Security and Privacy for Ubiquitous Mobile Devices

Edmund James Novak

The College of William and Mary
Williamsburg, VA

Bachelor of the Arts, Monmouth College, Monmouth, IL, 2010
Master of Science, The College of William and Mary, Williamsburg, VA 2012

A Dissertation presented to the Graduate Faculty
of the College of William and Mary in Candidacy for the Degree of
Doctor of Philosophy

Department of Computer Science

The College of William and Mary
August 2016

APPROVAL PAGE

This Dissertation is submitted in partial fulfillment of
the requirements for the degree of

Doctor of Philosophy

[illegible]

Edmund James Novak

Approved by the Committee, June 2016

[illegible]

Committee Chair
Professor Qun Li, Computer Science
The College of William and Mary

Weizhen Mao

Professor Weizhen Mao, Computer Science
The College of William and Mary

[illegible]

Professor Denys Poshyvanyk, Computer Science
The College of William and Mary

[illegible]

Professor Haining Wang, Electrical and Computer Engineering
University of Delaware

[illegible]

Professor Mark Hinders, Applied Science
The College of William and Mary

ABSTRACT

We live in a world where mobile devices are already ubiquitous. It is estimated that in the United States approximately two thirds of adults own a smartphone, and that for many, these devices are their primary method of accessing the Internet [99]. World wide, it is estimated that in May of 2014 there were 6.9 billion mobile cellular subscriptions, almost as much as the world population. Of these 6.9 billion, approximately 1 billion are smart devices, which are concentrated in the developed world. In the developing world, users are moving from feature phones to smart devices as a result of lower prices and marketing efforts [30]. Because smart mobile devices are ubiquitous, users trust them with an unprecedented amount of information. This makes security and privacy primary concerns. Threats such as mobile malware are already substantial, with over 2500 different types identified in 2010 alone [65]. It is likely that, as the smart device market continues to grow, so to will concerns about privacy, security, and malicious software. This is especially true, because these mobile devices are relatively new. Our research focuses on increasing the security and privacy of user data on smart mobile devices. We propose three applications in this domain: (1) a service that provides private, mobile location sharing; (2) a secure, intuitive proximity networking solution; and (3) a potential attack vector in mobile devices, which utilizes novel covert channels. We also propose a first step defense mechanism against these covert channels.

Our first project is the design and implementation of a service, which provides users with private and secure location sharing. This is useful for a variety of applications such as online dating, taxi cab services, and social networking. Our ultrasound modem, enables users to share relatively small pieces of information with those that are near by, also known as proximity based networking. It has several advantages over other, existing mechanisms for proximity based networking. Our third work exposes a novel attack vector utilizing physical media covert channels on smart devices, in conjunction with privilege escalation and confused deputy attacks. We show this attack preemptively and also propose a defense mechnanism to protect user data.

As a contribution to the field, we present these three systems, which together enrich the smart mobile experience, while providing mobile security and keeping privacy in mind. Our third approach specifically, presents a unique attack, which has not been seen ``in the wild'', in an effort to keep ahead of malicious efforts.

TABLE OF CONTENTS

ACKNOWLEDGMENTS

I would like to thank the numerous people who helped me in preparing this dissertation. Without their hard work, patience, and guidance it would not have been possible.
First, I would like to thank my advisor, Dr. Qun Li who provided me with an endless amount of guidance and support. Without his knowledge, expertise, and advice I would never have been able to even begin this work. Thank you also to my committee members Dr. Weizhen Mao, Dr. Denys Poshyvanyk, Dr. Haining Wang, and Dr. Mark Hinders of the Applied Science Department here at W&M. Their thoughtful feedback and critiques during my time here were much appreciated.
Next, I would like to thank my current and past research group members including Zhengrui Qin, Zijiang Hao, Yutao Tang, Shanhe Yi, Nancy Carter, Cheng Li, Yifan Zhang, Wei Wei, Hao Han, and Fengyuan Xu. It was a pleasure to collaborate and do research with each of them.
Lastly, I would like to thank our Computer Science Department Chair, Professor Robert Michael Lewis, and the wonderful Computer Science administration team, Vanessa Godwin, Jacqulyn Johnson, and Dale Hayes.

I would like to dedicate this dissertation to my parents, Tom and Cecelia Novak who provided endless support and love throughout my time at William and Mary.

LIST OF TABLES

LIST OF FIGURES

Security and Privacy for Ubiquitous Mobile Devices

Chapter 1

Introduction

Smart mobile devices have exploded in popularity since the introduction of the iPhone in 2007. We have now come to expect that smart mobile devices are commonplace in the modern world, and they are quickly growing in the developing world [30]. Users can choose from a variety of devices, including phones, watches, tablets, bracelets, and other wearable form factors running several different operating systems including Google's Android, Apple's iOS, and Microsoft's Windows 10 Mobile. Key to the success of mobile computing is the proliferation of the application (app) market paradigm, which provides developers and end-users a curated market place for software to run on their mobile devices. However, because the growth of these devices has been so fast, security and privacy have taken a back seat to convenience, innovation, and ease of use. Developers have created a wide variety of innovative applications, and because mobile devices are so practical and useful, users entrust a bounty of sensitive information to them, which is in jeopardy of being harvested by evildoers. Users trust their mobile devices explicitly and implicitly with many different types of sensitive information including passwords with password managers, their financial history through mobile banking, contact information for their friends and family, their location, web surfing habits, sensitive email, and sms messages just to name a few. Much research recently has shown that even more, implicit information can be extrapolated from these explicit sources [129, 17, 41, 53, 95, 108].

Indeed, it seems that smart devices are so useful and ubiquitous, there is hardly any sensitive information they aren't trusted with!

In this dissertation we present a three layer approach for mobile systems, which improves the security and privacy of sensitive user data in the modern mobile landscape at the physical layer, the application layer, and the operating system layer. Because mobile computing is at the center of several important crossroads, including social networking, privacy, energy efficiency, and ubiquity, these projects span a wide-range of research topics. Our three projects are: private and secure location sharing, an ultrasonic modem for proximity networking, and a preemptive demonstration of a novel attack vector utilizing physical media covert channels on mobile smart devices. In these projects we attempt to provide solutions to many important research challenges, and improve the overall topography of mobile computing. We feel that we provide a technical ``voice" to end-users, whose wants, needs, and concerns are often not well addressed in this area due to their lack of technical understanding.

1.1 Problems

In order to improve security and privacy in any domain, we must have an adequate definition for both. We define **security** as the technical mechanisms used to protect data. For example, encryption is used as a security mechanism to guarantee that only those with the appropriate keys (information) can access the protected data. Because encryption relies on some computationally intractable problem, we can safely assume that it is impossible for any attacker, with reasonable resources, to break the system and access the data. Security mechanisms can be complex, like encryption, which offers a variety of different tools. But they can also be very simple, such as physically isolating the system from any network access, or using physical locks to protect the hardware holding sensitive data. Designing systems that use several security mechanisms in conjunction with one another is non-trivial, and presents itself as a primary research challenge throughout this dissertation.

For security mechanisms to be useful, a sane policy must be crafted. In other words, there must be a set of rules to enforce, using security mechanisms. For example, a user may not want a social networking site provider to know their social security number. Conversely, the government needs to share social security numbers with their assignees, but not with other, potentially malicious, parties . Policies can be set by users, developers, service providers, or governments. However, each of these parties is likely to choose a policy that suits them best, without considering the needs of the others. We have seen this recently with the revelation of the United States government NSA spying programs [29, 44] and the San-Bernardino FBI--Apple encryption dispute [52]. Similarly to security, privacy policies are useless if they cannot be enforced. The two rely on each other fundamentally. Furthermore, designing policies on behalf of several parties with opposing perspectives is often intricately complex as different users' wants may conflict.

If a system provides satisfactory policies, and adequate enforcement, we say the system is **private** and **secure**. It is possible for a system to be only secure. This would mean that the fundamental technical mechanisms that protect the data are sound and cannot be broken by an attacker. However, as mentioned before, having a secure system that does not enforce sane policies is not useful, and likely is not meaningful. Indeed, many systems today may be considered secure, but not private. The canonical example is social networking, in which user data is highly secure from third party attackers, except that it is sold to advertisers. This policy may be considered satisfactory by the social networking site provider and the advertisers, but not by the end users. The notion of a system that is private, but not secure is meaningless. Privacy is built on top of security.

It is our hope that we can improve the state of security and privacy in the modern mobile landscape for end-users specifically, but in a way that is satisfactory by all involved. Unfortunately, systems that are considered private and secure by all are not easy to design or build. Each party has needs, and the systems must remain convenient and practical especially for end-users. It is unreasonable to expect users to hand craft elaborate policies, or to have deep technical understanding of the system. Systems that are secure and private, but even marginally less convenient will likely never be adopted. To summarize; our research improves the security and privacy of the mobile computing landscape at the physical layer, the operating system layer, and the application layer. We build systems that meet these four goals:

- **Privacy** - A sane set of rules, a policy, should be set forth that describes who has access to what data and in what context (i.e., time, place, relationship) in order to protect sensitive data from malicious parties. Policies may be government laws, rules enforced by the system itself, or may result organically (data that is not stored on a computer or publicly posted is inherently private).
- **Security** - Proper technical mechanisms should be used to enforce these policies. Popular mechanisms include encryption, information verification (such as checksums), and physical locks or barriers. Privacy is built on top of security.
- **Convenience** - Systems that may be secure and private will usually be passed by for systems that are convenient for end-users. It is unreasonable to expect users to craft elaborate policies, or to police the system manually. It is not unreasonable for users to expect a secure and private version of any existing service, with little or no sacrifices in convenience.
- **Mobility** - Our research focuses on mobile platforms. Mobile devices are already ubiquitous, with smart mobile devices continuing to gain ground, and they carry a plethora of sensitive information. It is clear that mobile is present and likely the future of personal computing for years to come.

In the current environment, end users maintain the vast majority of sensitive and valuable information. Additionally, they are not resilient against attacks, which can harm them personally. Governments and corporations on the other hand, are organizations that, by design, limit the liability of those personally involved. While we expect that all parties will act somewhat selfishly, putting their own best interests forward, our system focuses on providing systems that aid end-users most, which are currently at a disadvantage because they have little influence on the design of the systems they use, and lack technical knowledge to make properly informed decisions.

A good example of this disadvantage is the permission system used in the Android operating system. Users can install third party applications and, at install time, the user is presented with a list of special permissions the application requires to run. These permissions allow the application to access sensitive information sources such as the user's contact list, or hardware sensors. The user can choose to accept these permission requests, and use the application, or not. The user has little influence on the design of the system, and the granularity of their decision is far to coarse. Furthermore, it has become apparent that users seldom read or understand these permission system prompts.

1.2 Contributions

In this proposal we make three primary contributions, each of which makes several technical and research contributions in their own right. We present (1) a service for location sharing with location based access control to improve the application layer, (2) an ultrasound modem on common smart mobile devices for proximity networking at the physical layer, and (3) we preemptively expose a potential attack based on physical media covert channels. We also provide a defense mechanism at the operating system level. In the remainder of this section we elaborate on each of these three contributions.

Private Location Sharing - Ubiquitous mobile devices, along with extremely popular social media applications have spurred a recent surge in location based services, which leverage the precise locale of the user. Some examples of these applications include proximity based dating, cab hailing services, finding directions, and neighbor discovery based social networking. We focus on providing a specific, location based back-end service, while maintaining rigorous privacy and security. Specifically, we propose two different protocols for private proximity testing, which allow two users to determine if they are physically near one another, without revealing their actual, precise location to each other or any third party. Our protocols have two important features that are key to our contribution. Namely, they are purely distributed, only relying on third parties for blind message passing. We also are the only work, to the best of our knowledge, to allow the user fine grained control over what they consider to be ``physically near''. In private proximity detection, one user, Alice, wishes to learn if she is near a second user (Bob). Bob enforces a distance requirement P_b; Alice must be within P_b meters of him to be considered near him. Bob can alter his policy, P_b, arbitrarily between 10m and 160km, and he can specify different polices for different users or at different times of the day. In implementing our system we overcome several challenges including circumventing network address translation (NAT) to perform name resolution, fast and private proximity detection, and practicality concerns such as query rate. We implement both protocols in two different location based applications to demonstrate their usability. Both of our protocols are able to determine proximity in under 50 seconds for policies less than 160 kilometers with ten meter granularity. We believe that private and secure variations of mobile and social networking services are important for the future of mobile computing and can be supported by our private proximity detection protocols.

Ultrasound based Proximity Networking - Our second work proposes the use of ultrasound as a physical medium for proximity based networking on mobile devices. Proximity based networking is growing in popularity with mobile devices. It is obvious and intuitive to users that they should be able to share small pieces of information between their mobile devices when they are co-located. Examples include sharing URL links with an office mate, mobile payments at retail establishments, and sharing encryption keys to bootstrap private communication on a second channel. Proximity based networking also provides some primitive security, in that users must be physically co-located to exchange information. While there are several existing technologies for proximity based networking, including near field communication (NFC), bluetooth (BT), and WiFi Direct, none of them offer the same security and convenience as ultrasound. For example, ultrasound has an easy to configure range limitation, based on the device's existing volume controls. Additionally, it does not require any special hardware, because all mobile devices already carry speakers and microphones that support ultrasound. Our system faces several challenges including encoding and decoding information in a way that does create audible artifacts such as beating, and very precise synchronization to accurately recover phase information. By implementing this system, we provide a valuable and convenient mechanism for proximity based networking between virtually all mobile smart devices.

Physical Media Covert Channels - In an effort to protect the sensitive information that is stored on users' devices, we present a preemptive demonstration of a novel attack vector. Our attack utilizes a new type of covert channel based on physical media. For example, using the vibration motor as a sender and the accelerometer as a receiver. The attack runs as follows; an attacker first learns some sensitive information about a victim by running some code on that victim's smart device. Traditionally it is much more difficult to move this information off of the device, than it is to access it in the first place. The attacker encodes this sensitive information in vibration motor timings, and then recovers the data using the accelerometer, running in another application, to detect and decode the vibrations. The attacker can then send the information out over the Internet. This attack is dangerous because the second application can gain access to any information that is available to the first application, regardless of it's own permission allocations. We propose several novel covert channels, which are plentiful on smart devices due to their wide range of sensors and physical world hardware devices. By utilizing these covert channels, our attack circumvents the state of the art taint-tracking defense schemes to enable privilege escalation attacks and information leakage. We also improve the mobile operating system by proposing a defense mechanism against this attack. It is our hope that by proposing these new covert channels, and giving a first step defense, that more robust and secure defense mechanisms can be designed.

Chapter 2

Related Work

2.1 Location Privacy

Several papers propose methods for making location information private that can be applied to any location based service [87, 66, 13, 31]. These papers obfuscate location and temporal information, usually based on the k-anonymous measure. Although all of these systems intend for location based services to still function effectively, they all gradually erode service quality. As the user's location becomes more and more vague, location based services become less and less useful. For example, if a user's location data has been obfuscated to the granularity of 400m, a cab hailing service becomes useless. There is even literature that offers a framework for choosing the best location privacy system for the task at hand, [97, 96]. However, even after choosing the best system for preserving privacy, the location based service may not function correctly or adequately. The idea of a one-size-fits-all solution to location privacy is, at the least, controversial, as the idea of obfuscation has been shown to have security drawbacks [104, 51].

Obfuscation, even k-anonymous obfuscation, does not provide adequate privacy when applied to location information, because attackers can still perform trajectory attacks. Furthermore, k-anonymity does not protect users when they are in a densely populated area. While it is true the attacker cannot distinguish the user from the many other users near her, the attacker is actually given higher accuracy results, because many users are more likely to be near one another, and the location information needs to only be slightly obfuscated to be indistinguishable from other users. The result is that the attacker can learn your location with only a few meters of inaccuracy.

In order to achieve location privacy while still maintaining the proper operation of location based services, we must take each service on a case by case basis. Privacy and security must be built into the design. This is the goal of [113], [78], and [8]. Each of these papers attempt to design one specific location-based service in a privacy preserving way. Mobishare, [113], allows users to share their (authenticated) location values with one another, without allowing any third parties to link user (OSN profile) with location. However, it requires significant changes to the infrastructure including cellular towers. Our protocol avoids any changes to the network infrastructure.

There are several papers that focus on a service similar to ours [125, 109, 54, 98, 68, 79, 82, 46]. All of these papers offer the theory behind private proximity testing. Our first private proximity detection protocol closely relies on Nergiz's et. al., work [69] but makes a critical improvement, which allows our protocol to run in a much lower time. The main contribution of our work is a pragmatic, configurable, and performant implementation of the theory involved in these works. We provide a thorough evaluation, including end-to-end measurements to ascertain the system speed. The technical challenges here are the numerous engineering problems and security concerns. Additionally, we show an extension of our system that allows users to be alerted as soon as their friends are nearby efficiently. We also provide a second private proximity detection protocol, similar to [83], that is based on oblivious transfer, using bloom filter data structures.

Other approaches for private proximity detection are also being explored such as the use of ambient sensor data (such as radio dynamics [120], or background audio [106]) to act as a shared, location dependent secret. The drawback of these systems is that users cannot define what they consider to be ``nearby'' with fine granularity, as they can in our system using P_b. Other encryption systems, such as order preserving encryption, may also be used to build a system similar to the one described in this work [77, 5, 35]. We leave this avenue open as possible future work.

2.2 Proximity and Audio Networking

Our second contribution is an ultrasound modem for proximity networking. Audio communication has been proposed in the past, but these legacy systems are almost always either audible [26, 59, 60] or designed for use underwater [6, 102, 103, 118]. Although, one work, [59] does have a section on ultrasound transmission, they work with laptop hardware, and achieve only 8bps. There has also been a recent proliferation in attempts to replace NFC by commercial projects [23, 24, 80, 90, 100]. Unfortunately, none of these projects published technical details, so we cannot compare them to our system with much granularity. However, two of them advertise concrete bit rates; Zoosh by Narette advertises 300bps [80], and SSCConnect advertises 2.2kbps [90], both of which are much slower than our system.

Audio is an attractive choice as a networking medium in water, [6, 102, 103, 118], because its propagation is much better than traditional RF energy. These underwater audio modems are fundamentally different from our scheme because they don't have the additional constraints of remaining inaudible for humans, or the limited sample rates available on typical smart mobile devices . The hardware for under water transmission is usually custom built.

Some systems which try to use other media for proximity based networking have been been propose in recent literature [94]. A system called ``Cobra'' was proposed by Hao et al., [34], which makes use of devices' LCD screens and cameras to transmit data at roughly 100kbps. Their system produces 2D, five color bar-codes, which are displayed on the screen of the sender and decoded by the camera of the receiver. Another system called ``Pulse'' [42] uses the magnetometer for one-way communication, achieving roughly 44 bits per second. What we propose in our work may have similar application design challenges, but is fundamentally different in that we make use of high frequency sound as our transmission medium.

In addition to [59], there are three relevant academic works in which ultrasonic audio in air is used. Hanspach and Goetz [33] re-purpose two existing audio modems by changing the carrier frequency to 18.6kHz. They incorporate these modems in malware designed to bridge traditional ``air-gap'' systems. They can achieve at best 20bps at a range of 19.7 meters using two Lenovo brand laptops. Matsuoka et. al. [62] apply a similar OFDM based modulation scheme to our own, but are only able to achieve approximately 1kbps. And, a previous Ubicomp paper, AirLink [12], uses ultrasound for data transmission and also measures Doppler shift to identify the intended receiver. In contrast to these works, our system is specific to common smart mobile devices, and it easily outperforms all in speed, achieving greater than 4kbps on a variety of smartphones and scenarios.

In the recent past, work in audio based networking was focused on making the audio tones audible but melodic and more pleasant for the user [59, 60]. However, even melodic tones are not desirable in all situations. They are also subject to background noise, which is typically more prevalent for audio in the audible spectrum. Previous work, ``Dhwani'' [88], emits some noise in all of their modulation implementations (OFDM BASK, QPSK, and 8-PSK) because they operate on 1kHz of spectrum centered on 6.5kHz, and because they do not make any effort to work around the noises emitted as a result of modulation. Their system achieves data rates of 2.4kbps with 80% packet success, 1.6kbps with 95% packet success, and 800bps with 100% packet success using 8-PSK, QPSK, and BPSK respectively. Because our system uses ultrasonic audio, it is completely silent and faces a substantially greater technical challenge in modulating without making any human audible sounds.

There has also been some recent literature on protocols that enable secure, authenticated proximity networking. EnCore [4], Smokescreen [14], BlueID [38], and SDDR (secure device discovery and recognition) [50] are all protocols for neighbor discovery, which rely on existing network mediums; Bluetooth, WiFi (ad-hoc / Direct), or NFC. Our contribution is to provide a new networking medium that these works can be layered on top of. It may even be possible to create new protocols designed to leverage the unique characteristics of the underlying medium. For example, using a long range medium to find candidates and their distances [124, 127], and then using our ultrasound modem with the proper sending power to transmit encryption keys only to the nearest n candidates.

2.3 Information Leakage on Smart Devices

In our final work, we improve mobile operating systems by exposing information leakage through physical media covert channel attacks, and then designing and implementing a defense scheme to protect sensitive user data against them. Covert channels have a rich academic history [55, 63]. Butler Lampson first described ``the confinement problem'' [47], in which one entity (the client) must trust another (the server) with some data to perform some calculation. Ideally, the server would be confined, and unable to transmit or store the data. Mr. Lampson, however, was unable to envision all the possible ways that the service may transmit this data. To put our work in the context of the confinement problem, our work adds new channels by which a confined server can transmit data to itself, or to another process.

Covert channels on smartphones have been studied previously [11, 92], but to the best of the author's knowledge, we are the first to propose covert channels that use physical media, rather than internal / virtual media such as processor workload, cache manipulation, system wide settings, or file sizes.

There are several works [61, 92, 108, 111] that attempt to steal some data from the user, utilizing a variety of different internal / virtual covert channels to perform privilege escalation, and circumvent taint-tracking analysis, similar to our work. Our contribution is unique in that it is the first to use physical media covert channels. These channels are particularly dangerous, because they are much more difficult to identify as malicious.

Our defense scheme is an extension of the state of the art defense schemes, aimed at tracking sensitive information as it flows across a physical media covert channel. Works on defense mechanisms for the attacks similar to what we propose in Chapter 5 can be broken into three categories. In the first, researchers propose that we replace sensitive data with ``mock" data when feeding it to untrusted applications [7, 37]. These systems do protect user information, but they erode the quality of benign applications and are intrusive to users, who must make either a few broad decisions or many small decisions to protect their data. This goes against our goal of ensuring the user does not have to make elaborate or complex policy choices to protect their data.

The second category of defense is taint tracking analysis, [9, 22, 28, 45]. In these systems, sensitive information APIs are marked as ``sources." Variables that store this information (e.g., microphone data, user contacts) are marked as ``tainted." The tainted data are followed through the execution of the program, tainting other data that they effect explicitly (direct assignment). If tainted information reaches a sensitive sink (e.g., Internet Socket, IPC, etc.), the flow is stopped and/or the user is notified.

Machine learning is proposed in [89] to identify sources and sinks during run-time. Their system, ``SuSi", relies on supervised learning, which means a list of known (manually tagged as malicious or benign) Android APIs are used to train a classifier. The system classifies based on features such as method name (i.e., ``get...", ``put..."), and method call return type. These taint-tracking systems cannot be naively adapted for use to defend against our covert channels, because we cannot treat every invocation of a sensor API with sensitive information, to be malicious. An application may use a device benignly at one time, and maliciously at another time. For example, an incoming SMS will almost certainly activate the speaker or vibration motor or both benignly to notify the user.

The third defense mechanism category is alternate permission systems, [25, 56, 93, 112, 119]. These systems are not a good solution to the attack proposed in our work, because any user interaction with sensitive sources is interrupted with some GUI prompt. Because so many covert channels can be built, and sensitive information is commonly accessed on these devices, these prompts would arise far too frequently. Many systems propose more fine-grained permissions, [10, 75, 121]. These systems rely on the user to construct policies, which will stop malicious use cases, but allow benign ones. This, again, goes against our design philosophy of convenience for the end-user.

Our defense scheme is similar to the recent work ``ASM'' [36]. Both systems hook into key system events, and then allow application developers or security researchers to define specific actions that can be taken on these events. We are not able to leverage this work directly in our implementation because the current ASM implementation does not include the devices we focus on in our work such as the sensors, speaker and microphone, or the vibration motor. Our contribution, which ASM encourages, is the specifics of the treatment once these events occur.

Chapter 3

Location Sharing with Trust and Proximity Based Access Control

Users commonly need a mechanism to determine if they are near one another to support many of today's popular mobile applications. In this chapter two protocols are presented, which provide this functionality in a private and secure way. We aim to show that private variants of these applications are possible, thereby improving the security and privacy for users at the application level.

3.1 Background

Smart mobile devices are now ubiquitous in the developed world. It has been estimated recently that there are 2.6 billion such devices worldwide, and that the market is becoming saturated in the U.S. and Europe [58]. Modern smart mobile devices are able to assess their own location via a variety of techniques including built-in GPS radios, WiFi based location mapping, and cellular tower triangulation. Because of this, more and more location based services and applications are being offered to, and used by, consumers. Service providers can offer useful location based services such as cab hailing, directions, nearby friend alerts, fair rendezvous points between friends [8], and simple location sharing. Some popular OSNs, such as yik-yak [3] and tinder [2] are centered on the concept of enforcing distance based access control. However, location information is highly sensitive, and it must be protected and safe-guarded from potentially malicious parties. Providing these services in such a way that protects user location data, while at the same time remaining practical from a usability point of view is the core contribution of this work. Specifically, we aim to provide private proximity detection (PPD, sometimes called private proximity testing: PPT) in a pragmatic and highly usable way. In private proximity detection two users wish to determine if they are near one another, without revealing their actual locations to each other or to any third party.

