes.com/sites/ellenhuet/2014/09/08/uber-lyft-cars-arrive-faster-than-taxis/#3f819c3c5f73, last visited: Sep. 2018. [cited at p. 59]

[166] https://www.nerdwallet.com/blog/utilities/best-unlimited-data-plans/, last visited: Sep. 2018. [cited at p. 60]

[167] https://www.link.nyc, last visited: Sep. 2018. [cited at p. 60]

[168] https://developers.google.com/maps/documentation/distance-matrix/. Last visited: Sep. 2018. [cited at p. 61]

[169] "Collect points when running with the Helsana Trails App," http://www.helsana.ch/docs/Quick-guide-collect-points-when-running-iphone.pdf, 2014, last visited: Sep. 2018. [cited at p. 64]

[170] "Achievemint," http://www.achievemint.com. [cited at p. 64]

[171] "Higi," https://higi.com/. [cited at p. 64]

[172] "Fitstudio," https://www.fitstudio.com/, 2015. [cited at p. 64]

[173] "Oscar Health Using Misfit Wearables To Reward Fit Customers," http://www.forbes.com/sites/stevenbertoni/2014/12/08/oscar-health-using-misfit-wearables-to-reward-fit-customers/, 2014, last visited: Sep. 2018. [cited at p. 64]

[174] "Swisscom ski cup," http://www.swisscom.ch/en/about/medien/press-releases/2013/10/20131028_MM_Swisscom_Snow_Cup.html, last visited: Feb. 2018. [cited at p. 64]

[175] "Nike+ badges and trophies," http://www.garcard.com/nikeplus.php, last visited: Feb. 2018. [cited at p. 64]

[176] "Wear this device so the boss knows you are losing weight," http://www.bloomberg.com/news/2014-08-21/wear-this-device-so-the-boss-knows-you-re-losing-weight.html, last visited: Sep. 2018. [cited at p. 64]

[177] "Wearable tech is plugging into health insurance," http://www.forbes.com/sites/parmyolson/2014/06/19/wearable-tech-health-insurance/, last visited: Sep. 2018. [cited at p. 64]

[178] "Your boss would like you to wear a jawbone fitness tracker," http://www.bloomberg.com/news/articles/2014-12-10/jawbone-up-for-groups-a-plan-to-get-employers-to-buy-fitness-bands, 2014, last visited: Sep. 2018. [cited at p. 64]

[179] "Health insurer's app helps users track themselves," http://www.technologyreview.com/news/516176/health-insurers-app-helps-users-track-themselves/, 2013, last visited: Sep. 2018. [cited at p. 64]

[180] "Health account and health account bonus," https://www.css.ch/en/home/privatpersonen/krankenkasse/zusatzversicherung/gesundheitskonto.html, last visited: Sep. 2018. [cited at p. 64]

[181] "Invest in healthy behavior change," https://healthsolutions.fitbit.com/employers/#commerce, last visited: Sep. 2018. [cited at p. 64]

[182] M. Gruteser and B. Hoh, "On the Anonymity of Periodic Location Samples," in *Proc. of PerCom*, 2005. [cited at p. 64]

[183] B. Hoh, M. Gruteser, H. Xiong, and A. Alrabady, "Enhancing Security and Privacy in Traffic-Monitoring Systems," *Trans. on Pervasive Computing*, vol. 5, 2006. [cited at p. 64]

[184] Y. Matsuo, N. Okazaki, K. Izumi, Y. Nakamura, T. Nishimura, and K. Hasida, "Inferring long-term user property based on users," in *Proc. of IJCAI*, 2007. [cited at p. 64]

[185] A. Noulas, M. Musolesi, M. Pontil, and C. Mascolo, "Inferring interests from mobility and social interactions," in *Proc. of NIPS Workshops*, 2009. [cited at p. 64]

[186] D. Crandall, L. Backstrom, D. Cosley, S. Suri, D. Huttenlocher, and J. Kleinberg, "Inferring social ties from geographic coincidences," *Proc. of PNAS*, vol. 107, 2010. [cited at p. 64]

[187] W. U. Hassan, S. Hussain, and A. Bates, "Analysis of privacy protections in fitness tracking social networks-or-you can run, but can you hide?" in *Proc. of USENIX Security Symposium*, 2018. [cited at p. 64, 67]