Location information privacy is nuanced and location information is highly valuable to attackers for two reasons. First, using this information, adversaries can perform a variety of dangerous attacks, which range from mild inconveniences, such as dropping by for lunch unexpectedly, to targeted advertisements, and a variety of much more serious attacks. An adversary can use location information to more easily steal a user's identity, determine the answer to common security questions, and even stalk the user. The very nature of location information introduces the threat of physical harm. It can even be used by governments to enforce strict, totalitarian-style laws. Secondly, it can easily be used to derive other information about a person. For example, it is trivial, given a location trace, to determine the home address of a target victim (the address they are at most frequently between 12:00am and 7:00am), or their place of work (similarly) [1]. Inferring information from a location trace can also reveal other, sensitive information when more sophisticated analysis is employed [15], [85].

Currently, it is common for many location based service providers to require the user to upload their location information directly to the service provider. This information is then used to perform some computation, which is vital to their service, and offer some benefit back to the user(s). This means, however, that service providers have unfettered access to their users' location data. Unfortunately, services providers are weakly motivated to protect this information from the world at large, as has been demonstrated in part by the

NSA privacy revelations [29] and the widespread use of targeted advertising.

Clearly, providing location based services, such as those mentioned previously, in a private and practical way is profoundly important. However, doing so is not trivial. By providing a mechanism for private proximity detection, we enable these services to use our service, and provide the same functionality, *without* requiring users to expose their location information to anyone including other users, the service provider, and any third parties such as advertisers, Internet service providers, or malicious attackers.

Unfortunately, providing private proximity detection for location based, mobile applications and services is not trivial. The most naive approach to providing location based services in a secure and private way is simply encrypting user location data before transmitting it to the service provider. Unfortunately, this does little to protect the user, since the service provider is not necessarily trusted. Additionally, users have very high expectations about the running time and configuration of services and applications they use in general. These standards are especially difficult to meet for developers on smart mobile devices, which typically have limited computational resources such as CPU and RAM. Furthermore, users will have naturally different definitions of ``nearby'' in proximity detection, which depend on a variety of factors including the exact purpose of the application, the privacy desires of the user, and the other party with whom their proximity is being tested. These factors complicate the task of providing private proximity detection. Providing PPD in a naive way, results in a system that is too rigid, not robustly private, and most importantly, is not able to run fast enough on consumer grade smart mobile devices.

Compared with previous work, we are the first, to the author's knowledge to investigate this problem in a specific application scenario. While there is work on private set intersection, and proximity detection in general, we investigate the research issues that arise was attempting to provide these services to users in a pragmatic way. Specifically, we overcome limited CPU and memory resources, we provide reasonable message passing and reference finding systems, and we allow for a configurable system, which allows the users to specify, online, what they consider to be ``close'' or ``nearby.''

Our system, nearpri, aims to offer location proximity detection in a convenient, privacy preserving, and secure way. Specifically, we build a system which allows a user, (Alice), query another user in her social network, (Bob), to see if they are near one another. Our system, without exposing location information to either party, or the OSN provider, allows Bob to test if Alice is within a certain distance of him (private proximity detection). We refer to this distance as Bob's *policy* P_b. If Alice and Bob are not within P_b meters of one another, both parties learn only this fact. If they are within P_b, then Bob learns this, and he can optionally share this information with Alice or the service provider. Our system allows for every user to maintain their own policy. By requiring that users are near Bob, we provide a secure variant of what is the core of many popular mobile services today. We protect Bob's location information from his friends (or malicious peer users), the service provider(s), and any other, possibly malicious, third parties. Alice, although participating in the private proximity detection protocol, does not reveal her precise location. Only Bob learns anything about her, and he only learns whether or not she is less than P_b meters from him.

In this work we provide two distinct private set intersection protocols, each used to support proximity detection. Our protocols are flexible, in that each user is able to maintain their own policy. Users can change their policies online, and can even assign different policies for different queriers, or at different times. Our protocols are also fast in the face of limited computational resources, common to smart mobile devices. We make the following key contributions:

- We present the design of two private proximity detection protocols, which allow one party to determine, privately and securely, if a second party is within a certain pre-determined, user-configurable distance. Our first protocol is based on homomorphic encryption, and the second is based on oblivious transfer.

- We give two application examples, each making use of private proximity detection, to show the viability of enhancing the security and privacy of user locations in location based applications such as yik-yak [3], and tinder [2]. Our applications demonstrate the practicality of our implementations and are described in Section 3.5.

- We present a novel and elegant method of discretizing GPS coordinates for use in our system. We utilize a binary tree data structure, with a novel construction method, which allows us to support policy granularity as small as 10m.

- We build and evaluate a working implementation of both protocols. We put in great effort to make our systems practical and usable. Our evaluation shows that our protocols are fast, do not require modification of any infrastructure, and do not leak information to any third parties.

3.2 Problem Description

Private proximity systems allow one user, Alice, to query another user, Bob. For our specific implementation, it must be determined if Alice is within a certain range of Bob, called Bob's policy, P_b. Every user in the system defines their own policy, which may be changed at any time, but must remain static while the protocol is running. Determining if Alice is within P_b, without revealing the location of either user to the other, is our definition of **private proximity detection**. Below we define the problem in greater detail.

At all times, every user has a current *location*; a 2-tuple of that user's geodetic GPS coordinates $< lat, lon >$. Where $lat \in \mathbb{R}$ in the range $[-90, 90]$, and $lon \in \mathbb{R}$ in the range $[-180, 180]$. Each user also maintains their policy, P_b, which is a factor of ten meters in the range $[30m, 20000km]$, ($20000km \approx \frac{1}{2}$ the Earth's circumference). The goal is to determine if Alice's *location* is in the circle centered at Bob's location, with radius P_b.

To facilitate this, we divide the earth into a grid based on the longitude, latitude coordinate systems as shown in Fig. 3.1. Specifically, we descritize the grid with a granularity of 10m. Bob's location and policy define a spherical ``cap'' (circular area on a sphere), which is then approximated using grid squares. Because the squares only approximate the spherical cap, they introduce some error. Thankfully, this error is negligible and grows smaller as the policy grows larger. This is because a larger policy requires more squares to represent, and like pixels on a computer screen, the squares approximate the circular shape better.

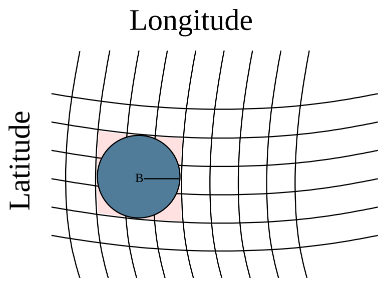


Figure 3.1: Geodetic grid and spherical cap. A small error is introduced due to the difference between the theoretical circle, centered at Bob, and the squares used as an approximation.

3.2.1 Assumptions

We assume that Alice and Bob are curious but not malicious. They will follow the protocol correctly, but once the protocol has ended they can perform any computation they want on the information (encrypted or otherwise) acquired. In related work this is usually referred to as ``semi-malicious'' or ``honest but curious.'' We do assume that users cannot forge their GPS coordinates, which obviates the possibility of many attacks such as [122]. While it is possible for users to forge their location using applications or the Android developer tools, it is detectable. Either the user has modified the developer option ``Allow Mock Locations'', or the user has root access on the device [18], which is technically challenging for typical consumers, and very difficult to be made undetectable even for experts. Although we did not implement these checks in our application, adding them would be trivial.

3.3 System Design

In this work we present the design and implementation of two proximity detection protocols. Both designs are fully distributed, as depicted in Fig. 3.2. All computation is done on the users' mobile devices. However, the clients do connect to a third party, centralized server for message passing. (Facebook Chat in the first system and a custom message passing server in the second). By doing this, we do not have to do name/address resolution of clients directly, we easily circumvent NAT and firewalls, which block incoming connections, and for the case of Facebook, messages are stored when users are not online, allowing users to be temporarily ``unavailable.'' Despite the fact that our system sends messages over the Internet, and/or directly through the Facebook, user location information is never leaked.

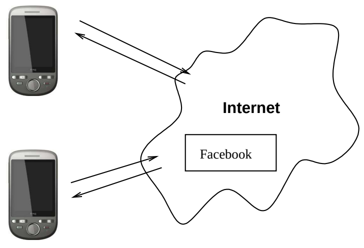


Figure 3.2: Physical Setup. Phones communicate by sending messages, over the Internet, in the Facebook chat API to one another.

3.3.1 Homomorphic Cryptography

Homomorphic encryption takes a foundational role first PPD protocol. We use Paillier encryption, in which the following equations hold true.

$$D[(E(m_1) * E(m_2))\%n^2] = (m_1 + m_2)\%n \tag{3.1}$$

$$D[(E(m_1)^{m_2})\%n^2] = (m_1 * m_2)\%n \tag{3.2}$$

$$D[(E(m_2)^{m_1})\%n^2] = (m_1 * m_2)\%n \tag{3.3}$$

Here E() denotes Paillier encryption, D[] is Paillier decryption, n is the product of two large primes, p and q, and m_i is the plain text message encoded as an integer. The implications of equations (3.1), (3.2), and (3.3) are very useful. Computation can be done with encrypted values that translates to operations in the plain text domain. This allows parties to do blind computation on encrypted values. For more information on Paillier encryption, including key generation, please refer to [114].

3.3.2 Main Idea

In order to better understand the first protocol, we first introduce a simple example. Alice wants to determine if she is within P_b of Bob. Bob is willing to participate in the test, but only if it is guaranteed that he will not reveal his location. In order to achieve this, we begin with two sets of location values α and β, which correspond to latitude and longitude respectively.

As a user moves North or South across the surface of the Earth, we can think of their location in terms of latitude values. However, we have discretized these values into 10m wide ``regions'' or ``bands'' which we number uniquely. These numbers comprise the set β. Similarly, when moving West or East the user moves across discrete longitude bands, which are uniquely numbered to form the set α. This is easily visualized in Fig. 3.3. Because the latitude line selected has an impact on the distance between longitude lines (more at the equator, less near the poles), the number of elements in this set (discrete locations) varies. However for both α and β, every location represents at most 10m of area. By overlaying these two, we arrive at the grid show in Fig. 3.1.

In order to determine if Alice is near Bob, Bob builds two subsets, one from α and the other from β. Bob's subsets contain the location values (ID numbers) of all the locations (bands / grid squares) in the range $[L_b - P_b, L_b + P_b]$ where L_b is Bob's location (longitude or

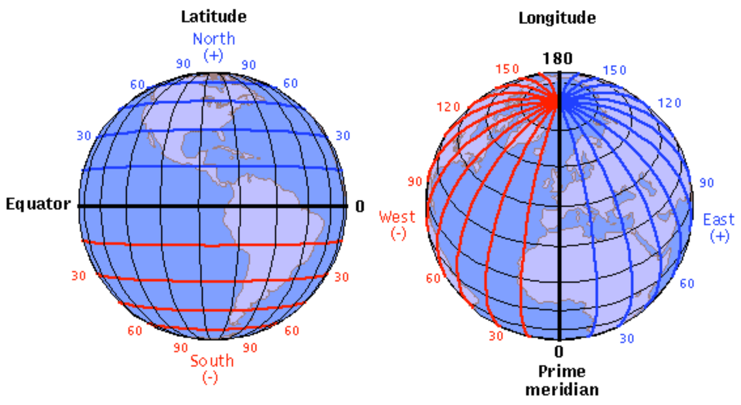


Figure 3.3: Latitude and Longitude bands. Note: in our implementation the bands are closer together than as shown (10m distance at the most).

latitude). Alice builds two singleton sets, each containing the location value corresponding to her latitude or longitude regions respectively.

If Alice's latitude and longitude singleton sets intersect with Bob's corresponding sets, we know that Alice must be within Bob's policy, P_b. Determining if one of Alice's singleton sets intersects with Bob's set is trivial if Alice and Bob can share their values with one another. However, our goal is to hide their locations. In order to do this, Bob generates two sets of several, degree one, polynomials. One set of polynomials are rooted at his longitude set values, and the other; his latitude set values. Specifically, each polynomial is rooted at one values in the range $[L_b - P_b, L_b + P_b]$. This can be seen in Eq. (3.4) where each C_i is equal to a corresponding band / region ID number.

$$(X - C_1), \quad (X - C_2), \quad ... \quad (X - C_n) \tag{3.4}$$

Bob sends the encryption of the negated coefficients ($E(-C_i) \quad \forall i \in [1, n]$) from these polynomials to Alice. Referring to Eq. (3.1) we can see that Alice can evaluate the polynomials at her singleton set element's value, L_a, even though she only has the coefficients in cipher-text by using homomorphic encryption.

$$D[(E(L_a) * E(-C_i))\%n^2] = (L_a - C_i)\%n \tag{3.5}$$

Therefore, Alice computes $E(L_a) * E(-C_i)\%n^2$, and sends back the resulting value to be decrypted by Bob. If the decrypted result is 0, Bob knows that Alice's value $L_a = C_i$ indicating that Alice's set intersects with Bob and that Alice is near Bob. This process is repeated once for longitude and then again for latitude.

3.3.3 An Improvement

If Bob uses every node in the range $[L_b - P_b, L_b + P_b]$, the protocol will not scale well as Bob's policy increases, because Bob will generate potentially thousands of polynomials. This is demonstrated in our evaluation in Section 3.8. To solve this problem, we use a binary tree to reduce the size of Bob's set, which is inspired by authors A.E. Nergiz et. al., [69]. For the following we perform everything identically twice, once for longitude and again for latitude. We begin by taking the set of longitude (latitude) location values, β (α), and use these as the leaf nodes in a binary tree.

The leaves are uniquely numbered, from $\mathbb{N}$, using the ID numbers of the set elements (the band/region numbers). The values used for the tree nodes above the leaf level are chosen carefully so that every node in the entire tree is numbered uniquely. For a given, non-leaf node, $value = leftChild + 4003017$. If the node does not have a left child, then $value = rightChild + (4003017 - 2^{h-1})$ where h is the current height in the tree. 4,003,017 is used because that is the largest longitude leaf node value. 4,003,003 is used in the latitude tree. This guarantees that any node at level h will be greater than any node at level $h - 1$ and that all nodes will be uniquely numbered. When building the parent of a particular node, we must know the orientation of the branch connecting these two nodes. Due to the careful choice of the node values above the leaf level, the correct branch directions are encoded in the binary representation of the leaf nodes' values (0 is used to represent a rightward branch, and 1 represents a leftward branch). This allows us to build only a segment of the tree, efficiently, from the leaf level up.

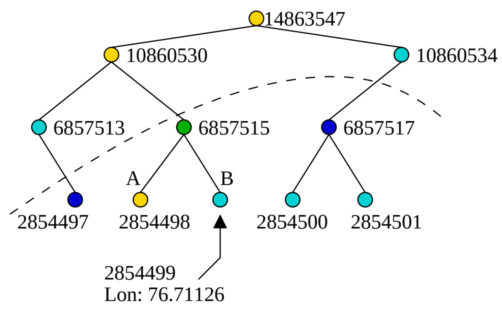


Figure 3.4: Example tree segment (longitude only). The tree was generated by Bob at 37.2710, 76.71126, and $P_b = 10$ meters. Bob's wall set = {2854497, 6857515, 6857517}. Alice's path set = {2854499, 6857515, 10860530, 14863547, ...}. Node 6857515 can be generated from the left child or the right child. $6857515 = 2854498 + 4003017 = 2854499 + (4003017 - 2^{1-1})$. The leaf node at longitude 76.71126 generates $2854499 \approx \frac{76.71126+180}{0.0000899321}$.

3.3.4 Utilizing The Binary Tree

Alice and Bob each construct two binary trees in this way, one for their longitude and another for their latitude values. Bob uses a subset of nodes from this tree to use as roots for constructing his polynomials. We refer to Bob's subset as the **wall set**. The *span* of a node is the set of leaf nodes which can be reached as children of that node, (i.e., can be reached by following depth first search from that node). Bob's wall set is the set of nodes, highest in the tree, with spans that collectively cover all of, and only, the elements in Bob's range, $[L_b - P_b, L_b + P_b]$. Bob uses the values from the wall set to build his polynomials. It might seem obvious that the root node alone would be an adequate (singleton) wall set. However, the root node often spans a larger range of leaves than P_b allows. In Fig. 3.4, Bob's wall set = {2854497, 6857515, 6857517}.

We refer to Alice's subset as the **path set**. The path set is simply the set of nodes connecting her location, (leaf node) to the root node. In Fig. 3.4 Alice is located at node 2854498 so her path set = {2854498, 6857515, 10860530, 14863547, ...}. Alice's path set actually extends further than is depicted in Fig. 3.4 but these nodes are omitted for brevity.

The intuition behind these two sets is that Bob's set forms a wall, (the dashed line in Fig. 3.4). Alice's path set will necessarily break through this wall if Alice is within P_b of

Bob. In Fig. 3.4 Alice's and Bob's sets intersect at node 6857515. The value at which the two sets intersect is an element of Bob's **wall set** and therefore, a root of one of Bob's polynomials.

3.3.5 Putting It All Together

Prior to the protocol running, Alice constructs her latitude and longitude trees and derives her path set node values for each. Then, when she wants to query Bob, she sends a simple message initiating the protocol. Bob constructs his trees and corresponding wall sets. Then, Bob encrypts the negated coefficients of the polynomials (rooted at his wall set values) and sends the encrypted coefficients, along the public key he used, back to Alice. Alice, uses the encrypted coefficients to re-create the polynomials, and uses homomorphic encryption, along with Bob's public key, to encrypt her path-set values, and evaluate the polynomials using them. Bob also includes information about the height in the tree of each coefficient. The height information is useful to Alice because without it, she does not know which of her values to use as input to the various polynomials and the method proposed in [69], is too slow. In the worst case, she must evaluate every polynomial at every one of her location values, approximately $O(n^2)$ work. Using the height information, she can reduce the search space, because she now knows which of her values correspond to which polynomials. So, she uses the height information to evaluate each polynomial only once, using her path set node at the corresponding height. Alice sends the polynomial evaluations, (encrypted), back to Bob, who can decrypt them. Bob decrypts these values and, if any value decrypts to zero, Bob knows that his wall set and Alice's path set intersect at this point, and that she is within his policy. This is repeated once for longitude and again for latitude.

When the policy testing is completed, Bob sends a message back to Alice indicated that they have passed the proximity test. Optionally, a third party service can also be notified, in order to provide some functionality such as hailing a cab for both users.

3.4 Alternate Approach

In our evaluation in Section 3.8 we show that our system is able to process queries in the order of tens of seconds. This is because of the expensive multiplication and exponentiation of large cipher text values as well as the limited transmission rate of the Facebook message passing system. Due to the limitations of this approach, we also present a second PPD protocol designed to achieve the same goal. This second protocol is a key contribution different from our previous work [71].. In this section, we detail the second private proximity detection protocol, which utilizes bloom filter data structures and oblivious transfer to process queries more quickly in order to be more practical for end users. First, we provide some background on these two tools.

3.4.1 Bloom Filter and Oblivious Transfer Background

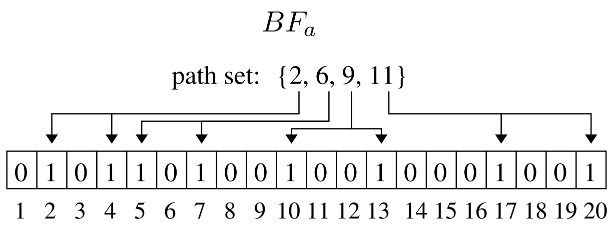


Figure 3.5: Example bloom filter. The map set of element 6 = $\{5, 7\}$.

A bloom filter (BF) is a relatively simple data structure, which is used to identify the elements of a set. BFs are very efficient for inserting elements, and verifying if an element is a member of the set, but there are also a few drawbacks. A BF is represented by an array of m bit values, all of which are initialized to zero. In order to insert an element, that element is hashed by k different hash methods. The output of each hash method (modulus m) points to an index in the array, which is then changed from 0 to 1 (values that are already 1 remain unchanged). The collection of all the index values for a particular element we call the ``map set'' of that element. In order to test if an element is a member of the set, that element is fed into the k hash functions to get the map set, similar to insertion.

However, instead of flipping these k locations to 1, the current values are inspected. If all of these values are 1 already, then the element is known to be in the set. An example bloom filter is illustrated in Fig. 3.5 with $k = 2$ hash methods, $m = 20$ bits, and the elements of the set $\{2, 6, 9, 11\}$ have been inserted.

Because there are only m bits, and the index values used are modulus this m, there is a relatively high chance of collision. In other words, the map sets of two different elements may have a non-zero intersection, and a single map set may have duplicate elements. Because of this, BFs are a probabilistic data structure in that they sometimes return false positives when checking if an element is a member of the set. Secondly, BFs do not store the data of the elements inserted, they only store the information about whether or not an element *has been inserted*.

Oblivious Transfer (OT) protocols allow a sender to transmit part of its data to a second, ``receiver'' party, in a manner that protects both of them. To understand OT conceptually, imagine the sender party has two pieces of information A and B. The sender is willing to reveal A or B, but not both (exclusive OR). OT allows the receiver to read A or B but not both, *and* the sender does not learn which value the receiver has read! This is commonly known as ``one-out-of two'' OT. The receiver must commit to a bit value, which determines which value they will read (e.g., 0 for A and 1 for B). In this work, we implement an OT protocol similar to that described in Naor's and Pinkas' seminal paper [67]. As described in this work, OT requires a group G_g of order q, which is a sub group of Z_p^* where g is a generator, p is prime and q is coprime to p. To provide $p, q,$ and g, we use a DSA key pair. In our implementation, the sender builds an n X 2 matrix essentially comprised of n ``one-out-of-two'' oblivious transfers. Because each row is an instance of ``one-out-of-two'' OT, the receiver will specify a ``selection bit'' for each row, to determine which of the two values to access. We refer to all of these selection bits collectively, as the ``selection vector'' or ``sel.'' Although the sender will send the entire matrix (encrypted), OT guarantees that the receiver will only be able to recover one of the two values in each row of the matrix. If the selection bit for that row is ``1'', the receiver can recover the value in the first column, and if it's ``0'', the second column. As stated before, the sender will not know which values the receiver actually recovers. For more detailed information on the nature of OT, please refer to [67].

3.4.2 BF and OT design

Our second approach is much faster for two reasons. First, because it does not rely on expensive homomorphic computation (specifically, multiplication as seen in Eq. 3.5 and exponentiation as seen in Eqs. 3.2 and 3.3 on large cipher-text values) the computation is faster. And, second, because we use a traditional TCP / IP server, with a custom server middle-man implementation, we are not limited by Facebook's messaging throughput. However, the setup is very similar. Alice generates her **path set** and Bob generates his **wall set** based on a longitude and latitude binary tree as described in Section 3.3.3. At this point, the goal is for Alice and Bob to privately determine if these two sets intersect. We now diverge from the original protocol. All of the following is repeated twice. First for latitude values, and then again for longitude values.

Alice and Bob each generate a total of three BFs, all of which use the same k hash functions, and have the same length m. Alice generates one BF, BF_a, in which she inserts all the elements of her path set. Bob generates two BFs, BF_{b0}, and BF_{b1}. In BF_{b1}, Bob inserts all the elements from his wall set. Then, Bob generates a new temporary set consisting of all the possible path set nodes that Alice may have, assuming Alice is near Bob, i.e., within Bob's policy. We call this set the **super path set**. In Fig. 3.4, the super path set would consist of all the nodes shown, because Alice may be at any of the leaf nodes shown in the figure. These elements are inserted into BF_{b0}. The key insight here is that, at this point, assuming Alice is near Bob, he can use BF_{b1} to find indices in BF_a that must be set to 1 and similarly BF_{b0} to find indices in BF_a that must be zero.

An illustration of the bloom filters ($k = 2$ and $m = 20$), along with a simplified example tree segment, is given in Fig. 3.6. Here Alice is at node 2, so her path set $= \{2, 6, 9, 11\}$, which are each inserted into BF_a. Bob is located at node 3 ,and his policy only includes nodes $\{2, 3, 4\}$ therefore his wall set is $\{2, 7\}$, which are inserted into BF_{b1}. In order to intersect with Bob, Alice must have either node 2 or 7 in her path set. Therefore, the mappings of either 2 or 7 must be 1 in both BF_{b1} *and* BF_a. To find the elements for BF_{b0}, Bob finds all the nodes in all possible path sets. In other words, Alice may be at node 4, with path set $\{4, 7, 10, 11\}$ or she may be at 3, with path set $\{3, 7, 10, 11\}$ or, she may be at 2 (in fact she is at 2) with path set $\{2, 6, 9, 11\}$. The union of these sets $\{2, 3, 4, 6, 7, 9, 10, 11\}$ is used to fill BF_{b0}. Alice may be at node 1, but in this case Bob would not consider her nearby, so this case is not included (similarly with 5). Because Bob has added any possible location of Alice, assuming she is nearby, any bit in BF_{b0} that has not been flipped to 1, should similarly not have been flipped in BF_a, and therefore Bob can use BF_{b0} to indicate places in BF_a that should be 0.

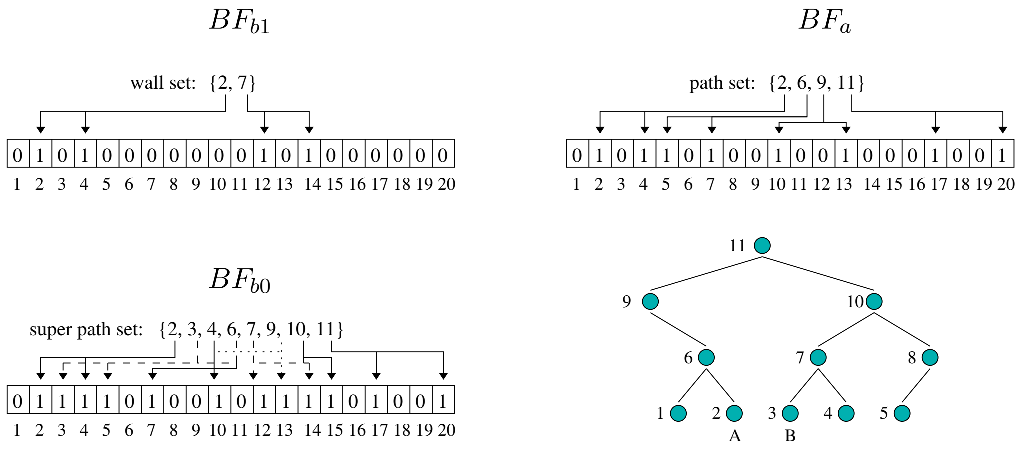


Figure 3.6: Example BFs and tree segment. Because the element 2 is present in BF_{b1} and BF_a, then the 1's in BF_{b1}, from the map set of 2, correspond to the same locations in BF_a. Additionally for any 0 location in BF_{b0}, the same location is 0 in BF_a.