[188] B. Carbunar and R. Potharaju, "You unlocked the Mt. Everest badge on Foursquare! Countering location fraud in geosocial networks," in *Proc. of MASS*, 2012, pp. 182–190. [cited at p. 64, 66]

[189] M. Rahman, B. Carbunar, and U. Topkara, "A secure protocol for managing low power fitness trackers," *IEEE Trans. on Mobile Computing*, 2015, to appear. [cited at p. 64]

[190] W. He, X. Liu, and M. Ren, "Location cheating: A security challenge to location-based social network services," in *Proc. of ICDCS*, 2011. [cited at p. 64, 66, 68]

[191] K. Zhang, W. Jeng, F. Fofie, K. Pelechrinis, and P. Krishnamurthy, "Towards reliable spatial information in LBSNs." in *Proc. of ACM UbiComp*, 2012. [cited at p. 64, 66]

[192] "Fake Track, Simulate GPS Routes," https://play.google.com/store/apps/details?id=com.trinus.faketrack&hl=en. [cited at p. 64]

[193] "How to tell if someone used Digital Epo to cheat on Strava," http://www.scarletfire.co.uk/how-to-tell-if-someone-used-digital-epo-to-cheat-on-strava/, last visited: Sep. 2018. [cited at p. 64]

[194] "Digitalepo," http://www.digitalepo.com/. [cited at p. 64]

[195] "Fitbit cheat-o-matic," https://www.youtube.com/watch?v=3rrRFW0j5Vk, last visited: Sep. 2018. [cited at p. 64]

[196] J. Brassil and P. K. Manadhata, "Proving the location of a mobile device user," in *2012 Virgina Tech Wireless Symposium*, 2012. [cited at p. 64]

[197] W. Luo and U. Hengartner, "Veriplace: A privacy-aware location proof architecture," in *Proc. of GIS*, 2010. [cited at p. 64, 66]

[198] B. Carbunar, R. Sion, R. Potharaju, and M. Ehsan, "Private badges for geosocial networks," *IEEE Trans. on Mobile Computing*, vol. 13, 2014. [cited at p. 64, 66]

[199] "Garmin connect," http://connect.garmin.com. [cited at p. 65, 78]

[200] "FON," https://corp.fon.com/en. [cited at p. 65, 78]

[201] I. Polakis, S. Volanis, E. Athanasopoulos, and E. P. Markatos, "The man who was there: validating check-ins in location-based services," in *Proc. of ACM ACSAC*, 2013. [cited at p. 66]

[202] M. Talasila, R. Curtmola, and C. Borcea, "Link: Location verification through immediate neighbors knowledge," in *Proc. of MOBIQUITOUS*, 2012. [cited at p. 66]

[203] Z. Zhu and G. Cao, "Toward privacy preserving and collusion resistance in a location proof updating system," *IEEE Trans. on Mobile Computing*, vol. 12, 2013. [cited at p. 66]

[204] S. Gambs, M.-O. Killijian, M. Roy, and M. Traoré, "PROPS: A privacy-preserving location proof system," in *Proc. of IEEE SRDS*, 2014. [cited at p. 66]

[205] M. Jadliwala, S. Zhong, S. Upadhyaya, C. Qiao, and J.-P. Hubaux, "Secure distance-based localization in the presence of cheating beacon nodes," *IEEE Trans. on Mobile Computing*, vol. 9, no. 6, pp. 810–823, 2010. [cited at p. 66]

[206] L. Hu and D. Evans, "Localization for mobile sensor networks," in *Proc. of ACM MobiCom*, 2004. [cited at p. 66]

[207] J.-P. Sheu, W.-K. Hu, and J.-C. Lin, "Distributed localization scheme for mobile sensor networks," *IEEE Trans. on Mobile Computing*, vol. 9, 2010. [cited at p. 66]

[208] S. Saroiu and A. Wolman, "Enabling new mobile applications with location proofs," in *Proc. of HotMobile*, 2009. [cited at p. 66]

[209] D. Singelee and B. Preneel, "Location verification using secure distance bounding protocols," in *Proc. of IEEE MASS*, 2005. [cited at p. 66]

[210] J. T. Chiang, J. J. Haas, and Y.-C. Hu, "Secure and precise location verification using distance bounding and simultaneous multilateration," in *Proc. of ACM WiSec*, 2009. [cited at p. 66]