3.4.3 Protocol

From this point on, Bob must present several challenges to Alice about the contents of BF_a. However, in so doing, he must a) not reveal any information about his own location, and b) assure Alice that he is not learning her location unless they are near one another.

In order to do this, oblivious transfer (OT) is used. An illustration of the protocol is given in Fig. 3.7. Throughout this section, we give the high level steps from Fig. 3.7 in form: $\textcircled{X}$.

First, Alice establishes a connection with the middle-man server and sends an ``a" query indicating to the server that Alice wants to connect with Bob specified by ip_addr $\textcircled{1}$. The server then establishes a forward such that any data sent from Alice will be redirected to Bob and vice-versa, any data from Bob will be redirected to Alice.

Then Alice creates a DSA public key and sends the components, $p, q,$ and g to Bob. Bob creates a subset of bloom filter index values we call the ``selection table" and a selection vector, illustrated in Table 3.1, which he sends back to Alice $\textcircled{2}$. We use $BF_{bx}^{y}$ to denote the index of the $y_{th}$ ``1" (``0") in BF_{b1} (BF_{b0}). Bob chooses $4 * k * l$ spots in total where l is the size of Bob's wall set, and k is the number of hash functions used in the various BFs. $k * l$ of these indices will be index locations in BF_{b1} that contain a 1 and another $k * l$ are locations in BF_{b0} that contain a 0. Then, another $2 * k * l$ are random or ``fake" indices. All $4 * k * l$ values are randomly shuffled into the selection table (right side of Table 3.1), with the corresponding selection vector (sel). It is important to note that the selection vector determines the location of the real value in the table. The selection table is then sent to Alice, excluding the selection vector. In the example in Table 3.1 $BF_{b1}^{1} = 2$, and $BF_{b1}^{2} = 4$ because the 1_{st} element in BF_{b1}, 2, maps to indices 2 and 4 according to the BFs given in Fig. 3.6.

	1	**0**	**sel**		**1**	**0**	**sel**
$2*k*l$	BF_{b1}^{1}	fake	1		fake	BF_{b0}^{3}	0
	BF_{b1}^{2}	fake	1		BF_{b1}^{2}	fake	1
	fake	BF_{b1}^{3}	0	$\xrightarrow{shuffle}$	BF_{b0}^{1}	fake	1
	fake	BF_{b1}^{4}	0		fake	BF_{b1}^{4}	0
	BF_{b0}^{1}	fake	1		BF_{b1}^{1}	fake	1
	fake	BF_{b0}^{2}	0		fake	BF_{b0}^{2}	0
	fake	BF_{b0}^{3}	0		BF_{b0}^{4}	fake	1
	BF_{b0}^{4}	fake	1		fake	BF_{b1}^{3}	0

Table 3.1: Example selection table of index values. On the left, the table is constructed in two columns of $2 * k * l$ values. The selection vector is given in the third column. The table is then shuffled (on the right) and sent to Alice, excluding the selection vector.

At this point, Alice uses the received selection table of index values to lookup bit values in BF_a and places the corresponding values from BF_a in a ``bit value table". For the example in Fig. 3.6 and Table 3.1, Alice's bit value table is given in Table 3.2.

1	**0**	**sel**		**1**	**0**
fake	$BF_{b0}^{3} = 9$	0		1 or 0	0
$BF_{b1}^{2} = 4$	fake	1		1	1 or 0
$BF_{b0}^{1} = 1$	fake	1		0	1 or 0
fake	$BF_{b1}^{4} = 14$	0	$\rightarrow$	1 or 0	0
$BF_{b1}^{1} = 2$	fake	1		1	1 or 0
fake	$BF_{b0}^{2} = 6$	0		1 or 0	0
$BF_{b0}^{4} = 18$	fake	1		0	1 or 0
fake	$BF_{b1}^{3} = 12$	0		1 or 0	0

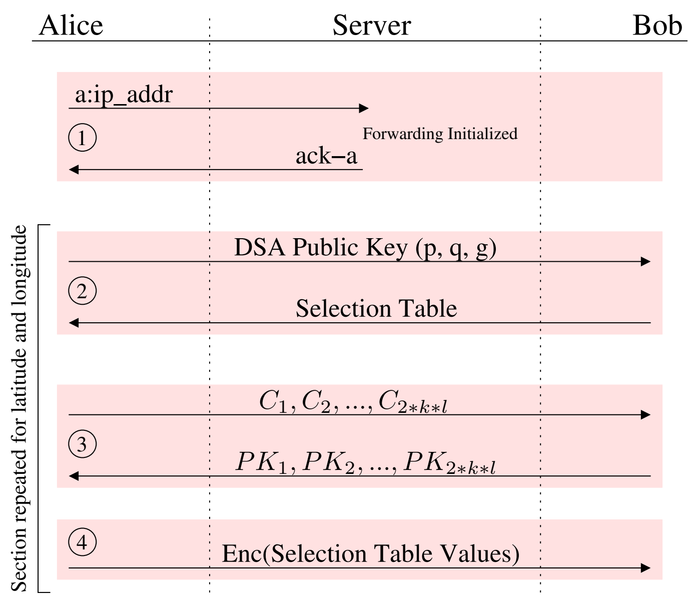
Table 3.2: Alice's bit value table (right side) derived by looking up the bit values of BF_a given the indices in the selection table (left side) provided by Bob. Example: BF_{b1}^{3} is the first hash location of element 7 in BF_{b1}, which is 12. Index 12 in $BF_a = 0$.

Alice generates a random value $C_i \epsilon Z_q$ for each row of the matrix and sends these values to Bob. Bob then chooses a random $1 \leq u \leq q$ and creates two public keys $PK_\sigma = g^u$ and $PK_{1-\sigma} = C_i/PK_\sigma$ for every row of the matrix. Here σ = selection bit at row i. For each row, PK_0 is sent to Alice, which she can use to compute the missing $PK_1 = C/PK_0$ (3). She chooses two random numbers $r0$ and $r1$ and, using the table of bit values, encrypts the left column as E_1 and the right column as E_0.

$$E_1 = g^{r1}, H(PK_1^{r1}) \oplus val \tag{3.6}$$

$$E_0 = g^{r0}, H(PK_0^{r0}) \oplus val \tag{3.7}$$

Here, $H()$ is a hash function and val is the value from the corresponding spot in the table of bit values. The values are sent to Bob, and he can then compute $H((g^{r_\sigma})^k) = H(PK_\sigma^{r_\sigma})$ and uses it to decrypt E_σ (4). Here σ is the bit from the selection vector for the corresponding row being decrypted. The selection vector (known only by Bob) determines which parts of the encrypted table Bob can recover. That is, for each row, if the selection vector bit is 1, Bob can only decrypt the left column value, and if it is 0, he can decrypt only the right column value. Because Alice does not know the selection vector, she cannot deduce which values are fake and which are real. Bob then ensures that for one of the values in his wall set, all of the corresponding bit values (the entire map set) are ``1", indicating an intersection and alerting Bob that Alice is near him.



Alice
Server
Bob
a:ip_addr
1
Forwarding Initialized
ack–a
Section repeated for latitude and longitude
DSA Public Key (p, q, g)
2
Selection Table
$C_1, C_2, ..., C_{2*k*l}$
3
$PK_1, PK_2, ..., PK_{2*k*l}$
4
Enc(Selection Table Values)

Figure 3.7: OT Protocol. First Alice sends a message to the third party server to establish a forward to Bob. Then, the OT protocol, as described in [67] is carried out using an DSA key.

This is repeated once for latitude and then again for longitude values and if Bob finds that there is an intersection in both cases, then he knows that Alice is nearby. He can now safely transmit his precise location to Alice (encrypted).

3.5 Applications

We propose two applications, which we use in our evaluation in Section 3.8. In the first application, (App1), Alice wants to learn Bob's location. However, Bob will only reveal his precise location to Alice if he can verify that Alice is within a certain distance of him. Private proximity detection is first used, using one of the protocols described previously, and, if Bob can verify the distance between Alice and himself is less than P_b, then he encrypts his location with Alice's pre-published public key, and sends the ciphertext to Alice for decryption. Location based access control of Bob's location makes sense in this application because Bob is relatively safe from targeted attacks, as the attacker effectively must guess a location near Bob, in order to learn Bob's location. The application is similar to other popular applications such as Tinder [2], which displays users to one another if they are in roughly the same area. However, it is private where Tinder is not.

Our second application (App2) is friend co-location notifications. Alice maintains a list of friends, in her social network, for which she would like to receive a notification when they become near her. In this application, one instance of private proximity detection is run repeatedly for each of her friends, in parallel. When one of the protocol instances reveals that the friend is near Alice, a notification is raised and Alice can contact that person directly to meet up. It is important to note that the nearby threshold is set individually by Alice's friends (various Bobs), which allows them to maintain respective levels of varying privacy. The feasibility and practicality of our protocols in supporting these applications is explored in our evaluation.

3.6 System Design Challenges

Pre-computing Encryption Keys Our system relies heavily on encryption for both private proximity detection protocols and applications. In order to speed up our system, our client software automatically pre-computes encryption keys, storing them in memory for use later. As soon as the application is started, keys are generated in a low priority background thread and stored in memory. The key generation thread sleeps between generating keys and the sleep time grows exponentially with the number of keys already in storage so that the system does not generate a surplus of keys, wasting memory.

Avoiding Unnecessary Array Copy Java provides many convenience methods and tools, which ease development, but can cause significant performance issues. In our implementation, we initially used an ArrayList$<$TreeLeaf$>$ to store the leaf-nodes (a custom class) when creating the binary tree. We first generate the leaf node corresponding to the user's location, then we generate a number of leaf nodes to the left and to the right of the user's location node. Because the nodes on the left are inserted into the ArrayList using ArrayList.insert(0, newNode); the ArrayList implementation performs an array copy on every insertion. We then updated our implementation to avoid situations like this, by using primitive TreeLeaf[] data structures and iterating through the array in order.

3.6.1 Messages

In any distributed system, it is assumed that nodes can send messages between one another. In our intended use case, the nodes are smartphones, which typically a) change IP address frequently and b) are commonly behind firewalls, network address translation (NAT) and other port blocking network infrastructure. Therefore, it is very challenging for two arbitrary devices to establish a TCP connection with one another. In this chapter we propose two solutions to this problem.

Facebook Messenger In our first solution, we allow the clients to connect through the Facebook Chat API, depicted in Fig. 3.2. By doing this, users can do name/address resolution using the real names of their friends in the Facebook social network. The Facebook Chat protocol is built on top of the Extensible Messaging and Presence Protocol (XMPP), which features, among many other things, guaranteed delivery of messages. Users start the application, set a username and password, and are able to log in to Facebook Chat using our application directly. Once the user is logged in, they can be make queries and be queried by others who have installed the application. Unfortunately, Facebook enforces a 1000 character limit on messages on their XMPP Chat protocol, which we discovered empirically. In order to circumvent this limit, we are forced to split messages into 1,000 character *packets*, which we encode using a base-32 encoding scheme for better efficiency.

When packets arrive, if the user is currently logged into Facebook chat using our application, the application will parse the packet. To distinguish our packets from normal usage, we prepend our messages with a special symbol: ``@@". When Bob sends messages to Alice, for the first private proximity detection protocol, they are in the form:

$$< \text{@@}T : sess : E(C_1), h_1, ..., E(C_n), h_n, p_k > \tag{3.8}$$

Here, $@@T$ is the packet type T preceded by our implementation's special character, and $sess$ is the session number. Each $E(C_i)$ is the i-th encrypted coefficient, and each h_i is the corresponding height. p_k is Bob's public key, which Alice uses to encrypt her path set values to perform the homomorphic computation of the polynomials. The type number T allows the receiver to determine what stage of the protocol this packet is intended for. For example, a type 2 packet contains the encrypted coefficients from Bob's longitude polynomials. Nine different parcket types are used to handle the different stages of the protocol, which are omitted for brevity.

Third Party Server Messenger Alternatively, we implemented a simple third party server used for message passing only. Because the server does not move, it can be given a static, public facing IP address and ports can be open specifically for our intended application. Then, both clients connect to the third party server, which simply forwards messages between the two clients. In order to establish a connection, two clients first must open a TCP connection with the server. Then, the server provides a simple protocol for selecting another user (Bob) to initiate a query with. Once one user (Alice) selects another (Bob) the message forwarding begins. Our messenger server, used in our evaluation, is implemented as a python script, running on a departmental machine.

Bloom Filter Configuration In our implementation of the second protocol, the bloom filters used are probabilistic, and some parameters must be chosen including how many bits are used in each bloom filter, and how many hashes are used to insert an element. We found empirically that using $m = 262144$ bits (32 KB) and $k = 2$ hashes results in a false positive rate (FPR) of $1.6 * 10^{-8}$ for BF_a, $2.8 * 10^{-9}$ for BF_{b1}, and $5.6 * 10^{-6}$ for BF_{b0} on a typical 300m query. BF_{b0} has the worst FPR, because it is holding the, much larger, super path set.

3.7 Security Analysis

The goal of our system is to provide a private and secure PPD service. This section analyzes the security of the system from the different perspectives of the parties involved.

Our protocols have one minor limitation, because the proximity detection is handled in two rounds. When the first round has finished, Bob can determine whether or not Alice is within P_b ***in longitude only***. At this point, information has been leaked to Bob, who should only learn location information about Alice if she is near him in **both** longitude and latitude. He can fool Alice by continuing the protocol regardless of the result so she cannot infer anything about Bob's location. Fortunately, the information exposed to Bob is limited. He knows that Alice is within P_b of Bob in longitude only. As a result, he can guess she is in a $2 * P_b$ wide vertical band, centered on him, of latitude values all the way around the Earth. This disclosure is relatively mild. There are still millions of latitude values to guess. Due to the weak disclosure of this issue, we felt it unnecessary to modify the system. We have implemented a trivial amount of protection in that the client software does not reveal any information to the user until the protocol has finished. Only at this point can it be confirmed that Alice is within P_b in both longitude **and** latitude.

We can solve this problem completely if we represent all the permutations of all possible $< lat, lon >$ pairs in a single tree. This is substantially different then the current system, which creates two trees, because determining which leaf nodes are within Bob's policy becomes much more complex. Due to the complexity, the significant difference from our current system, and the relative insignificance of the information leaked, we did not attempt this approach.

Messenger Service Messages are passed in-band (i.e., over the social networking service or third party server). We assume that the social network provider can see and read all messages. However, information is not leaked to the provider, because our system maintains that the sensitive contents of all messages are encrypted, and new key pairs are generated for each query. The social network provider (or any message provider) may attempt to perform IP geolocation. However, IP geolocation is coarse grained, and easily circumvented using IP address obfuscation techniques such as TOR or a VPN. While the messenger service is able to launch a man-in-the-middle attack, we can prevent this using some standard authentication mechanism, which is out of the scope of this work.

In the worst case, the messenger service can sabotage the protocol by scrambling messages or refusing to send messages. However, determining which messages are related to the protocol, and which are simply users conversing, comes down to identifying which messages are cipher-text and which are plain-text. This is non-trivial on a large scale and yields little reward. Therefore, it is not practical for the social network provider to do this. The third party messenger server may have a variety of different owners. In fact, the user themselves may choose to maintain one. Because of this, we cannot speculate on what the owner may consider ``practical'' but, at the same time, users can simply choose to use a different server if they find one that is unreliable or they feel it cannot be trusted. If a message passing service attempts to tamper with messages, by modifying or inserting data, this can be detected using simple checksums such as parity bits. Because the transmissions are encrypted, the service will not be able to modify the message in a meaningful way and modify the checksum appropriately.

Alice In the first protocol, Alice receives, from Bob, the coefficients for a polynomial rooted at Bob's wall set values. However, they are encrypted and, therefore, incoherent to Alice. If these coefficients were in plain text Alice could easily determine the roots. Alice may cache the encrypted coefficients sent to her in queries (from Bob). She might then recognize the encrypted coefficients sent to her in future queries. This would only occur if Bob was at the same location, with the same policy, and using the same key. Fortunately, in our system we generate a new key for every query. By doing this we significantly decrease the likelihood of one user recognizing another re-using the encrypted coefficients from a previous query. Bob also reveals height information about each coefficient to Alice. Alice may be able to deduce Bob's policy from this height information, (larger heights indicate a larger policy), but this disclosure is minimal and does not indicate to Alice anything about Bob's location.

In the second protocol, Alice only receives the selection table, and a collection of public keys (PK_i's). The selection table partially reveals to Alice which nodes may be present in Bob's wall set, partially revealing his location. She can brute force the solution by inserting every possible tree node value into the bloom filter and matching the resulting locations. However, because Bob inserts an equal number of arbitrary locations in the table, and because some of the locations correspond to 0 locations (from BF_b0) Alice cannot know which selection values correspond to his wall set nodes and Bob's actual location is still well hidden. Among the $4 * k * l$ table values, only $k * l$ of them correspond to his wall set values. Additionally, Bob can choose to increase the security by simply including more fake location values in the selection table.

Bob In the first protocol, Bob receives from Alice her evaluations of the polynomial and he has the polynomial (which he generated). Because this function is simple and well-behaved he can determine the input necessary to generate the given output. This allows Bob to determine Alice's location. In order to stop this attack we implement a random number r which Alice uses after she has generated her w_i values. $\forall w_i$ we modify $w_i = (w_i)^r \% n^2$. If $w_i = 0$ it remains 0. However, other values are hidden from Bob who does not know the value of r.

In the second protocol, Bob receives from Alice a series of random numbers (C_i)'s and the selection table with her 0 or 1 values. The C_i's reveal nothing, and the selection table reveals the values at the requested locations in BF_a. Because all the bloom filters are the same size and use the same hashes, Bob can brute-force Alice's location by inserting every possible tree node into the filter. Bob knows what value Alice has at each of the locations queried, so he must brute force node input until he finds which nodes set the state of the bloom filter correctly. Fortunately, the size of the selection table is much smaller than the size of the bloom filters, and there are many possible input series that would result in the same selection table query result, thereby hiding Alice's location.

Recent work by authors Olteanu et. al. suggests the idea of co-located users leaking information about the same user to an adversary [74]. This relies on several assumptions a) the victim must process queries from two or more users; b) these users must be ``near'' the victim ,and c) these users must be willing to share information about the victim with the adversary. Due to these strong assumptions, this type of attack is not practical for an adversary.

3.8 Performance Evaluation

In order to evaluate the performance of this system, we ran several experiments which are detailed in the following subsections. The first subsection details the speed of homomorphic encryption on smartphones. In the second subsection, we examine the savings from utilizing a binary tree. In the third subsection, we analyze the running time of our system using both protocols, in order to determine the practicality for end users.

For all of these experiments, we used one or both of two Android OS based smartphones connected to the Internet using WiFi. WiFi is always used in our experiments because of the excessive cost of mobile phone 3g or 4g data plans. The first phone *p1* is an LG-C800 with a 1GHz processor and 512MB of RAM running Android 2.3. The second phone *p2* is a Samsung Nexus S with a 1GHz Cortex-A8 processor and 512MB of RAM. Our Paillier cryptography implementation was borrowed from the University of Maryland Baltimore County [110].

3.8.1 Homomorphic Encryption

In the first experiments we wanted to determine if the weak ARM processors on typical Android cell phones are powerful enough to perform Paillier cryptographic operations in a timely fashion. A small Android application was written and used, which generates a Paillier key, encrypts a user selected value, and decrypts the value. We ran this program five times, with a random value as input, and summarized the results in Table 3.3.

We chose 1024 bit encryption because it is a good balance between security and speed. To get a better estimate of the protocol's running time, we examine lower-level, cryptographic functions, shown in Table 3.4. Here one key is generated and the random value is encrypted and decrypted once.

Key Size	Run 1	Run 2	Run 3	Run 4	Run 5	Avg.
256	123	75	82	74	82	115.3
512	208	139	374	134	177	257.3
1024	1536	1366	1042	933	1158	1176.5
2048	6773	8105	9453	8007	3710	6349.3
4096	91785	44144	31311	26047	81108	46415.2

Table 3.3: Time in milliseconds of generating a Paillier public, private key pair, encrypting a value with the public key, and decrypting that value with the private key.

Run	Key Gen	Encryption	Decryption	Total
1	769	80	147	1006
2	1601	81	147	1830
3	624	80	146	852
4	756	80	151	989
5	1891	80	146	2118
Avg.	1128.2	80.2	147.4	

Table 3.4: Time in milliseconds of generating a Paillier public, private key pair, encrypting a value with the public key, and decrypting that value with the private key on an ARM 1GHz processor

During our protocol, there are only a few dozen encryptions / decryptions performed, each one costing approximately 100ms and there are only two key generations, each costing approximately one second. This shows that 1024 bit encryption is a good balance of speed and security.

3.8.2 Binary Tree Efficiency

In the second experiment, we wanted to determine the savings we achieved by using a binary tree data structure in order to encode the discretized location values. We ran the protocol six times, on *p2*, while varying P_b. If the tree structure affords us a large saving, the protocol will transmit less information and there will be fewer cryptographic operations. In Table 3.5, we can see that, as the policy increases, the number of leaves increase. This is expected, $leaves = ((policy/10)*2)+1$. Bob's wall set grows at approximately O(log(n)) because of the nature of binary trees.

Policy	30	100	300	3000	30000	160000
Leaves	7	21	61	601	6001	32001
Wall Size	3	6	6	10	12	12
Message Length	1638	2867	2867	4505	5324	5324
Packets	2	3	3	5	6	6

Table 3.5: Analysis of wall size as a function of Bob's policy. Also included,is the approximate message length that can be expected for the given policy.

Paillier encryption generates cipher-text that is twice the bit length of the key [86]. We use 1024 bit encryption so we expect cipher-text will be 2048 bits long. Messages are trans-coded, for transmission, in a base 32 numbering system. As discussed in Section 3.11, messages are then broken up into 1000 character packets. Therefore, it is reasonable to see only a few packets generated from thousands of characters.

3.8.3 System Running Time

To get an idea of what users will experience when using our system, we implemented and ran App1 (described in Section 3.5) running on protocol as an Android application. We performed ten queries, for each of eight polices and measured the wall clock running time of the query. The devices used were *p1*, Bob, and *p2*, Alice. In this experiment Alice and Bob are at the same location (37.2701, -76.7119), which was selected randomly. The results of these experiments are summarized in Fig. 3.8. We see a small anomaly; when Bob's policy is 30000 meters the variance of the runtime is smaller. This is due to the variant nature of the running time. In some runs the device will have less or more contention for resources such as network, CPU, and memory. We left these resources uncontrolled to emulate a more realistic use case.

We can see that the protocol runs in a reasonable time. Even a very large policy of 160km only takes $[30 - 35]$ seconds to complete. More reasonable queries take tens of seconds and very small policies require under 15 seconds. Because the user is alerted via an Android notification when a query has completed and query progress is indicated to the user while waiting, this is a reasonable waiting time [70].

To compare with our alternate, Bloomfilter and OT protocol, we performed the same

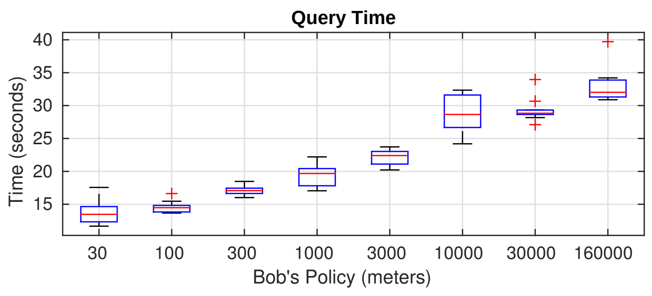


Figure 3.8: Box plot of the total running time of the system. Our system takes on the order of tens of seconds to run depending on the policy.

experiment, and give the results in Fig. 3.9, we see that the alternate system has smaller variance in query time, and is much faster for queries less than 160km. Comparing the mean query time of each system, shown in Fig. 3.10, we see that the alternate system is much faster, except for the largest policy 160km, where the alternate system takes about 50s to finish a query. This is due to the expensive construction of the super path set. Because the super path set contains almost every node in the tree segment, it grows quickly as the policy grows. In light of this, we tested the system on some more modern hardware; a Nexus 4 with 2GB of RAM, a 1.5GHz processor, and a Nexus 7 with 2GB of RAM, a 1.5GHz quad-core processor, both running Android 5.0. In this experiment we are able to complete a query, with a policy of 160km, in only thirty-four seconds on average.

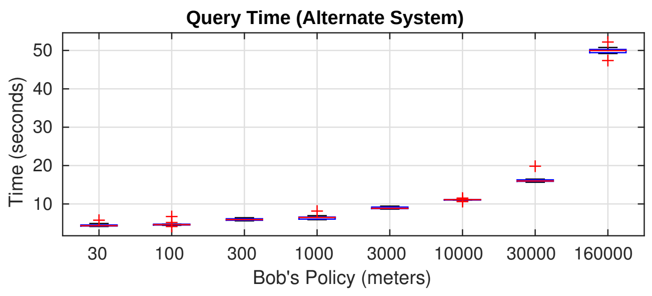


Figure 3.9: Box plot of the total running time of the alternate system.

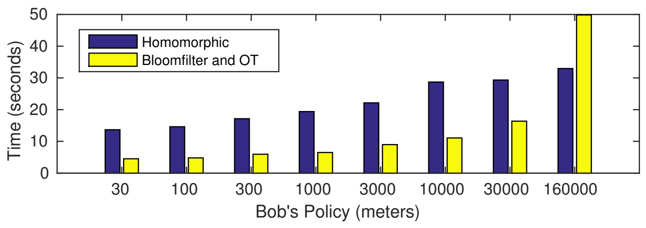


Figure 3.10: Comparison of the query time between the original and alternate systems.

3.8.3.1 Message Transmission Speed

Our evaluation thus far has shown that our second PPD protocol performs better than our first system. In this and the next section we aim to find the bottleneck(s) of our first PPD protocol. We begin by examining the data transmission speed of our Facebook chat message passing system used in the first system. We used one phone, *p1*, to send a random string of characters, to a second phone, *p2*. *p2* immediately responds back with the same characters. *p1* records the time it takes to send and receive the entire message, (round trip time). *p1* also records the time between receiving the first and last packet from *p2*. This shows the overhead incurred by the packet size limitation. We performed the experiment five times for each message length and plotted the results in Fig. 3.11.

Unsurprisingly, messages of 500 or 1000 characters take a short time to transmit; median of 1.6 seconds and 1.8 seconds respectively. As the message length grows larger, the round trip time and packet receiving time increases. In the bottom of Fig. 3.11, we can see plateaus in each pair of sizes (e.g., 2.5k and 3k, 3.5k and 4k, etc). This is due to the 1000 character limit. A 3000 character message and a 2500 character message are both broken up into 3 packets. When the message sizes are very large, > 5000 characters, the volatility increases greatly. We can attribute this to the fact that the Facebook message system is not designed to send such high volumes of data. Humans typically converse at a few dozen characters per second.

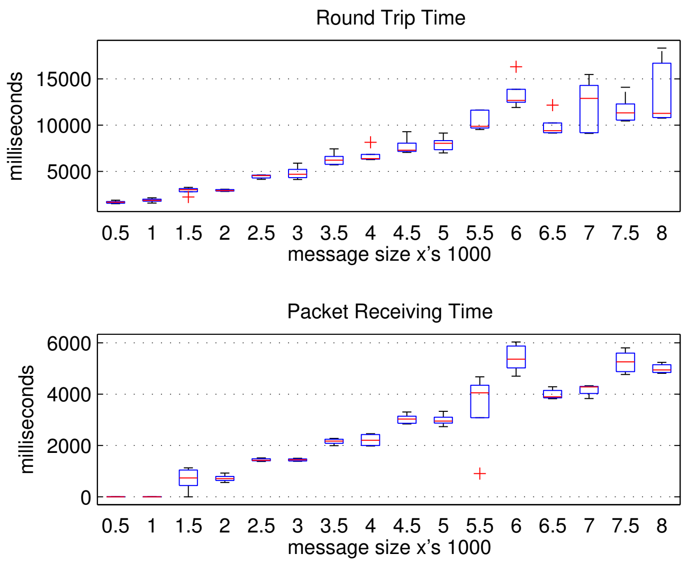


Figure 3.11: Top: Round trip time, in ms, of sending a random message. Bottom: packet receiving time, in ms, between receiving the first and last packet. For each size, a random message was generated and sent five times (round trip).

3.8.3.2 Message Size

To evaluate how much data is sent during a typical query using protocol one, we ran sets of ten queries varying Bob's policy, similar to our experiment in Section 3.8.3, on *p1* and *p2*. During each query we recorded the total bytes transmitted and received by Bob, (*p2*). It is safe to assume that the traffic on Alice is accurately reflected at Bob, because Bob and Alice are the only two parties communicating. The results of this experiment are summarized in Fig 3.12.

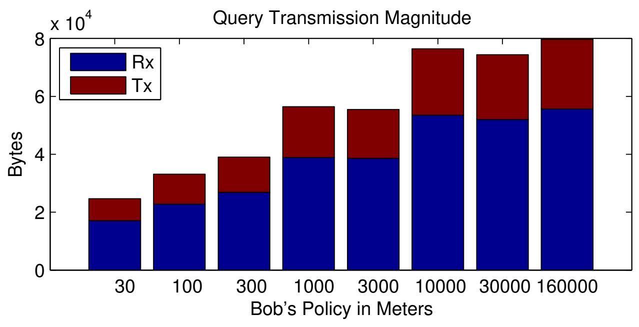


Figure 3.12: Transmission size of a query as measured by Bob.

Because the Android API only provides a measure of total bytes transmitted or received, our results include some small background data transmissions from other miscellaneous applications and services (e.g., email sync, automatic application updates). Also, the size of the cipher text transmitted varies slightly. Different keys will generate different cipher text values, requiring more or less characters to represent, given the same input. To capture these variations, the standard deviation of each set of ten queries is presented in Table 3.6. Our system takes only tens of kilobytes to perform a query, even for very large policies. The standard deviation, which is a small fraction of the bytes transmitted, demonstrates that our protocol is stable in data transmission size.

Policy	Rx Standard Deviation	Tx Standard Deviation
30	2441.8	54.4
100	1257.7	79.4
300	1283.7	113.9
1000	1338.2	223.6
3000	1776.4	133.8
10000	3186.4	122.7
30000	1877.0	458.9
160000	2905.2	404.8

Table 3.6: Standard deviation of the transmission size.

Combining all the results from Section 3.8, we conclude that the largest bottleneck in our system is message transmission, which takes up the vast majority of the query time. As evidenced by the message size, and the experiments in Sections 3.8.3.1 and 3.8.3.2, this is because the Facebook Chat protocol is slow, not because our system sends large amounts of data. Although all of our experiments utilize the mobile phones' WiFi connections, our system is practical for mobile 3g or 4g data plans. Message transmission speed is limited by the speed of Facebook Chat, only roughly 8k-bits per second. We can easily increase this speed by using Facebook Chat to initiate a query, allowing Alice and Bob to exchange references. We then offload all message sending to a third party server. Here the bottleneck for data transmission would be 3g speeds, which are at least 384k-bits per second [117].

3.8.4 Scalability Analysis

Our system is concerned with one user querying the location of another. However, users may want to learn their friends' location as soon as they are nearby (within P_b) as described in App2 in Section 3.5. We face many challenges when trying to scale our system up for many users in use cases like these. Each query takes approximately 40 seconds and 80 KBytes in the worst case using our first protocol, which means Alice can send queries at a maximum rate of 90/hr. and 7.03MB/hr. In the most naive approach, where Alice queries one user at a time repeatedly, it will take over three hours in the worst case for her to be alerted when a friend is nearby if she has 300 Facebook friends.

To scale our system we can make several key improvements. Firstly, we note that Alice can query many of her friends in parallel. That is, she can send messages using Facebook chat to query all of her friends simultaneously. In this way, Alice doesn't wait for each query to finish before starting the next one. If she does this, several other factors become the bottleneck. Alice will need to to generate one key for each friend, and a second key if the query is successful to send the location coordinates. To save time, we can pre-compute many keys and store them locally. Keys are only 1024 bits long, so a table of 10,000 pre-computed keys will only take up approximately 1MB of local storage and access to them will be instant. In our protocol, in the worst case, Bob will have a policy of 160km, which means he will generate a wall set of twelve nodes, or 5k characters. Bob will send this to Alice. Using our third party server for communication, and opening many connections, Alice can accept all of this data in parallel from each of her Facebook friends. However, her 3g bandwidth will limit the transfer to roughly 384kbit/s, the minimum speed set forth by the ITU [117]. So sending these encrypted coefficients will take 31 seconds with 300 friends. Once Alice has received these coefficients, she reconstructs and evaluates the polynomials which require multiplications of all the values. The time to perform these multiplications is negligible for the BigInteger java library with relatively small values. Once the values are calculated, they're sent back to Bob who decrypts to see if any of the results are zero. This incurs another 31 seconds of transfer time by opening multiple connections. Because each Bob is doing the decryption independently, it's effectively in parallel and only adds roughly one second to the running time. The data transfers, multiplications, and decryptions are all done twice. Once for latitude and again for longitude. Therefore, it will take between two and three minutes to complete the protocol for 300 friends.

Once this is completed, Bob knows if Alice was nearby and he can use her public key (sent to him in the final transmission) to transmit his actual location values. In the worst case, all of Alice's friends are nearby, and Alice must receive another set of 300 messages. Fortunately, the final location messages are small, only taking up about 1k characters. So this transfer only takes about one second. By showing that protocol one is able to achieve this result, there is little need to show that protocol two, which is more efficient per query will perform equal if not better. Therefore, it is safe to assume that both protocols will be able to serve this application in a practical or ``fast-enough'' time.

3.9 Conclusion

In this chapter we extend our previous work [71], which provides private proximity detection through private set intersection. Our previous work establishes this using Paillier homomorphic encryption. Our implementation of this system is done on the Android platform and uses the Facebook online social network for blind message passing. We also present a second private proximity detection protocol. The new protocol uses oblivious transfer, bloom filters, and a third party message passing server to greatly improve on the query speed. It is our hope that these two protocols can be used to build popular social location based services in order for ordinary users to enjoy the convenience of proximity detection and location services without worrying about the implications of strangers, or online service providers, gaining access to their location data. Our systems can be deployed as a layer in another location based service. The source code of both protocol implementations is freely accessible online for users to install or make improvements on [72].

Chapter 4

Ultrasound Proximity Networking on Mobile Devices

Our second contribution is a system that provides proximity networking based on ultrasound in smart devices. By improving security and privacy of communication at the physical layer, we allow users to share information, such as URLS, encryption keys, or mobile payment information on commodity devices.

4.1 Background

With smart mobile devices now so common, users are beginning to demand a simple, secure way of exchanging sensitive information with those around them. A key application example is sharing a phone number. While one user may be willing to tell another her number, they may not want random third parties and eavesdroppers to overhear it. To support this, and a wide variety of other applications such as exchanging email addresses, sharing pictures and links, exchanging credit card numbers for mobile payment systems, authenticating access to a physical space, and controlling Internet of Things (IoT) devices, we propose the use of ultrasound on commodity, ubiquitous smart mobile devices (similar to [12]) for data transmission. We aim to provide this functionality, also known as proximity based networking, to users in a way that is fast, easy, and does not require special hardware beyond users' phones.

Currently, proximity based networking is achieved through the use of Near Field Communication (NFC) radios [19], Bluetooth (BT), or WiFi (ad-hoc mode / WiFi direct). One major drawback of NFC is that it requires special radio hardware in users' devices and adoption of this hardware has been steady, but slow [91]. It is also known to have some security problems [32, 128]. WiFi direct and Bluetooth both are more common on smart devices, but they are relatively long range, which makes eavesdropping and man in the middle attacks easier. Bluetooth implements a manual pairing protocol at a low level, which cannot be easily avoided and WiFi direct necessitates special hardware (to host the connection), and the users must not be associated with any access point (AP). Additionally, the WiFi spectrum, 2.4GHz, is becoming increasingly crowded. However, the key drawback of these existing technologies is that they are omni-directional and therefore require some sort of pairing or private key exchange. Our proposed ultrasonic audio modem has inherently narrow propagation, due to the nature of high frequency sound, which allows for intuitive aiming of the transmission to the intended receiver. It does not require any pairing protocols, or prior key-exchange / key-management making it ideal for effortless sensitive information exchange.

Utilizing an ultrasound modem, on smart mobile devices, lends itself naturally to this problem for several reasons. First, the necessary hardware (speaker and microphone) and sample rate (44.1kHz) are already ubiquitous on commodity smart mobile devices meaning that our solution does not require any change to the industry, and supports the bring your own device paradigm. Second, ultrasound has a fundamental and intuitive security edge, in that it is directional, cannot pass through walls, and it has a limited, easily configurable transmission range of a few centimeters allowing users to aim their transmission at the intended receiver. No encryption key management is necessary. Third, ultrasound is inaudible by definition and therefore, non-disruptive to users.

In implementing our ultrasound modem we face several challenges. First, we must modulate sound waves in a way that supports fast data transfer, yet minimizes audible noise. This turns out to be a significant challenge because simple carrier wave modulation manifests as audible sound. Our second challenge is that typical devices have relatively poor frequency response in the ultrasonic spectrum due to the low cost and compact speakers and microphones they use. This means that we cannot reliably recover the amplitude of specific frequencies with fine granularity. Third, we find that recovering the phase is quite difficult, because we operate near the Nyquist frequency. Our system is the only academic work, to the authors' knowledge, to rely exclusively on ultrasonic tones for high speed communication between smart devices, with the goal of remaining inaudible. It achieves the highest throughput of any existing approach.

The current state of the art in this domain is a system titled ``Dhwani" [88], which attempts to transmit data using sound as a wireless communication medium as well. The authors use quadrature phase-shift keying (QPSK) on a 6kHz carrier achieving 2.4kbps in the best case. Our system operates in the ultrasonic spectrum (to avoid disrupting the user), which introduces several challenges. QPSK results in a very high error rate in this spectrum, due to the limited sample rate on commodity devices. Our system instead incorporates amplitude and binary phase shift keying, combined, using a novel analytic phase recovery technique to achieve a higher throughput. We also go to great lengths to ensure that our modem does not create disruptive or irritating noises that arise from modulation.

The design of our modem has several key components. We design schemes to decode both amplitude and phase changes. We make many key design choices to reduce transmission errors, and we experiment with a wide variety of modulation techniques to remain as inaudible as possible. An extensive evaluation is presented that demonstrates the speed, audibility, and energy use of our modem. Although our number one goal is private exchange of sensitive information, it is our hope that our system can be used by a wide array of ubiquitous computing applications in the future, such as sharing a link with several members at a meeting, neighbor discovery, home automation, and gaming. Because our system assumes users have common (already ubiquitous) devices, it is very valuable in many ubiquitous computing applications where assumptions about special hardware cannot be made. Our contributions can be summarized as the following:

- We propose the use of ultrasound as a means of data transmission for proximity networking on commodity mobile devices. We are the first to closely examine the different modulation / demodulation schemes for high data rate communication while remaining inaudible.
- We design and implement several novel and non-trivial techniques to overcome a variety of challenges working in the ultrasound spectrum including utilizing orthogonal frequency devision multiplexing (OFDM), volume fading for inaudibility, adaptive amplitude recovery in the face of poor frequency response, analytic phase recovery to overcome sub-sample syncopation, and dynamic sender power to control multi-path and increase security.
- We present an extensive evaluation of our system based on our prototype implementation working in the inaudible spectrum [18kHz, 21.5kHz]. We show our system is low energy, low noise, fast, and practical in several applications. Our best effective transmission rate is 6.1kbps, with 0.2% bit error rate (BER), before error correction.

We propose several techniques to overcome the challenges we face in implementing this system, which are highlighted throughout this chapter. We use *phase spreading* and *volume fading* to reduce audible noise. To overcome non-uniform frequency response we propose *dynamic amplitude recovery* and in response to the poor phase resolution from the limited sample rate, we propose *analytic phase recovery* and *phase calibration*. Finally, we show that multi-path can be reduced and security can be enhanced by using *dynamic sender volume*.

4.2 Challenges

In this section we detail four technical challenges in implementing our system on common smart mobile devices, each of which we overcome with a specific technical contribution. First, transmitting data while remaining inaudible. Second, we face poor frequency response in the ultrasonic spectrum. Third, phase recovery is difficult due to limited sample rates. Finally, multi-path effects (i.e., echo) are a challenge, because ultrasound is highly reflective.

Inaudibility Designing a system that achieves high throughput, and low audibility is a challenge. The common method of data transfer is to produce a sine wave with frequency f and phase θ. The sine wave's frequency, amplitude, and phase are all changed over time to encode information (modulation). This sine wave is sampled, digitally as $S = s_1, s_2, ..., s_i$ using Eq. 4.1, where the time t is derived from the sample rate F_s and index i, ($t = \frac{i}{F_s}$).

$$s_i = sin(2 * \pi * f * t + \theta) \tag{4.1}$$

The fundamental modulation schemes, in which amplitude, frequency, or phase are modulated separately, are referred to as binary amplitude shift keying (BASK), binary frequency shift keying (BFSK), and binary phase shift keying (BPSK) (illustrated in Fig. 4.1). We quickly discovered that these modulation schemes produce audible noise, even if the carrier sine wave is above the audible frequency threshold (e.g., higher than 20kHz), due to the modulation pattern. This can be most easily visualized in the case of BASK. In Fig. 4.2 we plot a BASK signal which encodes a repeating bit pattern. The periodic amplitude changes generate an audible ``artifact'' wave, shown in bold, equal to the envelope of the signal. If the bit pattern is regular, as is the case in our example, the artifact wave will be a specific frequency. When the bit pattern is random, which is the case for our intended application, the artifact wave is similar to white noise with many random frequencies.

Audible artifacts are also caused by any instantaneous changes in the waveform (near sample number 41 in the plots of Fig. 4.1). The speaker's diaphragm physically moves

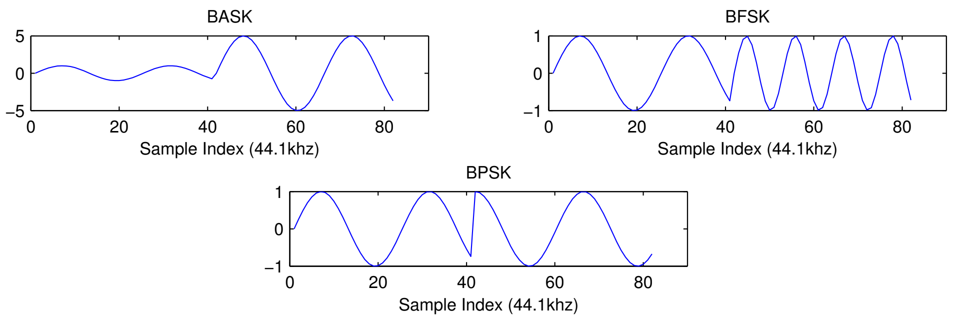


Figure 4.1: Example sine wave modulated in amplitude (left), frequency (middle), and phase (right).

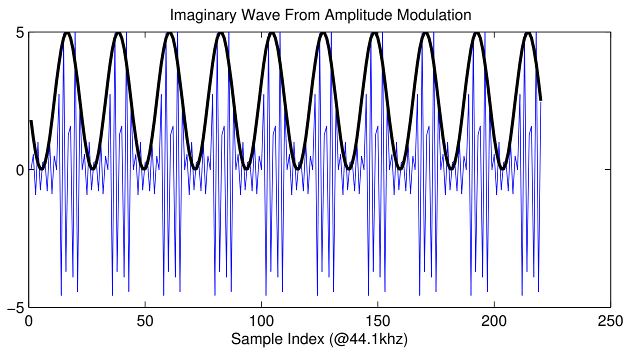


Figure 4.2: Imaginary wave (black) generated from ASK

synchronously with the path of the sine wave (in one dimension), so discontinuity causes the speaker to produce on audible ``click'' noise. To overcome this challenge, instead of encoding bits in the time domain, we focus instead on encoding information in the frequency domain, using orthogonal frequency division multiplexing (OFDM), which has much less frequent discontinuity. We also utilize two unique techniques to further reduce noise called *phase spreading* and *volume fading*.

Poor Frequency Response The frequency response of cell phone speakers and microphones is highly volatile. The physical size of cell phones limits their speaker and microphone quality and as a result, audio signals are heavily distorted. This is compounded by the complex interaction between frequencies when the speaker is reproducing many simultaneously. To demonstrate this problem, we generated frequencies, 50Hz apart in the range [18kHz - 22kHz] and [5kHz - 8kHz], one at a time on an Android smartphone (model Nexus S) with a power level of $\frac{14}{16}$. We recorded the sound using a second Android smartphone (model LG-800) and we plotted the recording, in the frequency domain, in Fig. 4.3. These experiments show that the smartphones we tested are much better at producing tones on the lower end of the spectrum (in line with [88]).

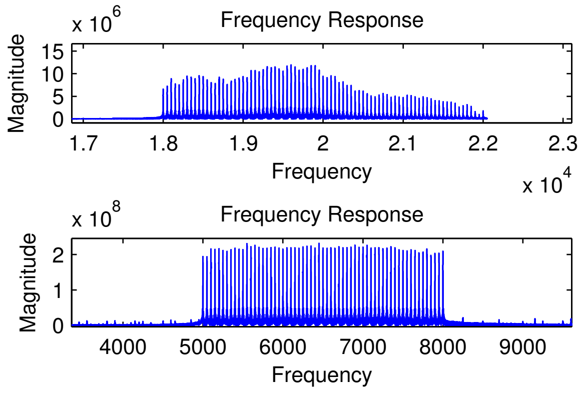


Figure 4.3: Frequency response of typical cell phones.

Recovering Phase Accurately Recovering the phase accurately is difficult for three reasons. First we will be operating above 18khz, but the sample rate of our devices is limited to only 44.1kHz. Because our frequencies are very near the Nyquist frequency (22kHz), there are only a few samples in each period. Therefore, when reading the phase of a signal, an error of one or two samples, or even sub-sample differences, results in a wildly different phase. An illustration of this phenomenon can be seen in Fig. 4.4. We overcome this problem using a hail signal, to approximate the starting point, and a novel technique called *analytic phase recovery*, which can recover the phase with sub-sample accuracy.

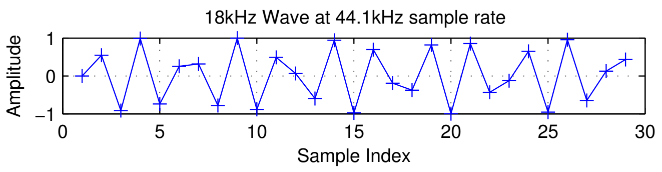


Figure 4.4: Due to the high frequency wave, and limited sample rate, the phase difference between adjacent samples is very large. For example, the phase difference between samples three and four is nearly 2π.

We also notice that the phase is typically corrupted during transmission. Similar to the non-uniform frequency response, even if the sender encodes several sub-carriers with the same phase, the receiver will decode many different phases. This is partially an artifact of the Fourier transform we use to decode the signal, because the transmitted signals are usually not symmetric. We refer to this as ``phase drifting." We overcome this challenge by utilizing a technique called *phase calibration*, which we detail in the Analytic Phase Recovery section. This problem is exacerbated when many different sub-carriers are present with different amplitudes and phases. The sub-carrier frequencies interfere with one another in the air, corrupting the signal. We explore many modulation schemes to find the one with the best throughput in the face of this challenge.

Multi-path The final challenge we face is that of multi-path. Because we are operating at high frequency, the sounds our modem emits are highly directional, due to the small speaker size on our devices, and reflect easily off of surfaces. We solve this problem by carefully controlling the sender volume so that it is strong enough to reach the receiver, but weak enough that multi-path is too quiet to have a significant effect.

4.3 System Design

Our system exists as a modulator and demodulator. One or both of these may be running on some hardware (For example; a base station kiosk, a mobile phone, or a building wide centralized system). A device may be a sender or a receiver or both. But, the receiver must have a microphone and the sender must have a speaker. A high level view of our system can be seen in Fig. 5.2.

At the receiver side, sound comes in through the device's microphone and is first parsed by the hail listener, which finds the approximate starting point of the data signal among background noise. The raw data is then broken into chunks of size n and passed to the amplitude (ampD) and phase (phaseD) demodulators, which recover the bits using the Fourier transform (FFT) module. The resulting binary data is then interpreted at the

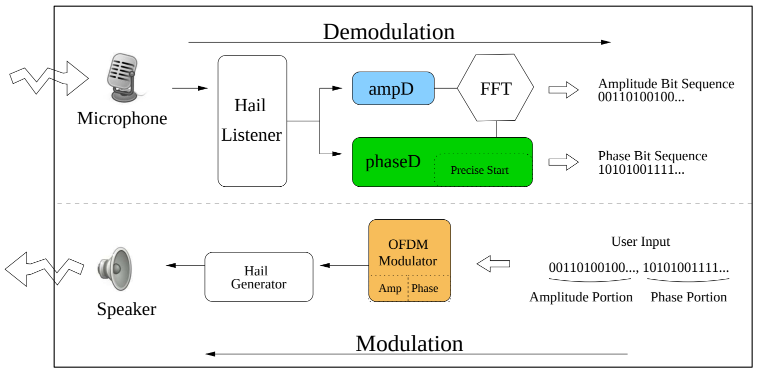


Figure 4.5: Conceptual view of our ultrasound modem

application level. In the case of sending data, the user inputs some data to be sent, which is broken into chunks of 65 amplitude and 65 phase bits, which are passed to the OFDM modulator. The OFDM modulator generates an ultrasonic audio signal, by modulating the amplitude and phase of each sub-carrier appropriately. Then, the system adds a linear chirp hail signal to the beginning and transmits the audio using the device's speaker.

Operating Spectrum We chose spectrum 18khz - 22kHz, because it is inaudible to humans, and below the Nyquist frequency for most smartphones. Also, previous work has shown that background noise in this spectrum is relatively low [43, 88].