[211] S. Capkun and J.-P. Hubaux, "Secure positioning of wireless devices with application to sensor networks," in *Proc. of INFOCOM*, vol. 3, 2005. [cited at p. 66]

[212] J. Reid, J. M. G. Nieto, T. Tang, and B. Senadji, "Detecting relay attacks with timing-based protocols," in *Proc. of ACM AsiaCCS*, 2007. [cited at p. 66]

[213] A. Uchiyama, S. Fujii, K. Maeda, T. Umedu, H. Yamaguchi, and T. Higashino, "Upl: Opportunistic localization in urban districts," *IEEE Trans. on Mobile Computing*, vol. 12, no. 5, 2013. [cited at p. 67]

[214] J. Koo, J. Yi, and H. Cha, "Localization in mobile ad hoc networks using cumulative route information," in *Proc. of ACM UbiComp*, 2008. [cited at p. 67]

[215] "NTP: The Network Time Protocol," http://www.ntp.org/, last visited: Sep. 2018. [cited at p. 69]

[216] T. Kohno, A. Broido, and K. C. Claffy, "Remote physical device fingerprinting," in *Proc. of IEEE S&P*, 2005. [cited at p. 69]

[217] D. B. Faria and D. R. Cheriton, "Detecting identity-based attacks in wireless networks using signalprints," in *Proc. of ACM WiSec*, 2006. [cited at p. 69]

[218] L. C. C. Desmond, C. C. Yuan, T. C. Pheng, and R. S. Lee, "Identifying unique devices through wireless fingerprinting," in *Proc. of ACM WiSec*, 2008. [cited at p. 69]

[219] "Interior Point OPTimizer," https://projects.coin-or.org/Ipopt. [cited at p. 72]

[220] "Freewifi," http://www.free.fr/adsl/pages/internet/connexion/acces-hotspot-wifiFree.html, last visited: Sep. 2018. [cited at p. 78]

[221] A. Pham, K. Huguenin, I. Bilogrevic, and J.-P. Hubaux, "Secure and Private Proofs for Location-Based Activity Summaries in Urban Areas," in *Proc. of ACM UbiComp*, 2014. [cited at p. 82]

[222] O. Goldreich, B. Pfitzman, and R. Rivest, "Self-delegation with controlled propagation—or—what if you lose your laptop," in *Proc. of CRYPTO*, 1998. [cited at p. 87]

[223] J. Camenisch and A. Lysyanskaya, "An efficient system for non-transferable anonymous credentials with optional anonymity revocation," in *Proc. of EUROCRYPT*, 2001. [cited at p. 87]

[224] A. R. Beresford and F. Stajano, "Mix zones: User privacy in location-aware services," in *Proc. of PerCOM Workshops*, 2004, p. 127. [cited at p. 88]

[225] L. Buttyán, T. Holczer, and I. Vajda, "On the effectiveness of changing pseudonyms to provide location privacy in vanets," in *Proc. of ESAS Workshops*, 2007. [cited at p. 88]

[226] C. Diaz, S. Seys, J. Claessens, and B. Preneel, "Towards measuring anonymity," in *Proc. of PoPETs*, 2003. [cited at p. 88]

[227] L. Bindschaedler, M. Jadliwala, I. Bilogrevic, I. Aad, P. Ginzboorg, V. Niemi, and J.-P. Hubaux, "Track me if you can: On the effectiveness of context-based identifier changes in deployed mobile networks." in *Proc. of IEEE NDSS*, 2012. [cited at p. 88]

[228] "Waymo takes the final step before launching its self-driving car service," https://www.wired.com/story/waymo-self-driving-car-service-phoenix/, last visited: Sep. 2018. [cited at p. 92]

[229] D. He, M. Naveed, C. A. Gunter, and K. Nahrstedt, "Security concerns in android mhealth apps," in *Proc. of AMIA Annual Symposium*, vol. 2014. American Medical Informatics Association, 2014, p. 645. [cited at p. 92]

Index

Anh Pham
12 Avenue de l'Eglise Anglaise
1006 Lausanne
+41 78 813 4474
vananhpham88@gmail.com
https://www.linkedin.com/in/vananhpham/

STRENGTHS

- Experience with building secure and privacy-preserving systems
- Experience with programming, data structures and algorithms
- Experience with leading and managing small teams of undergraduate students
- Eagerness to learn and willingness to explore new topics
- Commitment and teamwork