4.3.1 Quiet Transmission using OFDM

Due to the constraints outlined previously, we cannot modulate a carrier wave directly. Instead we work in the frequency domain. We generate data segments, each of which is the sum of many sub-carrier frequencies. We compute each sample in a segment according to Eq. 5.1.

$$s_i = \sum_{f=18kHz}^{22kHz} sin(2 * \pi * i * \frac{f}{F_s} + \theta) \tag{4.2}$$

Using the Fourier Transform (FFT), we are able to determine the frequencies present in a given set of samples $S = \{s_1, s_2, ..., s_i\}$ and we can recover each frequency's amplitude and phase, which encode the data. This is commonly referred to as orthogonal frequency division multiplexing (OFDM). In order to recover the amplitude and phase of frequencies with a granularity of 50Hz we need $n = 882$ sample segments (20ms at 44.1kHz sample rate with no guard interval) as input to the FFT. It is important to note that a perfect power of two is not necessary thanks to the interpolation done by the MATLAB® implementation of the FFT we used. This gives us 70 sub-carriers to encode data. Our sub-carrier construction is shown in in Fig. 4.6, which begins at 18.05kHz because 18kHz is a calibration frequency. We then combine the sub-carriers (on the right side of Fig. 4.6) using Eq. 5.1. In an attempt to improve throughput, we tested some modems which use 750 sample

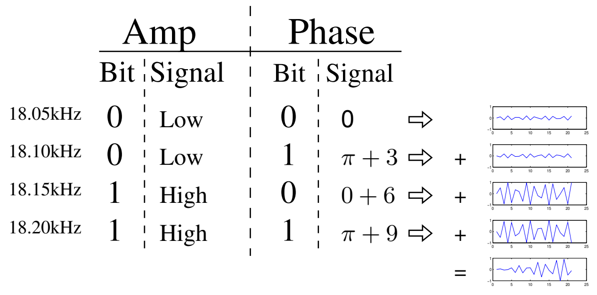

Amp
Phase
Bit
Signal
Bit
Signal
18.05kHz 0 Low 0 0 ⇨
18.10kHz 0 Low 1 $\pi + 3$ ⇨ +
18.15kHz 1 High 0 $0 + 6$ ⇨ +
18.20kHz 1 High 1 $\pi + 9$ ⇨ +
=

Figure 4.6: OFDM modulation scheme used. Each pair of bits determines the amplitude and phase of the sine wave for the corresponding sub-carrier.

segments, which are shown in Table 4.4 in our evaluation. We choose not to implement any clear cycle codes (cycle prefix) or similar techniques, because of the time they take, which would reduce our throughput.

Calibration Frequencies Our modem uses frequencies 18kHz, 19kHz, 20kHz and 21kHz as calibration frequencies, which carry constant amplitude (100%) and phase (π). We use these frequencies to account for the non-uniform frequency response and phase drifting challenges we mentioned previously. This leaves 66 sub-carriers to encode data, of which we use 65, achieving a theoretical bit rate of 6.5kbps.

Phase Spreading Fortunately, by utilizing OFDM, there is much less audible noise created. However, there are still some audible artifacts, because at the transition from one data segment to another, there is a small discontinuity. This occurs between every data segment (roughly 50 times per second), and produces an audible click, of varying amplitude, depending on the degree of discontinuity. This discontinuity is largest when many sub-carriers have a high amplitude and equal phase. In this situation, they experience additive interference in the first few samples, resulting in a large peak in the time domain. This can be seen on the left side of Fig. 4.7. We can avoid this problem by changing the phase of each sub-carrier frequency so that they do not combine at the beginning of the segment strongly. We propose a novel technique called *phase spreading* in which we add three to each phase iteratively, as shown in Fig. 4.6. By doing this, we guarantee that the frequencies interfere destructively in the first few samples, as can be seen on the right side of Fig. 4.7. This reduces the audible noise, and the demodulator can easily compensate for this change, since it is known.

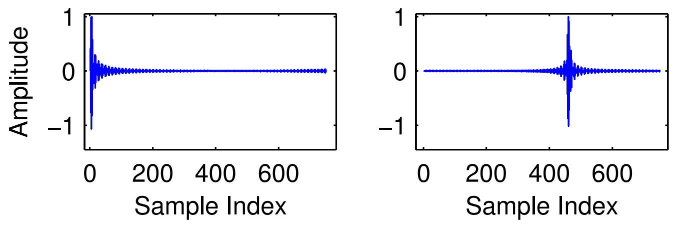


Figure 4.7: Left: Signal Before. Right: After phase spreading.

Volume Fading Envelope To further smooth the discontinuity between segments, the modulator applies an envelope according to a simple equation: $s_i' = \sqrt{\frac{1}{a} * s_i}$. We call this technique *volume fading*. Here a determines how many samples to fade. Choosing a larger a value decreases the noise created, but alters the signal, increasing the error rate when recovering the bits. We found $a = 5$ to be a satisfactory trade-off.

Hail Signal Our hail signal is generated by the *hail generator* module for the receiver to a) realize data is going to be transmitted in the near future, and b) find the approximate starting point of the data-modulated signal. To find the starting point with as few samples as possible, to maximize throughput, we design our hail signal as a 300 point, linear chirp, sine wave from 18kHz to 19kHz. It fades in and out in amplitude to minimize audible noise and can be located with high accuracy due to the frequency change. Empirically, our hail signal is able to achieve single sample accuracy. While more erratic hail signals may be used to achieve better accuracy, ours is almost completely inaudible. We also tried a simple 18kHz static frequency hail signal, which had much worse accuracy (about 20 samples).

4.4 OFDM De-Modulation

The demodulator is responsible, at the receiver, for finding data-modulated signals in the random background noise and decoding them. In Fig. 4.8 we can see an entire packet transmission (top), and the first data segment (middle). Using the FFT module, the amplitude and phase of the sub-carrier frequencies in the segment are recovered (third and fourth plot in Fig. 4.8).

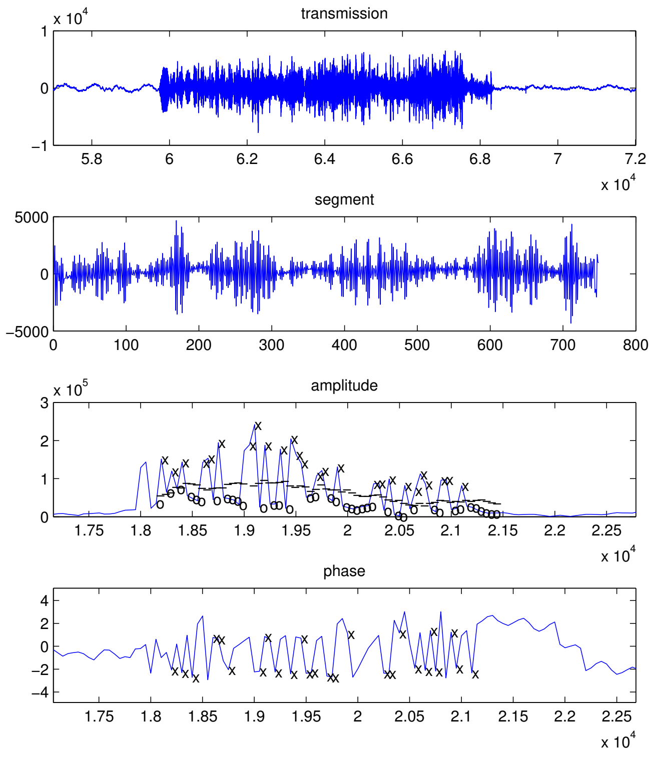


Figure 4.8: Transmission (top) shows the entire raw data transmission. Segment (middle) shows the first segment after the hail signal. Amplitude (middle) shows the amplitude of the frequencies (i.e., abs(FFT(segment))). An x marks ``1'' bits and a zero marks ``0'' bits. Phase (bottom) shows the phases of the the frequencies. An ``x'' marks the places where the phase was read.

Hail Listener The *hail listener* module, in the demodulator, continually listens to the raw audio signal waiting for the volume to become greater than a pre-defined threshold. When this is occurs, the hail listener runs a 300 sample, sliding window dot product (cross correlation) between the received microphone data and the known hail signal. The cross correlation between these two signals will be highest when they are synchronized. We use this maximum as the starting point for the first data segment. The starting point of the subsequent data segments are found using the known segment size $n = 882$ as an offset. To recover the phase, we need better accuracy, which we detail in our analytic phase recovery section.

4.4.1 Adaptive Amplitude Recovery: ampD

The amplitude demodulator takes a segment, in the frequency domain, as input. As mentioned earlier, and is clear in Fig. 4.8, the amplitude of each sub-carrier is non-uniform and therefore difficult to decode. To overcome this challenge, we design a technique called *adaptive amplitude recovery* which attempts to follow the trend of amplitudes. It first creates two queues, *up-queue* and *down-queue*, which contain the values of the previous α ``up'' amplitudes and ``down'' amplitudes respectively. The first frequency, 18kHz, is a calibration sub-carrier statically held at 100% amplitude. We use this reading to bootstrap our demodulator, placing the value in *up-queue* and placing a zero in the *down-queue*. From this point on, each frequency is read and placed in the *up-queue* if the reading is greater than $thresh$, which is defined below. Values less than $thresh$ are placed in the *down-queue*.

$$thresh = [(\textit{up-avg} - \textit{down-avg}) * \gamma] + \textit{down-avg} \tag{4.3}$$

Here **-avg* is the average of the corresponding queue. γ is a parameter (between zero and one) that determines where, between the $up - avg$ and $down - avg$, the threshold should be placed. The intuition here is that the threshold is placed somewhere between the averages, which will follow the general trend of the varying amplitudes. At the end, frequencies in the *up-queue* correspond to ``1" bits and frequencies in the *down-queue* correspond to ``0" bits. The threshold can be seen in Fig. 4.8 as the dashed line in the amplitude plot. As the sub-carriers are processed, the queues are emptied to guarantee they always contain only the most recent α readings.

Sometimes the ``up" amplitudes are abnormally high, as a result the threshold is pulled up very high and none of the subsequent ``up" bits are recovered correctly. To fight this phenomenon, we implement a third parameter, β. When an amplitude is recognized as ``up", it is only placed in the *up-queue* if it is $< \beta * avg($*up-queue*$)$. The system always records a ``1" bit, but β helps naturally remove outliers from shifting the *up-avg* significantly.

In our system, through manual investigation, we choose $\alpha = 2$, $\beta = 10$, and $\gamma = 0.35$, because they give satisfactory results on the variety of devices we tested with. Choosing and exchanging these values automatically is left for future work.

4.4.2 Analytic Phase Recovery: phaseD

For phase modulation, we have attempted to implement two, three, and four level modulation. In order to decode phase, we split the available phase reading area into the respective number of sections. Four level modulation is illustrated in Fig. 4.9. In this scenario, each level represents two bits. If the phase of the corresponding sub-carrier falls anywhere in the level, we can decode the corresponding bits.

Recovering the phase is straight-forward, we simply read the angle of each sub-carrier frequency from the Fourier transform of the segment. However, to decode the phase accurately, we must find the starting point with sub-sample accuracy. This is due to the problems outlined previously in the challenges section. The problem is exacerbated, because the speaker of the sender and the microphone of the receiver are likely not synchronized, resulting in sub-sample differences. To tackle this challenge, we use our novel *analytic phase recovery* technique. The *modulator* sets the phase of 18k and 19k to be the same,

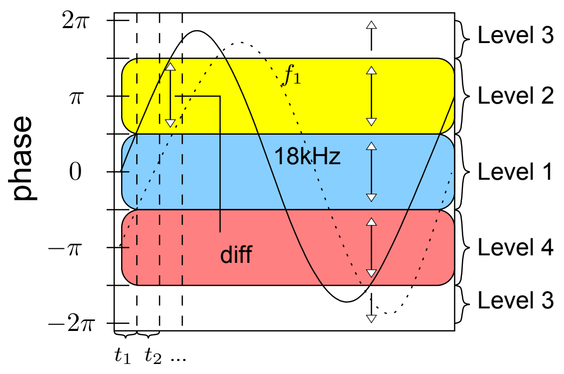


Figure 4.9: Depiction of four level, phase demodulation.

because they are calibration sub-carriers. The *phaseD* module first finds the phase of 18k (θ_{18})and 19k (θ_{19}) from the data segment directly. It then calculates the phase at 1000 sub-sample points before and after the starting point guess from the hail signal listener. At each point, we use the sine wave formula, Eq. 4.1 to find the phase. The sub-carrier frequency f is known, so we substitute in the relative time t and solve for θ. Each time $\Delta = abs(\theta_{18} - \theta_{19})$ is calculated. We have found the starting point when the difference between the two phase readings, Δ, is minimized. The phase of the sub-carrier frequencies can then be calculated the same way; substituting the appropriate frequency and the starting point t which is now known (the t that produced the smallest Δ).

In contrast, Dhwani [88] is able to decode phase modulations without finding the precise starting point, because they're working at a much lower carrier frequency (i.e., 6kHz - 7kHz). At these frequencies, there are three times as many samples per period so the phase reading difference from adjacent samples is small and sub-sample differences are negligible.

Phase Calibration As we move across the sub-carriers reading phase values, we note that the phase values tend to drift due to noise, the non-symmetric input to the FFT module, and distortion of the transmitted signal. To account for this drift, we propose a technique called *phase calibration*. We set the calibration sub-carrier phase values to constants in the modulator. To decode the phase we find $diff = \theta_c - \theta_f$, where θ_c is the phase of the nearest calibration sub-carrier and θ_f is the phase of the desired sub-carrier. Based on this difference, we can decode the phase. Phase readings are always between 0 and 2π, but we may sometimes calculate a negative value of $diff$. To solve this problem, if $diff$ is less than zero, we add 2π. This process is illustrated in Fig. 4.9 for QPSK, which has four possible levels each $\frac{1}{2}\pi$ wide, corresponding to two bits each. Unfortunately, we found empirically that using four phase levels produced too much error and instead utilize only two in our final modem. This is due to the limited sample rate on our devices.

4.4.3 Error Correction Coding

We can expect some errors when transmitting. We use hamming error correction coding (ECC) to correct single bit errors and detect multiple bit errors in each block [115]. Hamming codes allow for blocks of $n = 2^r - 1$ bits, which contain a message of $k = 2^r - r - 1$ bits, and r detection bits. Choosing r intelligently depends on the error rate, which we investigate in our evaluation.

4.4.4 Alternative Modulation Techniques

When designing our system we put great effort into finding a modulation scheme that had low error rate, high speed, and made minimal audible noise. We put in a lot of engineering work in order to extract as many bits as possible per second. Below we present the theoretical bit rates of some alternative modulation schemes using the best possible circumstances; 80 sub-carrier frequencies with 58 segments per second (750 sample segments @ 44.1kHz sample rate). In our evaluation, we show the actual BER (Table 4.3) and effective throughput (Table 4.4).

4.4.4.1 Binary Shift Keying (BSK)

Our first approach was binary phase shift keying BPSK, and binary amplitude shift keying BASK, (separately). Both of these systems are theoretically capable of 58 segments $* 80$ bits $= 4.6$kbps.

4.4.4.2 Ternary Phase Shift Keying (3PSK)

This scheme uses three levels of phase modulation rather than two. Because we can encode on average 1.5 bits in the phase of each sub-carrier, this scheme achieves a theoretical throughput of 6.9kbps.

4.4.4.3 Quadrature Shift Keying (QSK)

QSK, which uses four modulation levels, had very high error rate for phase modulation, and showed an even higher error rate for amplitude modulation. QSK has a potential throughput of 9.2kbps.

4.4.4.4 BASK + BPSK

We treat BSK, 3PSK and QSK as primitives and try to combine them. Our first approach was to perform BASK and then, on the frequencies present, perform BPSK (phase cannot be encoded on a frequency with zero amplitude). Roughly half of the amplitude frequencies will encode a one bit, which means that the same frequencies can encode a phase. On average we can expect to achieve $\frac{1}{2} * 80 + 80$ bits per segment, 6.9kbps. However this system is highly volatile. If the amplitude bits are incorrectly decoded, the system will attempt to decode the *phase* on the incorrect number of frequencies. This results in a mismatch between the decoded and correct bit stream that propagates throughout the data segment. We can solve this problem by inserting placeholder phase bits on the sub-carriers whose amplitude is zero.

4.4.4.5 BASK+BPSK-A

In this scheme we use a low, non-zero, amplitude level to encode ``0" bits, so that we can encode phase on every sub-carrier. The theoretical bit rate is 9.2kbps. This scheme is very similar to our final design, except that it sends 58 segments per second, and attempts to use all 80 sub-carrier frequencies. We show in our evaluation that this results in an unacceptable error rate.

4.5 Implementation and Evaluation

We evaluate our system in five aspects. First, we measure the bit error rate (BER) and perform ECC analysis. Second, we evaluate how sender power, and orientation can be used to enhance security. Third, we do an energy evaluation. Fourth, we show how our system performs in some typical ubiquitous computing application scenarios, such as authentication access to a lab. Finally, we evaluate the audibility of our system. We implemented a prototype of our modem on several android devices summarized in Table 4.1.

In order to run the Fourier transform fast enough, we borrowed a native C++ implementation, [48] and utilized the JNI on Android. Using this native code we are able to run a Fourier transform of 1024 samples in roughly 1ms. To expedite the evaluation, we often used a desktop computer running a small collection of python scripts, and MATLAB® R2013a to de-modulate the packets, measure the BER, and perform ECC analysis. We were careful to ensure that this has no impact on the results of our experiments.

Name	Make/Model	RAM	Proc.	Android Ver.
$p1$	LG/C800	512MB	1GHz	4
$p2$	Samsung/Nexus S	512MB	1GHz	4
$p3$	Motorola/MB525	512MB	800MHz	2.3
$p4$	Samsung/Galaxy Nexus	1GB	1.2GHz	4
$p5$	LG/Nexus 4	2GB	1.5GHz	4.4.4

Table 4.1: Devices used for evaluation. We included the Motorola MB525 which is circa 2010 to investigate how older and lower quality phones perform.

4.5.1 Transmission Error

To get a basic idea of the error rate of our system, we performed ten transmissions of ≥ 1000 random bits at various distances, locations, and orientations. These results are summarized in Table 4.2. (In this table, *inline* refers to holding the phones upright one in front of the other and *oriented* refers to pointing the speaker of the sender directly at the microphone of the receiver.) We can see that the error rate is generally acceptable,

Sender	Receiver	Orientation	Location	Distance	Trials	Power	Min Err.	Mean Err.	Max Err.
$p1$	$p2$	inline	inside	15cm	10 x 1kbits	14/16	0.40%	5.64%	9.40%
$p1$	$p2$	oriented	inside	10cm	10 x 1kbits	14/16	0.0%	0.33%	0.80%
$p1$	$p2$	inline	inside	10cm	5 x 100kbits	11/16	2.06%	2.29%	2.96%
$p1$	$p2$	side by side*	inside	15cm	11 x 1kbits	14/16	9.40%	22.64%	32.50%
$p1$	$p2$	side by side	inside	15cm	5 x 1kbits	14/16	3.20%	17.26%	28.95%
$p1$	$p2$	inline	outside	15cm	5 x 1kbits	14/16	0.30%	1.96%	3.50%
$p1$	$p2$	oriented	outside	15cm	6 x 1kbits	14/16	0.20%	2.22%	5.30%
$p1$	$p2$	side by side	outside	15cm	5 x 1kbits	14/16	0.50%	2.40%	6.80%

Table 4.2: Error Rate Evaluation. The high error in rows four and five can be attributed to multi-path effects. The * indicates this experiment was performed with the devices held above a desk, increasing multi-path effects even further.

in many cases below 3%. The long experiments (when 100k bits are transmitted in each trial) we see much lower variance with an error rate near 2%. In the fourth and fifth experiments, when the phones are held side by side inside, we see very high error rate. This is caused first by the poor orientation of the devices. Because the high frequency and relatively small speaker size, the sound is very directional, which means when the devices are not aimed appropriately, the sender will receive a signal that is corrupted by multi-path. This is supported by rows one and two. In the fourth experiment, a desk is present, which increases the multi-path effect.

4.5.1.1 Alternative Modem Schemes

We performed more random transmission experiments using our alternative modem schemes to find the error rate of each. The results, show in Table 4.3, helped guide our decisions when designing our modem scheme. We find that only binary shift keying has acceptable error rates. To evaluate how well our system transmits data over different distances,

Name / No.	Amp	Phase	Amp BER	Phase BER
BASK	$0 \vee 100$	i*3	0.3%	x
BASK (const)	$0 \vee 100$	0	22%	x
BPSK	100	$0 \vee \pi$	x	0.8%
3-PSK	100	$0 \vee \frac{2\pi}{3} \vee \frac{4\pi}{3}$	x	7.4%
QPSK	100	$0 \vee \frac{\pi}{2} \vee \pi \vee \frac{3\pi}{2}$	x	20%
BASK+BPSK	$0 \vee 100$	(i*3) + $(0 \vee 1)$	2.8%	x

Table 4.3: Comparison of the various modulation approaches. Sending power: 12/16, distance: 15cm

we transmitted 1000 random bits, ten times, from *p1* to *p2* at three different locations, five different distances, and using two different orientations. We used our BASK modem and a data segment size $n = 750$, to get a baseline of what errors rates to expect in different scenarios. For each set of ten transmissions at a particular distance/location, we recorded the BER and box-plotted the results in Fig. 4.10. These results show that when the devices are nearby and oriented correctly the error rate is very good and eavesdroppers more than a few meters away, which the user is not oriented towards, have very high error rate. The results in Table 4.2 and Fig. 4.10 together account for 852,000 bits transmitted.

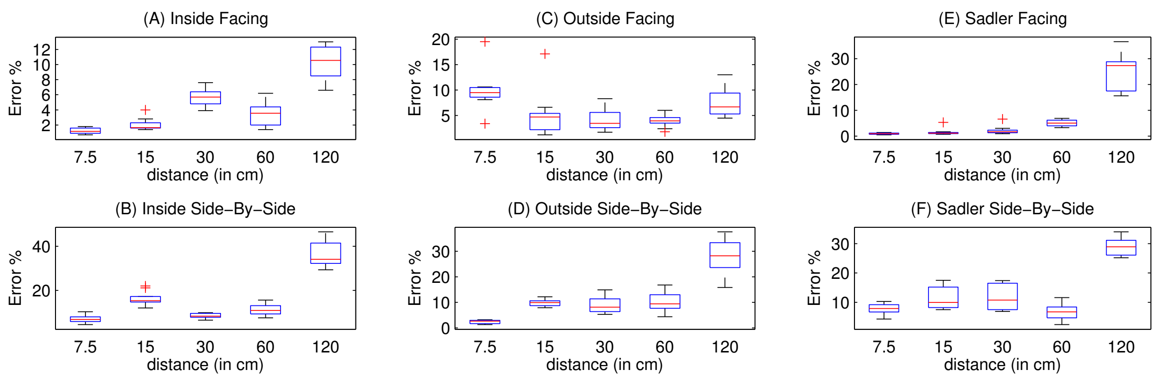


Figure 4.10: Ten transmissions of a 1000 bit, random message at three different locations, two different orientations. ``Sadler'' is public building on our campus.

4.5.1.2 Transmission Power

By increasing or decreasing the transmission power at the sender side, we can control the error rate of the transmission. This is useful to reduce the effects of multipath, and to guarantee that eavesdroppers cannot hear the transmission. To evaluate the effect of sender power, we transmitted 10k random bits five times for each of the 16 possible power levels. In Fig. 4.11 we plot the min, mean, and max error rate for each set. From this figure we see that at very small volume levels (1 - 4), the transmission error is very high with high variance. This is because the signal to noise ratio is very low. At higher sending powers (6-10), the error rate is very low. In the best case, level nine, the mean error rate is 1.27%. Then, as the sending power increases (11-16) the error rate rises again. This can be attributed to two factors. First, as the volume increases, the effects of multi-path increase and second, producing very high frequencies at very high volumes places a lot of kinetic energy in the physical speaker. Because the speaker does not have perfect dampening, there is some residual noise after the signal has ended, caused by the speaker returning to its resting, neutral position. These experiments show that be selecting the correct sending power, we can reduce the effects of multi-path, increase the security, and improve the error rate for the intended receiver.

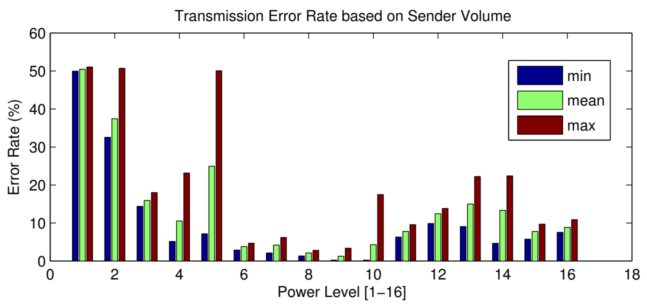


Figure 4.11: Transmission error rate, depending on the sender's volume. Distance: 15cm.

4.5.1.3 ECC Analysis

We selected a typical transmission test from row two, row three, and row eight, of Table 4.2 and calculated the effective transmission rate given various values of r. The results are shown in Fig. 4.12 and Table 4.4. We include a density metric $d = \frac{throughput}{spectrum}$, which is to account for different amounts of spectrum used. Although it might appear that Dhwani achieves the best transmission rate (best density), it is important to note that our system is operating in the spectrum from 18kHz to 22kHz. This spectrum is especially hard to utilize because of the numerous problems outlined in the challenges section. Also, our system is nearly inaudible unlike the Dhwani system [88]. SSCconnect [90] operates on a similar spectrum to our own and, although it is commercial, offers a closer comparison to our work. Their operating spectrum is 17kHz to 20kHz, which we avoided in fear that 17kHz sound may be heard by users with more sensitive hearing.

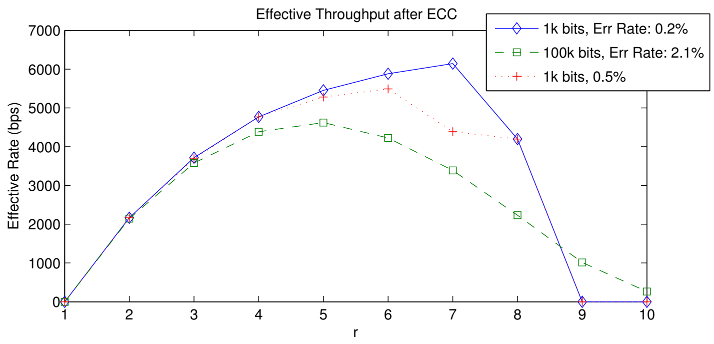


Figure 4.12: Effective throughput after ECC analysis. The three datasets plotted represent a random transmission from our previous experiments in Table 4.2 (rows two, three, and eight respectively).