EDUCATION

École Polytechnique Fédérale de Lausanne (EPFL) Lausanne, Switzerland
Ph.D. in Computer Science 2013-2018
Thesis: Privacy-Enhancing Technologies for Mobile Applications and Services
Advisors: Prof. Jean-Pierre Hubaux and Prof. Kévin Huguenin

KTH Royal Institute of Technology Stockholm, Sweden
Aalto University Espoo, Finland
M.Sc. in Mobile Computing and Network Security 2011-2013
Thesis: Security of NFC Applications
Advisor: Prof. Tuomas Aura
Graduated with Distinction – Top 2%

Vietnam National University (VNU) Hanoi, Vietnam
B.Sc. in Computer Science 2006-2010
Graduated with Distinction – Top 2%

EXPERIENCE

École Polytechnique Fédérale de Lausanne (EPFL) Lausanne, Switzerland
Graduate research assistant 2013-2018
Leader of three projects:

- A system for hiding the presence of sensitive apps on Android [1]
- Privacy-preserving ride-hailing services [2, 3]
- Privacy-preserving fitness-tracking applications [4, 5]

Vietnam National University Hanoi, Vietnam
Teaching staff 2010-2011
Lectures and exercise sessions for students of two courses: Information security and Introduction to programming (using C++).

SKILLS

- Good understanding and experience with security-protocol design
- Good understanding and experience with the Android platform
- Programming skills: proficient with Java; familiar with C++, Python, SQL; previously used MATLAB, bash
- Experience with conducting user studies
- Knowledge about computer and mobile networks
- Experience with leading and managing small teams of undergrad students

AWARDS

One-year fellowship from the EPFL doctoral program in computer science	2013
Erasmus Mundus scholarship for the entirety of my master studies	2011-2013
IBM scholarship for the most excellent students in Vietnam	2009
Reward for outstanding female students in IT, from the Vietnamese government	2008
VNU scholarship for the entirety of my bachelor studies	2006-2010

PUBLICATIONS

[1] **A. Pham**, I. Dacosta, E. Losiouk, J. Stephan, K. Huguenin, and J.-P. Hubaux. *HideMyApp: Hiding the Presence of Sensitive Apps on Android*. Under submission.

[2] **A. Pham**, I. Dacosta, G. Endignoux, J. R. Troncoso-Pastoriza, K. Huguenin, and J.-P. Hubaux. *ORide: A Privacy-Preserving yet Accountable Ride- Hailing Service*. Proc. of the 26th USENIX Security Symposium, 2017 (acceptance rate: 16.3%).

[3] **A. Pham**, I. Dacosta, B. Jacot-Guillarmod, K. Huguenin and T. Hajar, F. Tramer, V.Gligor and J.-P. Hubaux. *PrivateRide: A Privacy-Enhanced Ride-Hailing Service*. Proc. of the 17th Privacy Enhancing Technologies Symposium (PoPETS), 2017 (acceptance rate: 22.5%).

[4] **A. Pham**, K. Huguenin, I. Bilogrevic, I. Dacosta, and J.-P. Hubaux. *SecureRun: Cheat-Proof and Private Summaries for Location-Based Activities*, in IEEE Transactions on Mobile Computing, Volume: 15 , Issue: 8, 2016.

[5] **A. Pham**, K. Huguenin, I. Bilogrevic and J.-P. Hubaux. *Secure and Private Proofs for Location-Based Activity Summaries in Urban Areas.* Proc. of the ACM International Joint Conference on Pervasive and Ubiquitous Computing (UbiComp), 2014 (acceptance rate: 16.5%).

PATENTS

A. Pham, I. Dacosta, J.-P. Hubaux, K. Huguenin. *Method and system for hiding the presence of sensitive apps in mobile devices*. US Provisional Patent Application n° 62/693,948, filed on 04/07/ 2018.

LANGUAGES

- English: Full professional proficiency
- Vietnamese: Native proficiency

PERSONAL INFORMATION

30 years old, female, unmarried, Vietnamese.

OTHERS

Media coverage about my research: WIRED.com, ComputerWorld.com, ibtimes.com, Engadget.com

**Ce document a été imprimé au Centre d'impression EPFL,
imprimerie climatiquement neutre, certifiée myClimate.**