Technique	Spectrum	PSR	Effective Speed	d
Ultrasound (0.2% Err)	3.5kHz	100.0%	6.1kbps	1.75
Ultrasound (2.1% Err)	3.5kHz	84%	4.6kbps	1.32
Ultrasound (0.5% Err)	3.5kHz	93%	5.5kbps	1.57
Dhwani 8-PSK	1kHz	80%	2.4kbps	2.40
Dhawni QPSK	1kHz	95%	1.6kbps	1.60
Dhawni BPSK	1Khz	100%	800bps	0.8
SSConnect	3Khz	unknown	2.2kbps	0.73
BASK	3.8k	92.5%	3.4kbps	0.89
BPSK	3.3k	95%	3.0kbps	0.91
BASK+BPSK	3.9k	87.9%	4.2kbps	1.08
BASK+BPSK-A ($n = 750$)	3.5k	71.2%	3.9kbps	1.11

Table 4.4: Packet Success Rate and Effective Transmission Speed. Power: 12/16 for the first three rows, 14/16 for the final four. Distance: 15cm. ``Ultrasound'' is the modem as described in Section 4.1

4.5.2 Security

An attacker may try to eavesdrop on the communication between two benign users. We show that our ultrasound modem is not easily susceptible to these attacks for two reasons.

First, it is relatively short distance. Second the transmitted signal is directional, giving the user a powerful, unique, and intuitive way to indicate the intended receiver.

To verify this, we perform seven experiments to exhibit the users ability to inhibit the successful transmission by aiming their device. We transmitted 50k random bits from $p5$ to $p1$, varying the sending power, and the sender's angle (pointing directly at the receiver's microphone is $0°$). The results are given in Table 4.5. In the final experiment, the user places their hand between the sender and receiver, which causes the BER to jump to over 40%.

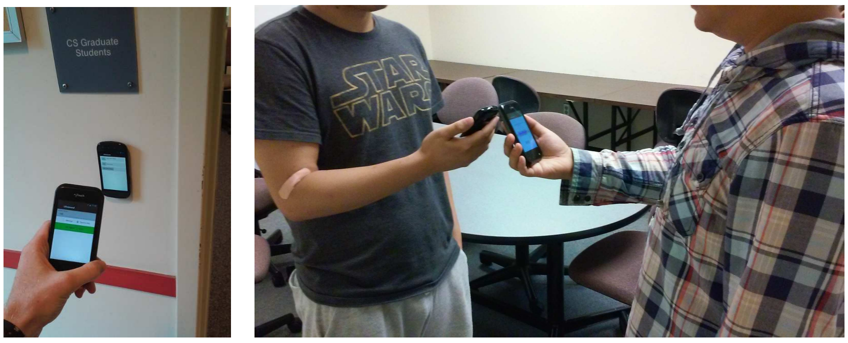
Sending Power	Angle	Bit Error Rate
14 / 16	$0°$	0.59%
14 / 16	$45°$	1.23%
14 / 16	$90°$	2.66%
6 / 16	$0°$	3.25%
6 / 16	$45°$	12.18%
6 / 16	$90°$	14.97%
6 / 16	$90°$	47.34%

Table 4.5: BER with varying angle and sending power. In the final row the user's hand was placed between the sender and receiver.

4.5.3 Ubiquitous Computing Applications

In order to investigate how our system performs under typical usage scenarios, we devised three additional experiments. In the first, two users each hold a device, shown in Fig. 4.13(b), and use the system to send 10k bits, which is roughly equivalent to the size of a small 50 x 50 pixel .jpg thumbnail / contact image with a phone number. We measure an average BER near 1.5% over five trials. For our second experiment, we set up one device to act as a base station, shown in Fig. 4.13(a), to simulate scenarios such as point of sale systems, personal authentication access to a lab, or Internet of things style home automation. The base station in this case is always listening and uses a sliding window root mean squared (RMS) algorithm to detect when a client is sending data. In deployment, the base station is likely to have a direct power connection.

We used this system to transmit 1700 bits (similar to a credit card swipe [116]), repeated forty times and achieved an average BER of 1.31%. In Fig. 4.14 we choose transmissions from each of these two experiments, which demonstrate average error rate, and calculate the effective throughput. Similar to Fig. 4.12 we vary the number of ecc bits r to find the optimum.



(a) Basestation (b) Two Users

Figure 4.13: Practicality Experiments Setup

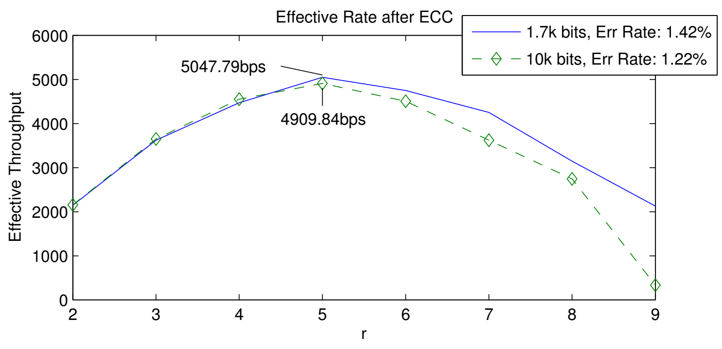


Figure 4.14: Effective throughput for typical transmissions using a base station and for two users holding the devices. Power:12/16, Distance: 15cm. The best packet success rate (PSR) is 92.6% and 90% respectively.

In our third experiment, we evaluate our modem using every combination of $p1$, $p2$, and $p3$ as sender and receiver. For each combination, we send 1000 random bits five times. We measure the BER and report the highest effective throughput after error correction in Table 4.6. These experiments show that our system works well with different devices carrying microphones and speakers from different manufacturers (Motorola, Samsung, LG), which is common in a BYOD environment. Even our slowest effective rate, 2.9kbps, is higher than any competitors' speed. When using the oldest phone, $p3$, we note higher BER rates. Surprisingly, the same device also achieves the best effective transmission of 6.4kbps as the sender.

Tx / Rx	BER.	r	PSR	Effective Rate
$p3$ / $p2$	0.1%	9	100%	6.4kbps
$p3$ / $p1$	0.34%	6	98.7%	5.8kbps
$p2$ / $p1$	0.46%	6	95.2%	5.6kbps
$p2$ / $p3$	4.9%	4	85.2%	4.0kbps
$p1$ / $p2$	0.5%	6	90.7%	5.3kbps
$p1$ / $p3$	12.4%	3	77.0%	2.9kbps

Table 4.6: Comparison using several different devices as sender and receiver. BER is the average of five trials. Power: 12/16. Distance: 15cm.

4.5.4 Energy Consumption

To evaluate the energy consumption of our system we implemented WiFi, Bluetooth (BT), NFC, and our ultrasonic audio modem using a sender power of $\frac{14}{16}$. Using $p2$ as the sender and $p4$ as the receiver, we utilized each interface and measured the total consumed energy, using a Monsoon power meter [40]. In Fig. 4.15 we present the energy consumed on average per bit. It is important to note that the energy used by our system, roughly 32uAh / 1s, is actually small, causing roughly 56% increase over the idle rate of 57uAh / 1s.

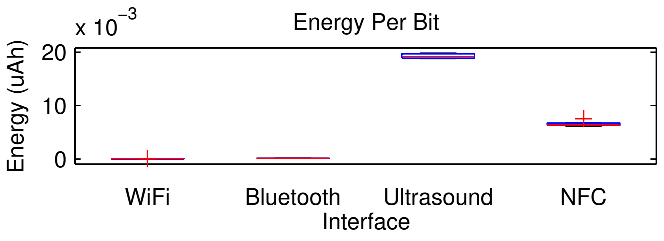


Figure 4.15: Consumed Energy per bit utilizing different physical media interfaces.

4.5.5 Audibility User Study

We used $p1$ and measured about -19dB of sound over the background noise of -50dB according to a dB measurement application [101] in a quiet office room. However, this measurement is not very insightful, because it includes ultrasonic frequencies, which humans can't hear.

To evaluate how much sound from our modem is perceived by the user, we asked eight participants (mean age: 27) to listen to our modem. The modem sending volume was set at 60%. The user is asked to listen to the modem transmitting random data five times, and rate how much noise they heard on a scale of 1 to 100 each time. The experiment was implemented as an Android application using a scrollbar, with end labels "silent" at 1% and "fire alarm" at 100%. The results are plotted as a histogram in Fig 4.16. The highest rating (21%) indicates that our modem is very difficult to hear. The modem is best described as a emitting a ``soft buzzing" sound. Some participants made comments that they felt they couldn't hear anything at all, and thought perhaps the modem was malfunctioning.

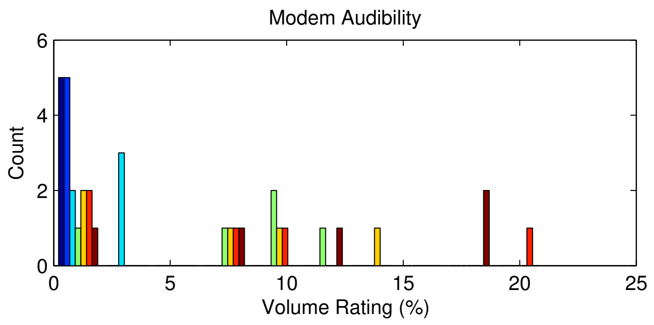


Figure 4.16: Histogram of user audibility rating. Results are color coded by user.

It is possible that our modem is more audible to young children, or animals, or that ultrasound might have adverse or unforeseen effects on users in general. Recent studies have shown that younger people are more sensitive to ultrasound [21, 49]. However, we leave this for future work, as these questions are largely out of the scope of this work. Young children and animals are not the target users, and the breadth and magnitude of investigating these issues rigorously is extremely large.

4.6 Conclusion

In this chapter we propose the use of ultrasonic audio as a physical medium for proximity networking. We overcome several challenges including poor frequency response, and limited sample rate. Our evaluation shows that our system can achieve an effective transmission rate of 6.1kbps with a PSR of 100% in the best case and 4.6kbps with 84% PSR in the average case.

Chapter 5

Physical Media Covert Channels

To provide better security and privacy at the operating system level, we present a novel attack vector based on a technique we call physical media covert channels. These channels can be used to leak sensitive information from a users device. They are plentiful on smart devices, because of the wide array of sensors they carry. Due to the fact that they are not hindered by even state of the art, taint-analysis based defense schemes proposed in recent research, it is necessary for the research community to create new defenses against them. It is our hope that by exposing these new attack vectors, and by providing a first step defense mechanism built into the operating system, we can spur innovation in more robust and effective defense mechanisms before these attacks are widely deployed by attackers.

5.1 Background

Since the introduction of the iPhone in 2007 mobile computing using smartphones and tablets has exploded. There is currently a plethora of Windows, Android, and iOS devices available to consumers containing a wide assortment of physical sensors. Consumers have come to expect smart mobile devices as common place. They are now carried everywhere and relied on daily. This is proving to be a fundamental shift for computing in two ways. First, these devices carry an unprecedented amount of sensitive information such as contact information, passwords, GPS location traces, financial information, and personal attributes such as sexual orientation and health data. Secondly, smart mobile devices are commonly equipped with a variety of physical interface hardware such as accelerometers, cameras, vibration motors, speakers, and microphones. Users of smart devices are expected to be conscious of a myriad of security *best practices* to protect this data, such as using unique and strong passwords for their various services, enforcing reasonable permission allocations to applications, and protecting highly sensitive information with encryption (or keeping this information off the device). These sort of defense schemes are largely lost on users, and can be difficult to implement for developers [20, 76]. In this work we preemptively propose a new form of malware, tailored specifically to mobile computing, along with a first step defense mechanism in hopes to alert the community of this potential threat and spur future research.

This work is interesting because we can exploit several properties of mobile computing. Firstly, the abundance of sensitive information found on these devices is highly valuable to attackers. Passwords are an obvious example, but attackers may also try to learn users' addresses and financial information, which can be used to steal their identity. Users' political views or sexual orientations, can be used by oppressive governments or organizations. Their personal traits and interests can be used to answer common second factor ``security questions'' such as their childhood home, or pet's name. Secondly, we exploit the variety of physical world sensors to establish several unique covert channels, which we refer to as ``physical media covert channels'' (PMCC). Because of the architecture currently in place in mobile operating systems, we show that we can easily design malware, using PMCCs, which appears benign fooling both non-expert humans and software systems seeking to eliminate malicious software; this makes it a variant of trojan horse malware.

Designing and implementing the PMCCs we use in our malware is difficult, because we must achieve two goals at the same time. First, speed. Previous work has shown that as little as 100 bits per second is enough to pose a serious threat, we show that at least one of our covert channels can achieve a few thousand bits per second. Secondly, stealth. The stealth of the covert channels is very important so they are difficult to defend against. They must go unnoticed by the user, and appear to be benign from the point of view of the software. Additionally, defending against these channels is challenging, because we must differentiate between benign and malicious sensor use without interrupting key features or annoying the user with confirmation dialogs.

Our attacks overcome not only the current, widely deployed defense mechanisms, but also the defense schemes posed in recent literature. Most recent work in this area can be divided into three areas: taint-analysis, elaborate security policy mechanisms, and application market curation. Taint-analysis can be used to identify sensitive information as it flows through an application and notify the user or stop this sensitive information from leaving the device and being leaked. Sensitive information sources are marked as such (e.g., the user's contact list) and sinks are identified that will ultimately leak this information (e.g., the Socket API). Security policies can be written by users, or automatically, that disallow certain types of malicious behavior. Market curation techniques aim to identify and remove malicious applications from the market before users even have a chance to install it. Our attacks overcome all of these approaches because of our physical media covert channels, which are novel and difficult to differentiate from benign behavior accurately. Our defense scheme provides a framework, that can be extended in the future, to identify and defend against variants of the malware we propose.

In this work we identify a new malware specifically designed for mobile devices. PM-CCs used in the malware are constructed using the sensors found on smart devices. We show that our malware can easily leak sensitive user information, such as bank statements, or location traces despite any current defense mechanism. Building such a system introduces several challenges including achieving a high bit rate, and low detectability (stealth). We go on to propose a defense scheme against these attacks, which can be implemented by the operating system provider, that aims to enable taint tracking over these channels. To summarize, in this project we make the following contributions:

- We propose a new class of covert channels for smart mobile devices that utilize real world interfaces (e.g., the vibration motor and the accelerometer). We generalize these as ``physical media covert channels'' and demonstrate that they can achieve varying levels of stealth, and speed.
- We design, and implement five example PMCCs, which utilize ultra-sound, physical vibration, light, and the user themselves. We spend extra effort on our ultrasound channel to show that with some engineering effort we can achieve a very high transmission rate.
- We use our PMCCs to design a new variant of Trojan Horse malware, which appears to be benign but actually leaks sensitive user information. We give a specific example called ``Jog-Log.''
- We propose and implement a novel defense scheme that takes a framework approach. The defense aims to propagate taint information across these channels, while still maintaining high usability of benign applications.
- We evaluate prototypes of each covert channel and the defense mechanism. We show that our fastest channel, the ultrasound covert channel, achieves 3.71kbps, and our gyroscope channel is very stealthy. Our defense scheme maintains high usability for the user, while stopping all of the proposed attacks with low overhead.

5.2 Threat Model

We assume the attacker is remote. He or she does not have any physical access to the user's device and cannot alter the operating system, or any other software, already running on the user's device. The attacker's goal is to convince the user to run their malicious code, which is used to obtain some information about the user. By ``attain'' we mean that the attacker's code accesses some sensitive information, and transmits it, over the Internet, to an attacker controlled host machine.

In order to send information out over the Internet in the malicious application, without requesting the Internet permission, the attacker can exploit a simple capability leak, which is a form of a *confused deputy attack* [28] using the web browser. Android applications can ask the browser to open URLs on their behalf without declaring the Internet permission. The attacker can include some CGI parameters (e.g., attackerhost.com/collector?usersecret=val) to transmit sensitive information to their own controlled host. Although there are protections against these attacks [9, 57], they are not widely deployed. Furthermore, the attacker can always just declare the Internet permission and communicate with their host directly, although this is less stealthy to the user.

Creating Applications The attacker is able to implement Android applications which are, from the point of view of the victim, normal applications. The attacker does not have to provide the source code to these apps, distributing only .apk files, which are essential .zip archives of java byte code and application assets such as image files and text files containing localized strings.

In order to convince the user to install these applications, the attacker can employ a few well studied techniques. Specifically, to circumvent application market curation techniques [124, 126], a malicious application developer can employ obfuscation such as utilizing native code, java reflection, remote code downloading, identifying malware-detection environments [81], or other more sophisticated techniques [84, 16, 111]. This allows the application to appear in application markets, drastically increasing the number of people that will run it.

To be more specific, the attacker aims to build a trojan, which by definition, appears to be benign from the point of view of the user, but actually performs some malicious activity. The attacker can even implement complete benign functionality to produce a convincing trojan.

Timing The attacker can employ standard Android timing mechanisms such as the

``ScheduledExecutorService'', in order to execute certain portions of their code at certain times of day (to increase stealth), and to ensure that two different components (possibly in two different applications) run at the same time to facilitate covert channel transmission.

Accessing Sensitive Information As mentioned previously, there are many pieces of information that may be considered sensitive on a typical smart mobile device. This information can be organized into three forms based on how it is accessed.

First, applications maintain sensitive information and may expose it for use by other applications. For example, the contact manager application maintains contact information for other people. Typically individual contacts can be ``shared'' via an interface between the contact app and another app (e.g. the SMS app may be able to access an individual contact to send the phone number of that contact to a friend). Typically this type of information is guarded with some permissions. For example, to read the phone records of the user, the application must declare the ``READ_CALL_LOG'' permission, which is displayed to the user at install time. For first party applications such as the phone call app, a permission is always necessary, for third party applications it is less common.

Secondly, certain hardware devices can provide sensitive information immediately, such as the GPS radio. These devices are guarded by permissions, but other sensors, which are not traditionally thought of as sensitive, do not require permissions to access, such as the accelerometer and gyroscope.

Third, sensitive information can be extrapolated from these sources. For example, by polling the GPS sensor regularly the attacker can likely learn the user's home address and work address by examining where they are late at night and during the day on work-days. These sort of inference or extrapolation attacks are plentiful in recent research [56, 108].

Attacker Limitations The attacker cannot access the hardware devices, such as the accelerometer and microphone, except by going through the provided Android APIs. This is enforced already in commodity Android due to user permissions, and SELinux. This means that the attacker must abide by the permission system that governs these APIs [39]. We also assume the attacker does not have root level access on the user's device.

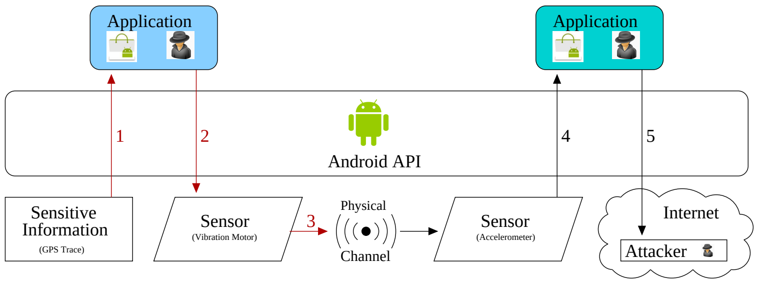


Figure 5.1: Information flow using physical covert channels.

We do not focus on the case where the attacker tries to use some physical media covert channel to communicate with another proximal device. Although it may be possible to form some sort of ad-hoc network out of nearby attacker controlled devices, we consider this scenario to be largely impractical and unlikely.

5.3 Trojan Horse Malware Design

In this section we present a general trojan horse malware, specifically designed for commodity smart mobile devices that utilizes PMCCs. Our malware aims to appear benign to users and software systems, while actually leaking sensitive user information to an attacker controlled host over the Internet. A high level view of the malware can be seen in Fig. 5.1. The attack works in four phases. First the malware is installed by the user on their personal device. Then it accesses some sensitive information, such as the user's GPS location trace. Third, the malicious application component encodes the information and sends it out over a physical device. In Fig. 5.1 the vibration motor is used as an example. At the same time, a second attacker controlled application component is used in the background to gather samples from the corresponding sensor. This second application component decodes the signal from the sensor, obtaining the original data (untainted). Finally, the receiver application component sends this information to some attacker controlled host on the Internet.

By ``laundering'' the information over a PMCC, we circumvent the current, widely adopted permission systems on mobile devices, which try to isolate the data of each application in respective sandboxes. Our channels allow applications to share information without the OS or even current state-of-the-art defense schemes, (including those based on taint-tracking [9, 22, 28, 45]), being alerted. Because none of the systems in recent literature (to the best of our knowledge) account for transmission over physical media, our covert channels pose a novel threat.

5.3.1 Example Trojan Application --- Jog-Log

An example trojan application might be a fitness app, which helps users track their running progress. The attacker's goal is to attain the user's home address. The application implements a simple ``jogging journal'' which determines when, for how long, and where a user jogs to help them track their progress. The application is closed source and is submitted to a well known application market.

The user, interested in jogging, finds the app in the market and installs it. Because they are weary of their information being leaked, they run the taint-droid system on their device [22]. When they want to begin a run, he or she starts the application and the GPS radio is used by the app to track the run. The application declares the permission to access the GPS radio, but it does not request the Internet permission. The application also requests access to the microphone to allow the user to add simple ``voice notes'' to their journal entires (e.g., ``I was greeted with a beautiful sunrise this morning thanks to my new jogging hobby!'').

Later, at night when the user is sleeping, the application uses the ScheduleExecutorService to wake up and use a PMCC to transmit the location information gathered earlier during the most recent run. Specifically, the speaker is used to produce an ultrasonic signal. At the same time, the microphone is used to decode the signal in a second component (part of the same application, running simultaneously in a second thread). Because the information has been laundered over a PMCC, the original taint-tags associated with location have been lost. Then, the second component forms a URL with the attacker's host as the domain, and the user's location as a CGI parameter and asks the browser to open this page.

The attacker sets up a special web server to respond to these requests by recording the CGI parameters in a file associated with the IP address of the user. Once the data is at the attacker's host, the attacker can find the street address nearest the GPS coordinates logged at the start and end of the user's runs, which is likely their home address.

5.3.2 Attack Variations

An attacker must control two application components, (e.g. a foreground activity and a background service), one sender and one receiver, to implement a covert channel. These components can be part of the same application, to circumvent taint-tracking analysis, or two different applications, to circumvent taint-tracking *and* the current, widely deployed permission system and information sand-boxing. If the attacker chooses to use two independent applications, they must convince the user to install both of them. To achieve this, the attacker can craft these applications so they are related, with some combined functionality and do in-app cross promotion. This is common practice with many developers in the Android ecosystem. The ``Go Launcher'' [107] and ``Yahoo Weather'' [123] apps serve as just two prominent examples.

In our Jog-Log example, the attacker may instead choose to place the voice note feature in a second, stand alone application and do in-app cross promotion. If the user installs both apps, then the attacker can use the covert channel to move the location information from Jog-Log into the voice recording app using a PMCC, and then use the voice recording app to transmit the data to their host. In this case, the voice recording app can declare the Internet permission, and still the user will not expect that there is any way that their location information (in the Jog-Log app with no Internet permission) could possibly be sent out over the Internet.

If the developer chooses to implement two components in the same application, then the malware can defeat current state-of-the-art defense techniques, but will be less stealthy to users, because it must present all of the corresponding permissions together at install time. Fortunately, it has been shown that the permission system is commonly abused by users and developers [76], so this is not a significant challenge for the attacker.

If the attacker is a large enterprise that distributes smart mobile devices, they can include the malware via pre-installed applications. This approach is much simpler because the attacker does not need to be concerned with market curation, or convincing users the permissions required do not pose a threat. Pre-installed applications are also significantly more difficult for users to remove, due to both technical and potential legal challenges. However, few attackers have control of an enterprise or a similar level of influence over which apps a user has installed.

5.4 Covert Channel Design

The foundation of our newly proposed malware, is our novel physical media covert channels. In this section we discuss our implementation of five such channels, to increase the attack surface, making the defense more difficult, easing the task of creating seemingly benign malware, and to show that the different channels have different strengths.

Each channel utilizes different permissions. We summarize what Android permissions are used in Table 5.1. For the remainder of this chapter, we will use the row numbers of this table to refer to specific permissions. It is important to note that permissions (4) and (5) are actually members of Android's ``uses-feature'' tag, not the ``uses-permissions'' tag, which are *optionally* used by the developer.

All of our channels are stealthy to software, because until now, none of them are considered to be able to transmit and receive information. They have varying stealth in regards to the user, which we detail in each channel subsection.

Ultrasound We put extra effort into the development of our ultrasound covert channel

	Permission	Hardware	Description
1	MODIFY_AUDIO_SETTINGS	Speaker	Change volume
2	CAPTURE_AUDIO_OUTPUT		Direct access to speaker buffer (e.g., record phone call)
3	RECORD_AUDIO	Microphone	Use microphone
4	android.hardware.sensor.accelerometer	Accelerometer	Use accelerometer
5	android.hardware.sensor.gyroscope	Gyroscope	Use gyroscope
6	VIBRATE	Vibration Motor	Use vibration motor
7	WAKE_LOCK	Screen	Prevent device from locking automatically
8	CAMERA	Camera	Take pictures
9	FLASHLIGHT	Camera Flash	Use lamp
10	READ_PHONE_STATE	Phone	Learn if phone is locked or a call is active
11	INTERNET		Open network sockets

Table 5.1: Summary of various permissions used by our covert channels

to show that these physical media covert channels have potential for relatively high bit-rates. This channel uses the speaker and microphone found on smart devices. The main idea is we send very-high frequency, modulated sound waves (above 18kHz) in packets from the device speaker to the microphone. Permissions (1) and (3) are used.

A high level view of the ultrasound covert channel can be seen in Fig. 5.2. Here the attacker can control one malicious application (housing both the modulator and demodulator), or two applications, where the demodulator is separated out. The malicious

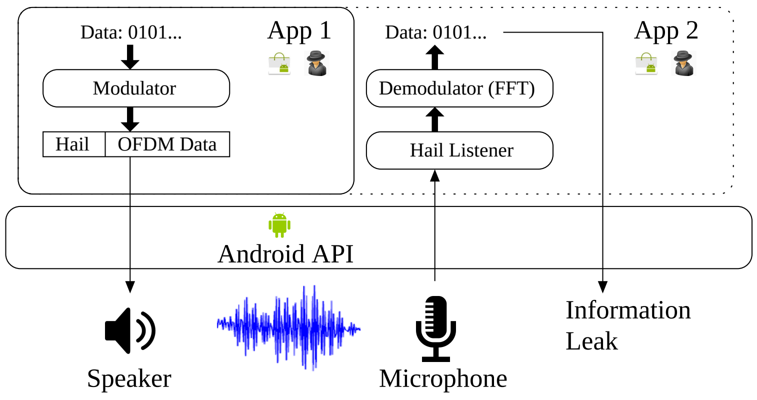


Figure 5.2: High level module view of the ultrasound covert channel.

sender generates packets in the ``Modulator'' module and sends sound data over the device's speaker. The receiver uses the microphone to record this sound. The signal is first parsed by the hail listener, which finds the starting point using the cross correlation between the received signal and a pre-established ``hailing'' signal. Then, using the Fourier Transform (FFT) module, the malicious receiver can recover the data.

We choose to use 18khz - 22kHz, because this spectrum is inaudible to humans (stealthy), below the Nyquist frequency for most smart devices, and has relatively low background noise. In order to remain inaudible, but still achieve a high bit-rate, we choose to work in the frequency domain. We generate data segments 882 samples long, each of which is the sum of many modulated sub-carrier frequencies.

$$s_i = \sum_{f=18kHz}^{22kHz} sin(2 * \pi * i * \frac{f}{F_s} + \theta) : \forall i \in [1 - 882] \tag{5.1}$$

Each sub-carrier (of which there are 70, spaced 50Hz apart) encodes a binary phase value and a binary amplitude (19% or 100%). We reserve four sub-carriers for calibration (18k, 19k, 20k, 21k), which are used to reduce error when recovering bits. This affords a maximum theoretical throughput of roughly 6.5kbps, which is high compared to traditional covert channels [27]. The receiver can use the FFT, to determine the amplitude and phase of each sub-carrier frequency present in a given packet $S = \{s_1, s_2, ..., s_i\}$. This is commonly referred to as orthogonal frequency division multiplexing (OFDM). When designing our system we used the following parameters. Packets of length $i = 882$ samples (20ms), $\Gamma = 50$Hz sub-carrier width, and sample rate $F_s = 44.1$kHz. We use 3.5kHz of spectrum from 18kHz to 21.5kHz.

Speaker and Accelerometer The speaker on most smart devices can cause the entire device to vibrate, if the tones are played with a loud enough volume. We use the speaker as a sender, playing standard Dual-Tone Multi-Frequency (DTMF) tones, and the accelerometer to measure the vibration of the phone, which will resonate with the tones being played. We then perform binary amplitude shift keying to modulate the data. DTMF codes are used because smart phones are designed to produce them well, and they are less conspicuous in this context. Permissions (1) and (4) are used. Although this channel is slower, and less stealthy than our ultrasound channel, we present it to demonstrate that seemingly arbitrary sensors can be combined to form a channel.

Vibration and Accelerometer Our vibration channel uses the vibration motor (6), normally used for silent notifications, as the sender, and the accelerometer (4) as the receiver. Again, binary amplitude shift keying is used for modulation. The stealth of this channel can be increased, by transmitting immediately after benign vibration events. In this way, the user may mistake the transmission for a long notification or may not notice at all.

Flash and Camera In this channel the camera's flash (9) is used to send data and the camera (8) is used to receive it. We simply turn the lamp on and off and run some simple image processing on the captured image preview from the camera to transmit data. The average brightness of the image should be much higher if the lamp is on, compared to when it is off. This channel has several draw-backs. Firstly, the camera can only be used by an application if a preview is shown in the foreground. Secondly, the camera light is very obvious and suspicious to even non-savvy users. Thirdly, this channel relies on some assumptions about physical orientation and environment. However, we show in our evaluation that the channel works even when the device is placed camera-down on a flat surface, which increases the practicality and stealthiness toward users.

User and Gyroscope The main idea of this covert channel is much different from the others. Here, we tilt the phone, and measure this action using the gyroscope (5). Bits are encoded in the angle of the device over time. To transmit the bits, we fool the user into tilting the device in the correct sequence by implementing a simple ``endless running'' game. In the ``endless running'' genre, an avatar moves down a track collecting items and avoiding obstacles. The user is tasked with tilting the device (or swiping) left and right to move the character on and off several different tracks. Endless running games typically feature randomly generated tracks, with the challenge being how long the user can avoid the obstacles. We simply generate the track according to the bit stream to be transmitted. This channel is highly stealthy in that the user is interacting directly with the device, and will have no idea the channel is being used. In fact, the user is part of the channel!

We make a novel generalization of this concept we call ``user-sensor'' covert channels, which fool the user into taking specific actions (touching areas of the screen, pressing hardware buttons, etc.), which act as data transmission symbols. User-sensor channels present many unique challenges such as handling inevitable user errors, and clever game design, which are out of the scope of this work.

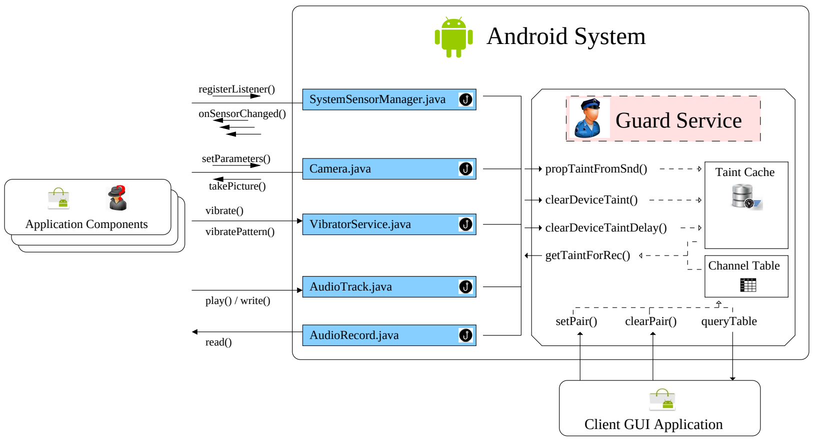
5.5 Defense

The physical media covert channels (PMCC) we've outlined thus far exhibit the precarious nature of sensitive user data on modern smart mobile devices. Our goal in presenting them is to spur research in defending against these attacks, which pose an obvious threat. To further our contribution in this work (above and beyond our previous publication of PMCCs [73]) we present a defense framework, which protects against information leakage attacks done using PMCCs. A prudent defense must take into account many critical design decisions such as maintaining high usability by avoiding nagging user prompts, complex user-configured privacy policies, and coarse-grained or blanket rules. Applying previous defense approaches, like TaintDroid[22] in a naive way will lead to poor usability and therefore, poor user privacy. This is especially difficult for information leakage using PMCCs, because it is very difficult to differentiate, using static or dynamic analysis, benign from malicious sensor use. Sensors are commonly used in a wide variety of benign applications with similar device combinations, and frequency of use. For example, any teleconferencing application will make use of the speaker and microphone simultaneously, for a sustained period of time, in a completely benign way that is very similar to an implementation of the speaker + microphone covert channel.

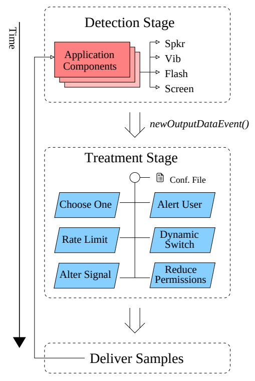
In our previous work, we laid out a ``first-step'' defense scheme that had extensibility limitations, and was substantially different from the defense scheme we present in this dissertation. Firstly, our original defense scheme used data structures that were different. The system recorded which Android component(s) (service, activity, etc), accessed hardware devices (microphone, speaker, etc.). This required a special ``component'' data structure, which stored the fully qualified *package:component* name, along with what device was being access, and taint-tag information. The previous defense maintained a dynamic list of these components as different hardware devices were used. As far as limitations are concerned, our previous approach required a lot of effort to define new covert channels between existing hardware devices, and to incorporate hardware besides those implemented directly in our work (i.e., accelerometer, gyroscope, camera, flash, vibration motor, speaker, microphone, and custom canvas elements / the screen). Therefore, in designing our new defense system, we summarize our three goals as follows:

- **Coverage** We demonstrate several covert channels in this work to demonstrate that seemingly arbitrary devices can be used to form a covert channel. Our defense should stop each of these channels, as well as provide extensibility to detect and treat future channels.
- **User Experience** Our system should make it easy for the user to define the covert channels that they are concerned with.
- **Overhead** The system should be low performance overhead and be minimally disruptive so that it does not take away from the user experience.

Our proposed defense system architecture, illustrated in Fig. 5.3, works in two stages (Fig. 5.3 (b)). The *detection stage* aims to maintain an always up to date record, during run time, of which potential covert channel devices are being used, and the data flowing in/out of them. When sensor data is read (e.g., the microphone), if there is an active corresponding sender device (e.g., the speaker), we propagate the taint-tag from the data on the sender side (speaker), to the data on the receiver side (mic), thereby extending the existing taint tracking solutions. Then, in the *treatment stage*, we provide some additional protections according to a user configuration file such as alerting the user, or limiting the use of the sensors in some way.



(a) Defense System Architecture. Various system API classes are altered to hook into the ``GuardService'' Android system service. The GuardService detects potential channels and applies the configured treatment.



(b) Defense System Framework.

Figure 5.3: Architecture and Framework for our defense system. Applications that use sender devices are recorded. When there is a new data event for some hardware, the taint-tag is propagated by our service, before the samples are delivered to the appropriate application component(s).

5.5.1 Taint-Tag Propagation

Fortunately, any application that is trying to transmit data secretly, using a PMCC, must use an API provided by the operating system to access the hardware device(s). In order to detect the API use, we add a system service called the ``GuardService'' to the operating system, and we modify the internal Android provided API of each device, to hook into the GuardService. As illustrated in Fig. 5.3 (a), each physical device has a corresponding .java file in the Android source code, which is modified to call the GuardService provided functions *propTaintFromSnd()*, *clearDeviceTaint()*, *clearDeviceTaintDelay()*, and *getTaintForRec()*. These methods allow the GuardService to monitor device usage by any and all applications throughout the system, and to read/write the taint-tag of data being sent to / received from any device. We also provide a mechanism for the user to specify which covert channels they're concerned with monitoring. This information is stored in the channel table, which is explained in greater detail in Section 5.5.2.

Sender Devices When any application sends data to a ``sender'' device (e.g., the speaker or vibration motor), by calling the appropriate API function(s), there will be some data passed in that describes what the device should do. When this occurs, we use *propTaintFromSnd()*, to read the taint-tag values from that input data, and store it in the ``taint-cache'' for the corresponding device. Because we extend the existing Taint-Droid implementation, this taint information is a 32-bit bitmask and the taint cache is an array of integers, one for each of the sender devices.

Receiver Devices When a receiver device (e.g., the microphone or accelerometer) is accessed, *getTaintForRec()* is called, which causes the GuardService to look up the taint-tag for the corresponding sender(s), in the taint-cache, and propagate the taint-tag from the cache, onto the receiver data, before it is delivered to the application. The specifics of the API, and the semantics of the data, are unique to each device and vary in complexity. We describe in detail our implementation for the commonplace physical world sensors and actuators on Android devices in Section 5.5.1.2. This approach allows us to monitor the use of devices, and propogate taint-tag information from the sender side of the covert channel to the reciever side.

5.5.1.1 API Implementation Details

In the following we outline the semantics of our four API function calls. Each of which takes an *int dev* input parameter, which specifies a device. Our service provides integer constants for the eight, most common devices (TYPE_VIB = 0, TYPE_ACCEL = 1, ...).

propTaintFromSnd(dev, taint) is called when a sender device begins transmitting, or outputting. The *dev* specifies which device is transmitting, and *taint* specifies the taint-tag to be cached. The taint-tag is stored in the taint-cache, indexed by the device being used as the sender (e.g., the top or 0-index spot for the vibration motor). An example taint-cache, with random values, in shown on the left side of Table 5.2.

getTaintForRec(dev) is called when a receiver device is providing data (usually done in batches), to an application. The function uses the channel table, which stores a mapping of which covert channels should be monitored. This table is user configurable (see Section 5.5.2), and defaults to include five of the covert channels outlined previously in Section 5.4. As an example, if an application begins reading the accelerometer data, the GuardService will taint that value with the taint-tags stored in the taint-cache for the vibration motor and the speaker, because those pairs are known to form PMCCs, and are therefore monitored by default. In short, *getTaintForRec()* triggers a very small one-column search in the channel table, and looks up the taint-tag from the cache for the corresponding sender devices. In this function, *dev* specifies the **reciever** device.

clearDeviceTaint(dev) is called to reset the taint-cache for *dev* to 0 when a sender device stops transmitting. For example, the vibration motor provides a *cancel()* function, which immediately stops any vibration. When this function is called, we should also call *clearDeviceTaint()* to reset the taint-tag value for the vibration motor to zero, as it is no longer able to communicate any data. We describe how this function is used in more detail for each device in section 5.5.1.2.

clearDeviceTaintDelay(dev, delay) is used to call *clearDeviceTaint()* after the specified *delay* in milliseconds. Some devices are able to transmit data with little to no correlation to the timing of API function calls, such as the speaker. The speaker API (AudioTrack) allows the user to write data to the speaker buffer, and then begin playback by calling *play()*. The *play()* function returns immediately, and the speaker produces the signal previously written. The time it will take to produce the signal can be determined, based on how much data was written to the buffer, and some parameters such as the sample rate of the buffered data. After estimating the playback duration, *clearDeviceTaintDelay()* can be used to clear the taint-tag after this time. This particular function implements a thread, which waits for the given duration (*delay*) using the built in Thread.sleep() API. After sleeping, the thread then calls *clearDeviceTaint()*. Because Thread.sleep() only guarantees that the calling thread will sleep for at least the given amount of time (lower bound), the same is true of *clearDeviceTaintDelay()*. However, this error is tolerable, because clearing the device taint-tag too late results in a false positive, which is overly cautious instead of a false negative, which covert channels could game to transmit data without taint-tag propagation.

When the malicious code uses a covert channel, the previous work breaks down. By propagating the taint-tag information from the sender side data to the reciever data, using the GuardService provided API, our system allows the taint information to traverse these covert channels, as depicted in Fig. 5.3. And, by using taint-tracking analysis, we can ignore benign instances when physical devices are being used, but there is no sensitive information flow because the taint-tag of this data should be: 0.

5.5.1.2 Device Hook Implementation Details

We implement a prototype of our defense scheme on Android by downloading and modifying the Android Open Source Project (AOSP) source code. For taint tracking, we extend the existing Taint-Droid system [22].

By default, none of the Android device API classes will notify our GuardService when devices are in use, which is necessary in order to propagate taint information over the covert channel. We modify the API for each device to hook into our ``GuardService'' accordingly using the API methods outlined in the previous section. While we strive to make our GuardeService API simple and easy to use, a trivial approach cannot be taken to implement the hooks. Because the devices vary greatly, the source code for these classes is complex, and there are many potential pitfalls. In the following we analyze the API of each of the sender devices and detail how we implement the hook.

Vibration Motor for the vibration motor, Android provides a function *vibrate(long millies)*, and a second function *vibrate(long[] pattern, int repeat)*, which both return immediately. In the former, *long millies* describes how long the vibration motor should vibrate, which begins immediately (ignoring negligible overhead) when the function is called by the application. In the latter, the application provides an array of longs, that ``are the durations for which to turn on or off the vibrator in milliseconds" [39] and an int *repeat*, which specifies the index at which to repeat the pattern from (-1 to disable repeating). When *vibrate(long millies)* is called, we read the taint-tag data from millies and store it in the taint-cache by calling *propTaintFromSnd()*. At the same time, we call *clearDeviceTaintDelay()* to set a timer, which expires after *millies* milliseconds and then clears the taint-tag value. This way, when the vibration motor has stopped vibrating, we no longer propogate any taint-tag values. Similarly, when *vibrate(long[] millies, int repeat)* is called, we concatenate the taint-tag values from long[] millies and int repeat and pass the resulting value into *propTaintFromSnd()* to be stored in the taint-cache. When *cancel()* is called, the taint-cache is cleared, so we do not have to set any timer in this case. Clearly, even the simple vibration motor has a relatively complex API.

Flash For the camera and flash covert channel, we describe how to transmit data using the flash, but there are actually several parameters that can be used to encode data such as the image resolution, which are provided in a *Camera.Parameters* class. When the camera is opened, the *propTaintFromSnd()* method is called, and we propagate the taint-tag of the entire class instance to the taint-cache. *getTaintForRec()* is called when a picture is taken, or preview frames are delivered, and *clearDeviceTaint()* is called when the camera is closed.

Speaker For covert channels using the speaker, the developer can implement the speaker API in one of two modes; a streaming mode and a static mode. In the static mode, described previously, the developer first instantiates an AudioTrack class instance, and writes any sound data using the *.write()* method, which writes an array to an internal buffer. Later, the developer calls a *.play()* method, which instructs the hardware to actually play the signal previously written. Alternatively, the developer can choose to implement streaming mode, in which the developer calls the *.play()* method first, and then, after some time, the *.write()* method, which will instantly generate the sound.

Our system places *propTaintFromSnd()* and *clearDeviceTaintDelay()* appropriately depending on the mode; i.e. with *.play()* in static mode and with *.write()* in streaming mode. We can estimate the duration that the speaker will be making noise based on the sample rate and the size of the array(s) passed to *.write()*. This duration can then be passed to *clearDeviceTaintDelay()*. This microphone implementation is straightforward and relatively similar to the speaker implementation, therefore, we have left it out of this work for brevity.

User and Game For this channel, the sender appears to be the user, but actually, the user responds to the information presented on the screen by the malicious game. Because the attacker has implemented a game for this channel, it is safe to assume that they will implement a canvas element to draw the game graphics. The Android canvas element exposes an *onDraw* method to the developer, which is called rapidly to update the current frame on the screen. We can propagate the taint-tags from the variables and data used in *onDraw* that are used in helper functions such as *drawRect()* and *drawArc()*.

As mentioned previously, to implement our defense scheme, we leverage an existing run-time based, taint-analysis system called TaintDroid [22]. The key difference between TaintDroid, and our work, is the inclusion of the physical media covert channels in the taint-propagation logic. While TaintDroid focuses primarily on internal or ``virtual'' information flow paths such as variable assignment, function calls, and inter-component communication, we focus on including external or ``physical media'' information flows. Namely, physical media covert channels.

5.5.1.3 Extending GuardService

In our implementation, we include all of the most common physical world devices commonly found on smart mobile devices (i.e. speakers, microphones, vibration motors, accelerometers, gyroscopes, and cameras). The variability in the API semantics from device to device poses a challenge for our defense scheme. While we fully support all commonplace devices, and allow the user to easily specify which channels should be monitored, it is difficult to add support for new or different devices when they are added to handsets, and it becomes apparent that they may be used as part of a covert channel in the future; such as temperature sensors or notification LEDs.

To address this limitation, we strive to design our defense scheme in such a way that it is easy to extend to add support for new devices like these. Our GuardService provides a simple API for hooking in new hardware devices that is focused on the simple design concept of input, and output. Although manufacturers, privacy advocates, or even technically savvy end-users will have to modify the Android system source code to hook into our GuardService, our simple API makes it easier. Therefore, when a device manufacturer produces a hand-set with a new hardware device (such as a temperature sensor), it is necessary to make only a few changes. A new constant (e.g., TYPE_TEMP = 5) should be defined, and the GuardService API methods should be called in the API code that controls when and how the device is accessed. Because it is already necessary to write the API code (for the device to be used at all), only a few lines need to be added to hook into the GuardService framework.

Specifically, for devices that output in some way, there should be an API call that allows applications to send (output) data to them. This API call should include a call to *GuardService.propTaintFromSnd()* passing in the taint-tag from the output data. *GuardService.clearDeviceTaint()* should be called when the device has finished outputting. Similarly, for devices that accept data (sensors), the API should call *GuardService.getTaintForRec()* to retrieve the taint-tag information and apply it to the output data. The GuardService maintains the taint-tag data from other devices, and coordinates user configurations about which channels should be monitored.

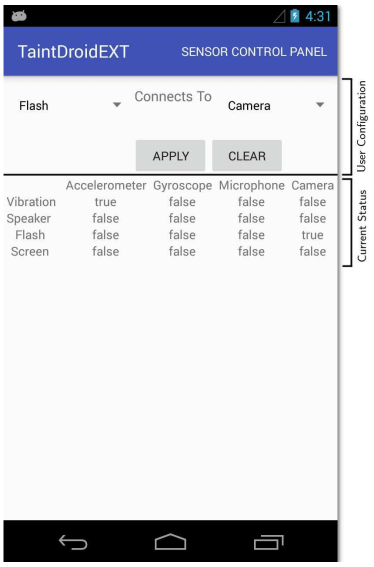
Furthermore, we design our defense so that GuardService hooks are inserted at the Android API layer (as opposed to a kernel module or the driver level). This means that all instances of a particular type of device (e.g. microphones from different manufacturers) will all be supported when a single, corresponding API file (e.g., AudoTrack.java) is modified just one time.

5.5.2 User Configuration

We provide an end-user facing graphical user interface, which allows the user to specify which PMCCs should be monitored. PMCCs can be used under certain circumstances only. Namely, the sender application must (1) have access to a source of sensitive information, and (2) have permission to use the hardware to transmit the information in a PMCC. The attacker must also (3) have control of some secondary component / application to receive the information, and (4) the receiving component must have permission to use the reciever hardware device. Furthermore, the system must make use of the PMCC without the user noticing. Therefore, users may feel confident that certain covert channels cannot be utilized, because the user would notice. They may act cautiously with the applications they install, or they may believe that it is simply impossible for some device pairs to communicate (e.g., the speaker and camera). Alternatively, new PMCCs may be discovered in the future, which the user may wish to monitor. For these reasons, we provide the user the ability to specify which covert channels they wish to monitor.

We arrange all known sender and receiver devices as the rows and columns of a table respectively. This table is known as the Channel Table, mentioned previously and shown in Fig. 5.3. An example channel table, with default values, is shown on the right side of Table 5.2. The table cell at the coordinates indicated by the device constants (e.g. position (2, 3) for the Falsh + Camera covert channel), stores a boolean value, which indicates to the system that the user is concerned with this covert channel, and that the taint-tag from the sender (Flash) data should be propagated to the reciever (Camera) data. The GuardService also exposes three API function calls to support GUI client applications as shown in Fig 5.3. The first, *setPair(int snd, int rec)*, allows the user to set the corresponding spot in the channel table as True. The second, *clearPair(int snd, int rec)*, sets the corresponding spot False. The third, *queryTable()* allows the client application to view the current status of the channel table.

In our implementation we provide a simple client Android application, which exposes a user facing GUI. This application allows the user to view the current state of the table, and it provides a simple interface, which is connected to *setPair()* and *clearPair()*. A screenshot of our GUI can be see in Fig. 5.4. Default values for the table are provided according to the channels we've described here (see Table 5.2). The application is able to call *setPair()* and *clearPair()*, which are part of GuardService, running in another process, because the GuardService exposes these methods for any client to call via the Android interface definition language (AIDL). The GuardService also exposes *queryTable()*, which allows applications to learn the contents of the table (which may have been changed by other client applications or processes). In our implementation, this method is polled once every two seconds as long as the application is in the foreground.



Flash Connects To Camera

	Accelerometer	Gyroscope	Microphone	Camera
Vibration	true	false	false	false
Speaker	false	false	false	false
Flash	false	false	false	true
Screen	false	false	false	false

Figure 5.4: Screenshot of our user facing PMCC configuration application. The user can turn monitoring on and off for specific channels using the controls on the top, and view the current monitor status in the table below.

Taint-Cache	**Channel Table**				
		Accelerometer	Gyroscope	Microphone	Camera
0x0	Vibration Motor	True	False	False	False
0x100	Speaker	True	False	True	False
0x0	Flash	False	False	False	True
0x800	Screen	False	True	False	False

Table 5.2: GuardService Channel Table, which is used to indicate which covert channels the user is concerned with. The values shown above are the default, which correspond to the channels outlined in our work. The taint-cache (shown on the left) stores the current taint-tag value from the data sent into the API call(s) for the corresponding Hardware.

5.5.3 Advanced Treatment Techniques

To improve our defense system, we also present several potential treatment methods in the second stage to further enhance the security of the framework. As illustrated in Fig. 5.3 (b), when a covert channel is detected, our system intervenes and reads a user controlled configuration file to determine what additional action to take, if any. Providing an array of different treatments options allows our framework to be very flexible and allows us to include the best ideas from recent literature, as well as leave the system open for improvement in the future, when new covert channels are discovered. Below we describe each of these options in greater detail.

Alert The User The weakest choice of treatment is for the system to alert the user of a possible information leak. This is the default action.

Choose One One device is turned off (the samples are dropped) for the duration the other device is in use. Which device is allowed and which is prohibited is a tunable parameter.

Dynamically Switch Allow only one device to operate transiently based on a simple threshold. For example, a video conferencing application may access the speaker and microphone concurrently. However, in typical conversation it is uncommon that both parties are actually speaking at the same time. Based on which signal is stronger (i.e. RMS for audio signals), we allow only one to flow, as illustrated in Fig. 5.5. By switching dynamically between the two sensors, we guarantee the covert channel cannot be utilized, because there is never a time when both sensors are active, but benign applications are still usable.

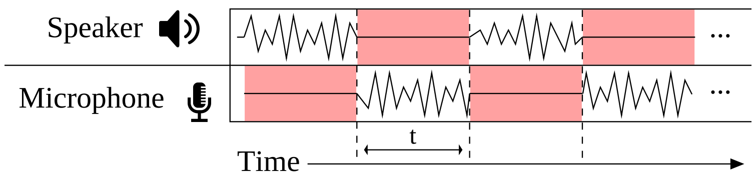


Figure 5.5: ``Dynamically Switch" treatment method. The speaker and the microphone cannot be accessed at the same time. The active device is chosen in each time window by calculating which root mean squared (RMS) is larger. The other device is silenced.

Rate Limit For the flash and camera channel, the two devices must be used at the same time in order to take pictures in low light conditions. Instead of prohibiting access to one device, one possible solution is to limit the rate at which that device can be used.

Altering the Signal For all of these channels, we can increase the error rate the attacker achieves by changing the signal. For example, filtering out ultrasonic audio, removing samples to create random pauses, or inserting noise in the signal. However, we must be careful not to effect benign behavior. Clever approaches may be possible, but unfortunately, must be tailored specifically to the encoding scheme of the attacker, as well as the signal processing of the benign application(s).

Reduce Permissions There are several ways we can defeat the channel by altering the permissions of the application components involved. For example, each component can be re-assigned the *intersection* of the permissions of both temporarily, which follows the principle of least privilege.

5.5.4 Limitations -- Implicit Flow Taint Tracking

As we mentioned previously, we leverage the existing Taint-Droid work for dynamic taint tracking analysis [22]. This system only tracks *explicit* flows, meaning direct assignments from one variable to another as shown in Fig. 5.6. However, in this case, it is very likely the attacker will use *implicit* flows to encode the sensitive information, as shown in Fig. 5.7.

To overcome this challenge, we propose a method inspired by previous work [64]. We identify these implicit flows using static analysis and include implicit flow taint propagation

```
s = api.getSensitiveInfo();
a = s;
a = s.getCharAt(3);
networkAPI.transmit(a);
```

Figure 5.6: Examples of explicit information flow.

```
signalSampleList = new List();
s = api.getSensitiveInfo();
for bit in s.toBinary(){
  if(bit == 0){
    signalSampleList.addSilence(500);
  }
  else{
    signalSampleList.addNoise(500);
  }
}
vibrationAPI.vibrate(signalSampleList);
```

Figure 5.7: Example of implicit information flow.

rules following a simple heuristic: If a branch depends on a tainted value, then we should propagate the taint-tag to the variables assigned inside the branch. Our modifications are done in three stages. First, at install time, the application is decompiled from the dex files to Java code. Then, we use static analysis to automatically find implicit flow blocks. We re-write the application code to propagate taint information into the variables in these blocks. Then, in the final stage, we recompile the application, re-generate the .apk file, and install it on the user's device. To reduce false positives, we only apply our implicit flow propagation if the output variable data depends on the entirety of the sensitive input data.

This approach is not perfect, malware has a long history of resisting such techniques. We leave more robust automatic application re-writing, implicit flow taint-tag propagation, and code block analysis to future work.

5.6 Evaluation

In our system we implement five covert channels, which utilize physical mediums (e.g., sound, vibration). We implement all five covert channels as single user-space applications. The two application scenario, with one sender and one receiver will not have any effect on transmission speed, errors, or stealth, so we did not implement it. The ultrasound channel was evaluated on an LG-C800 smartphone. The speaker / accelerometer, vibration / accelerometer, and light / camera channels were tested on a Samsung Nexus S and the game / gyroscope channel was tested on a Google Nexus 4. The dynamic switching defense evaluation was done on the Nexus S and LG-C800. The taint tracking experiments were done with a Galaxy Nexus.

5.6.1 Covert Channels

To evaluate the ultrasound covert channel, we implemented an android application that generates ultra-sound packets, as described previously. We transmitted 1000 bits in twelve iterations and plotted the percentage of bits recovered incorrectly in Fig. 5.8. We

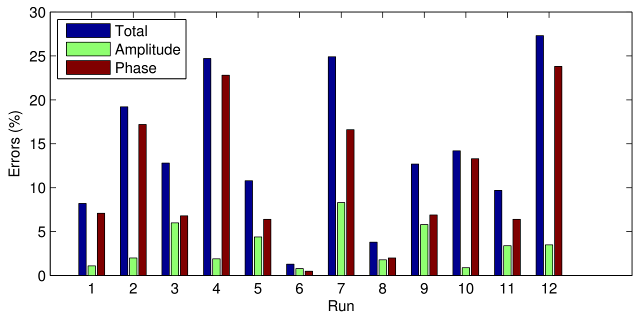


Figure 5.8: Ultrasound Bit Error Rate. Phase and Amplitude correspond to bit errors from demodulating these sub-carrier attributes respectively.

can see that the total error percentage and phase error percentage have high variance (6.98, and 6.01 respectively). Trial number six shows that, occasionally, the channel can achieve very low error rates. Amplitude error has lower variance (0.57) and shows a lower mean (3.3%). Using both phase and amplitude, we can achieve a bit rate of 6.5kbps with relatively high mean error (14%), and by utilizing amplitude only, we can achieve a lower rate of 3.25kbps that is more consistent.

We performed a short audibility user study to show that our ultrasound scheme is stealthy. A random string of 1000 bits was encoded and transmitted three times in the presence of ten individuals between the ages of twenty and thirty. The experiments were held in two different meetings, with a volume of $\frac{6}{16}$. Participants were not notified beforehand of the experiment and were questioned shortly afterwards about having heard anything. Unanimously, nobody was able to hear our system being used. Even after being told, and actively listening, untrained users have great difficulty hearing the ultrasound signals emitted.

Speaker & Accelerometer We implemented the speaker and accelerometer covert channel described previously and we used it to transmit 1024 random bits, ten times, at a rate of 2bps to measure the bit error rate. Because we don't have robust synchronization, our demodulator sometimes inserts extra (incorrect) bits or drops bits. We count the total number of bits in error as the sum of the number of inserted, dropped, or incorrectly decoded (flipped) bits. Each type is only counted as one error, even though missing a bit will propagate errors through the rest of the bit stream. For this channel, the recovery is very good; we had only four bit errors (0.039%). Six of the runs transmitted with no errors at all.

Vibration Motor & Accelerometer To evaluate this channel we transmitted 1024 random bits ten times and recorded the bit errors, by type, in Fig. 5.9. This channel has a low error rate and achieves 2bps throughput. Similar to the Speaker and Accelerometer channel, we can see that there are only a few bits which were incorrectly decoded, the rest of the errors are due to improper synchronization.

Camera & Flash We transmitted 1024 bits using the channel ten times and measured the bit error rate and transmission time. The experiments were done in a dark room (similar to when the user may be sleeping), which can be seen in Fig 5.10. Over all ten trials, there were no bit recovery errors and the transmission time was very consistent

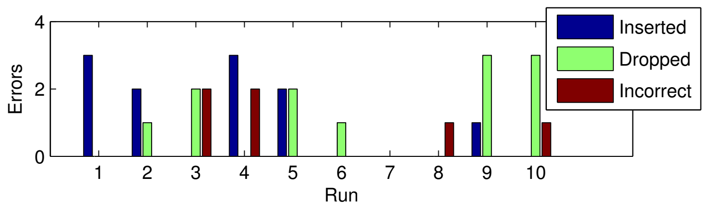
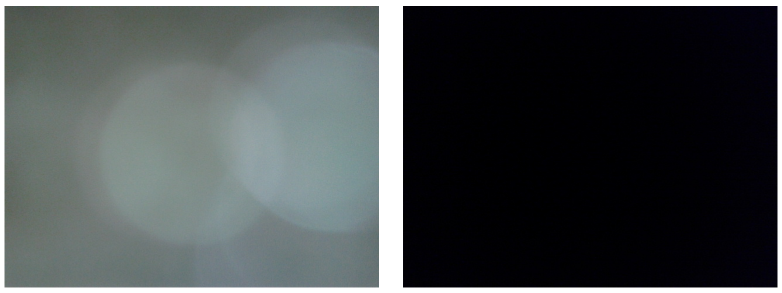


Figure 5.9: Vibration bit errors by type. Inserted: extra bit was decoded. Dropped: bit is missing. Incorrect: bit was decoded incorrectly (flipped).

at 664 seconds (± 2). The bit rate we achieve is 1.5bps. The limiting factor here is the camera preview, which takes about one second and must be started and stopped each time to change the flash's state. We also performed a simple test to confirm that the



(a) Camera Down, Flash (b) Camera Down, No Flash

Figure 5.10: Camera images captured with and without flash. Images gathered when the phone was camera down, on a desk.

camera can be in different orientations and environments. We placed the phone flat on the ground, in a poorly lit area under a desk, held upright in a dark room at night, and flat, camera down on a desk. It's important to note that even when the device is placed flat, face down on a smooth service, it's still possible to determine if the image taken had the flash on or off, as can be seen in Fig. 5.10. This is because the camera has a small piece of acrylic which covers both the flash and the camera lens. This piece of acrylic allows light to flight laterally from the flash lamp directly into the camera lens. We transmitted eight bits using this channel; each time and we were able to recover all eight bits with no errors in every scenario. We can attribute this success to the physical design of camera hardware on the phone (Samsung Nexus S). Because some light is able to travel from the flash to the camera lens, even when the camera is facing down on a table the stealthyness of the channel is greatly improved, as the user will not notice the flash when blocked by the desk.

User / Gyroscope We implemented a very primitive game prototype in which the user is instructed by a foreground application to rotate the phone in one of the three axes (6 different symbols). A background service measures the gyroscope at the same time to decode the bits. The user is tasked with finishing each rotation task in as short a time as possible. We played this ``game'' ten times and transmitted twenty symbols each time. On average, the user is able to rotate the phone in 1.02 seconds, making for a bit rate of roughly 2.5bps. There were no bit errors whatsoever.

Comparison In Table 5.3 we compare the average case error rate and speed of each covert channel. In order to remove transmission errors entirely, we can use hamming codes to correct single packet errors and retransmit packets containing more than one error. We estimate the effective speed after hamming codes are applied in the ``Eff. Speed'' column.

Channel	Error	Speed (w/errors)	Eff. Speed
Speaker/Mic Spkr/Mic (Amp Only)	14.4% 3.3%	6.5kbps 3.25kbps	3.71kbps 2.73kbps
Speaker/Accelerometer	0.04%	2bps	1.99bps
Vibration/Accelerometer	0.28%	2bps	1.94bps
Light/Camera	0%	1.5bps	1.5bps
User Game/Gyroscope	0%	2.5bps	2.5bps

Table 5.3: Comparing best case speed and average error rate of various covert channels before error correction coding (Speed) and after (Eff. Speed).

5.6.2 Defense

For our defense system, effectiveness and usability are top concerns. Unfortunately, performing a robust effectiveness evaluation is difficult, because there is no known or cataloged malware in the wild that, to the best of our knowledge, takes advantage of PMCC as described in this work. Therefore, we can only test the system on our own contrived examples, in all of these cases, the system was able to identify active covert channels, and propagate the taint-tag information. Our system currently breaks down in

Channel	Delivery Time (system off)	Delivery Time (system on)
Ultrasound	37.5ms	38.5ms
Speaker/Accel.	5ms	7.5ms
Vibration/Accel.	4.9ms	6.1ms
Light/Camera	8.36ms	10.1ms

Table 5.4: Overhead measurements for delivering samples from various hardware devices with and without our system running.

situations involving implicit flows. Currently, we do not have a robust solution for taint-tracking through implicit flows, but in the future, we plan to implement a better solution, at which time an effectiveness evaluation will be more interesting.

To measure the usability, we record the time it takes for the system to deliver new samples to the various devices (overhead), the effort of implementing the GuardService hooks (lines of code), the time added by our application re-writing to support implicit flows, and the usability of benign applications under our dynamic switching treatment technique.

5.6.2.1 Taint Propagation Overhead

We implemented our defense by leveraging the existing TaintDroid implementation [22]. Therefore, the memory overhead is identical to their work. The stack is basically doubled in size, due to the extra space needed to store the taint values (except for arrays which share one taint-tag for the entire structure). However, when new samples are delivered, our system introduces some time overhead, because we must lookup and propagate (bitwise OR) the taint value from the sender data to the receiver data. We implemented the ultrasound, speaker + accelerometer, vibration + accelerometer, and the light + camera channels. We measured the average time for the *dispatchSensorEvent()* method to finish delivering the data with and without our system running over ten runs. The results are presented in Table 5.4.

5.6.2.2 Implementation Effort

When designing our GuardService we aim to make it easy for developers to implement our hooks in the operating system device API files. To evaluate the effort required, we analyze our own implementation of the hooks in five of the most common device API files (i.e., the speaker, microphine, vibration motor, accelerometer & gyroscope, and the camera & flash). For each file, we measure the total lines of code before our changes, and the number of lines added to implement the hooks (the amount of lines deleted was negligable). We present the results in Fig. 5.11. The percentages given are the percentage of unchaged code for each file. In general we see that the hooks represent a relatively small change to each file. In SystemSensorManager.java our code contributes almost 25% of the total code. It's important to remember that this is actually the hooks for two devices (the accelerometer and the gyroscope), and in raw values, we only added 103 lines of code.

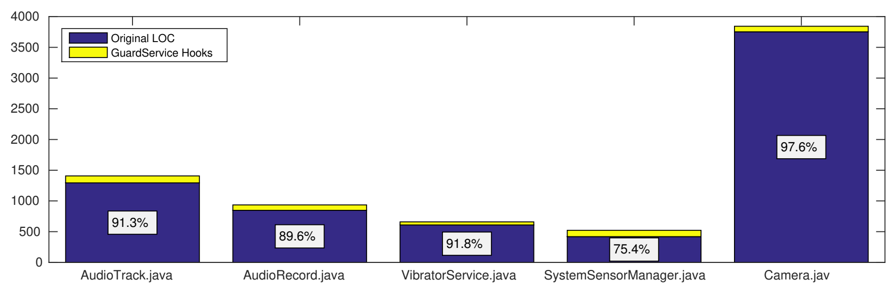


Figure 5.11: LOC required to implement GuardService hooks. Percentages given are the percentage of unmodified code.

5.6.2.3 Implicit Flow Propagation

To measure the overhead introduced by our implicit flow taint propagation solution, we manually re-wrote a representative if statement in an Android app using the d2j decompiler, and the Google provided application packaging tools. The body of the if statement contains a for loop, similar to Fig. 5.7, a primitive variable assignment, an array element assignment, a function call, and a custom object construction. To ensure the taint values are propagated within the implicit flow code bodies, we multiply the to-be-tainted variables by a tainted variable with the value one. By doing this, we guarantee that the variable will receive the taint-tag (through the direct assignment taint-droid rule), but also that the variable data will not by modified (multiplying by one has no effect). In 25 trial runs of this if statement, our re-written version, which propagates taint-tags into all of this data, introduced only 1ms of overhead on average.

5.6.2.4 Dynamic Switching

To evaluate the effectiveness of this defense treatment, we implemented a very simple speaker / microphone covert channel. The speaker makes some audible sound and the microphone records this sound. Loud and quiet periods correspond to ``1" and ``0" bits respectively, with symbols 0.5s wide. We sent the bit sequence 01010011 twice; once using our defense and once without. We plotted the data the microphone measured in Fig 5.12. We can see that when the defense is running (plotted in red), the symbols

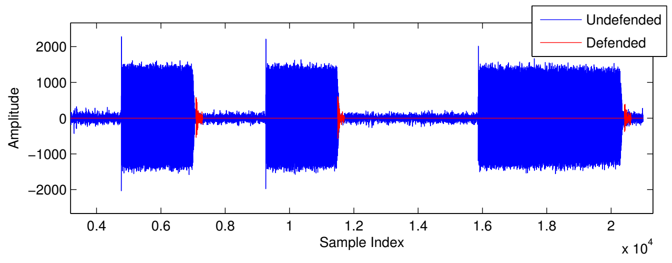


Figure 5.12: Samples gathered from the microphone while transmitting from the speaker. The data with the defense scheme running is plotted in red.

are almost completely erased. The small tail at the end of each symbol is the result of the slow propagation time of sound and hardware introduced latency. We consider this problem to be negligible, because the tails are very brief, and conceal the transmission of consecutive ``1s".

We also used a VoIP app (Skype), to measure usability. We instrumented our system on one phone and made a phone call to a second phone. Both users were able to hold a brief ten second conversation without any words being dropped or misunderstood.

5.7 Conclusion

In this chapter we present an attack on mobile smart devices that leverages physical media covert channels to enable privilege escalation and ultimately leak sensitive user information. We also present a novel defense technique that balances usability with the security and privacy concerns raised by this attack.

Chapter 6

Conclusions and Future Work

This dissertation contributes to the security and privacy of mobile devices from three different aspects. Firstly, in Chapter3 we present a solution at the application layer, which allows users to determine privately if they are near one another. This is tremendously useful because it can be used as a building block to create private variations of popular social networks, location based services, and applications available today. By using our system, users no longer have to worry about their private location information being leaked to third party advertisers, government organizations, or miscellaneous attackers and stalkers. Our system is the first to provide private location proximity detection with a configurable threshold for what is considered ``nearby."

Secondly, at the physical layer, we provide a novel mechanism for proximity networking with our ultrasound modem. Now that users are carrying their devices everywhere, a natural and intuitive expectation is to share small pieces of information with one another when together. We support this using ultrasound on commodity mobile devices, which has several advantages. It is readily deployable on nearly all commercial smart devices, because it does not require any special hardware. It is also directional, due to the nature of ultrasound. This means that users can simply aim their transmissions at the intended receiver without worrying about eavesdroppers in nearby rooms.

Thirdly, we propose and implement a defense mechanism, at the operating system layer, to defend against our novel physical media covert channels. Our defense mechanism extends existing state-of-the-art research to protect users from information leakage. Our system is able to stop all of the proposed covert channels we introduced, and can even be easily extended and configured to include other covert channels and devices.

6.1 Lessons Learned

In doing this research, I have learned several broad and high level aspects about the complex intersection of ubiquitous computing, mobile devices, security, privacy, and the state of the industry. The first is that the research world in all of these subfields is dominated by the Android operating system, as opposed to iOS, because it is open source. Therefore, researchers can inspect it, critique it, and make improvements on it. The number of top conference papers that would not be possible without a wide-spread, popular, open source mobile operating system is staggering.

Secondly, research in security and privacy on mobile devices is interesting because to achieve satisfactory results it is important to take into account the needs and wants of several different parties. Each of these parties is complex and cannot be easily explained. End users strongly prefer functionality, convenience and price over nearly every other feature including privacy and especially over security. Although something like security or privacy are major selling points, users are generally poorly educated and a) do not properly understand these features and b) usually choose price and functionality over, for example, secure but costly or private but less functional. Corporations and large enterprises need to recuperate from financial losses spent developing services and applications and therefore will usually try to monetize via advertisements. Developers are generally lazy, and will almost always choose the path of least resistance. Combining all of these self-interested actors into one ecosystem causes a variety of problems. Fortunately, researchers are able to take a less bias approach and can find success in providing solutions that strike a compromise between them. It is my hope that this dissertation provides this in at least a small way.

6.2 Future Work

Covert Channel Improvements In Chapter 5, I discuss in detail the idea of physical media covert channels and a substantial defense mechanism is proposed that aims to be easy to use and even easy to extend to support other devices. As a possible avenue for future work, I would like to explore the idea of providing an entirely different API for sensors on a user's device.

Currently, unfettered, raw sensor data can be accessed with little or no permissions by application developers. We propose an API, which would instead offer higher level features and metrics derived directly from the raw sensor data. By providing higher level ``meta'' data, we support a variety of application use cases (in fact we make the job of implementing many applications easier), while at the the same time making it significantly more difficult to use these devices in a physical media covert channel. For example, a developer may wish to build a pedometer application, which counts steps based on accelerometer data. In our system, the developer calls our API functions, which return various metrics, rather than the raw sensor data, such as a sliding window RMS value, frequency analysis, and so on, from the accelerometer. The developer can then configure the size and overlap of the window, in order to maximize the accuracy of their pedometer.

Because the developer is limited to only RMS output and a handful of other metrics on the data, developing a malicious covert channel, which uses the accelerometer (as described in Chapter 5) is much more difficult and can transmit fewer bits per second. Secondly, this API alleviates some work for the developer, who no longer must implement the window, the RMS calculation, fast Fourier transform, or the buffering of raw sample hardware interrupts. Thirdly, this API leads naturally to an alternate permission system. Instead of granting all or nothing access to the raw sensor stream, we grant access only to each metric on the data, which we can present to the user as less dangerous.

Internet of Things Mobile computing is currently thriving, but it is not the future of my research. While it is likely that mobile phones will continue to remain dominant in the computing lives of nearly everybody in the developed world, it is likely that new research challenges will diminish. Looking forward, the Internet of Things (IoT) is starting to grow in popularity at the intersection of mobile computing, ubiquitous computing, social networking, sensing, and data science. The IoT is defined as a proposed development of the Internet, in which everyday objects have network connectivity, allowing them to send and receive data. Example applications include small simple conveniences such as turning lights on in your home when you park your car in the garage, to elaborate and sophisticated sensing and actuation systems such as buildings with humidity, temperature, and vibration sensors used to aid in building maintenance and city planning.

Unfortunately, as innovation and exploration take the lead, security and privacy typically fall by the wayside. While the future of the Internet of Things is promising, the current reality is that users demand that these devices are compatible with each other, and easy to use, which means that extra security measures such as encryption, proper network configuration, and passwords are usually ignored. The possible future for the IoT is exciting and vast, which poses a promising picture for future research in this field [105].

Bibliography

[1] Saga android and ios application. `http://www.getsaga.com/`, July 2013.

[2] Tinder social network and application. https://www.gotinder.com/, November 2015.

[3] Yik yak social network and application. `http://www.yikyakapp.com/`, November 2015.

[4] Paarijaat Aditya, Viktor Erdélyi, Matthew Lentz, Elaine Shi, Bobby Bhattacharjee, and Peter Druschel. Encore: Private, context-based communication for mobile social apps. In *Proceedings of the 12th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '14, pages 135--148, New York, NY, USA, 2014. ACM.

[5] Rakesh Agrawal, Jerry Kiernan, Ramakrishnan Srikant, and Yirong Xu. Order preserving encryption for numeric data. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, SIGMOD '04, pages 563--574, New York, NY, USA, 2004. ACM.

[6] Ian F. Akyildiz, Dario Pompili, and Tommaso Melodia. Underwater acoustic sensor networks: research challenges. *Ad Hoc Networks*, 3(3):257 -- 279, 2005.

[7] Alastair R Beresford, Andrew Rice, and Nicholas Skehin. Mockdroid : trading privacy for application functionality on smartphones. *HotMobile '11 Proceedings of*

the 12th Workshop on Mobile Computing Systems and Applications, pages 49--54, 2011.

[8] Igor Bilogrevic, Murtuza Jadliwala, Kübra Kalkan, Jean-Pierre Hubaux, and Imad Aad. Privacy in mobile computing for location-sharing-based services. In *Proceedings of the 11th international conference on Privacy enhancing technologies*, PETS'11, pages 77--96, Berlin, Heidelberg, 2011. Springer-Verlag.

[9] Sven Bugiel, Lucas Davi, Alexandra Dmitrienko, Thomas Fischer, Ahmad-Reza Sadeghi, and Bhargava Shastry. Towards taming privilege-escalation attacks on android. *Proceedings of the 19th Annual Network & Distributed System Security Symposium*, 2012.

[10] Sven Bugiel, Stephan Heuser, and Ahmad-Reza Sadeghi. Flexible and fine-grained mandatory access control on android for diverse security and privacy policies. In *Proceedings of the 22Nd USENIX Conference on Security*, SEC'13, pages 131--146, Berkeley, CA, USA, 2013. USENIX Association.

[11] Swarup Chandra, Zhiqiang Lin, Ashish Kundu, and Latifur Khan. Towards a systematic study of the covert channel attacks in smartphones. *Proceedings of the 10th International Conference on Security and Privacy in Communications Networks, SecureComm*, 2014.

[12] Ke-Yu Chen, Daniel Ashbrook, Mayank Goel, Sung-Hyuck Lee, and Shwetak Patel. Airlink: Sharing files between multiple devices using in-air gestures. In *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, UbiComp '14, pages 565--569, New York, NY, USA, 2014. ACM.

[13] Chi-Yin Chow and Mohamed F. Mokbel. Enabling private continuous queries for revealed user locations. In *Proceedings of the 10th international conference on Advances in spatial and temporal databases*, SSTD'07, pages 258--273, Berlin, Heidelberg, 2007. Springer-Verlag.

[14] Landon P. Cox, Angela Dalton, and Varun Marupadi. Smokescreen: Flexible privacy controls for presence-sharing. In *Proceedings of the 5th International Conference on Mobile Systems, Applications and Services*, MobiSys '07, pages 233--245, New York, NY, USA, 2007. ACM.

[15] Justin Cranshaw, Eran Toch, Jason Hong, Aniket Kittur, and Norman Sadeh. Bridging the gap between physical location and online social networks. In *Proceedings of the 12th ACM international conference on Ubiquitous computing*, Ubicomp '10, pages 119--128, New York, NY, USA, 2010. ACM.

[16] Sajal K. Das, Krishna Kant, and Nan Zhang. *Handbook on Securing Cyber-Physical Critical Infrastructure*. Morgan Kaufmann, 2012. http://www.sciencedirect.com/science/book/9780124158153.

[17] Sanorita Dey, Nirupam Roy, Wenyuan Xu, Romit Roy Choudhury, and Srihari Nelakuditi. AccelPrint : Imperfections of Accelerometers Make Smartphones Trackable. *21st Annual Network and Distributed System Security Symposium, NDSS 2014.*, pages 23--26, February 2014.

[18] Dan Doonan. Android gps forensics. http://dandoonan.blogspot.co.uk/2013/03/mock-locations.html, March 2013.

[19] ``ECMA''. Near field communication interface and protocol (nfcip-1), June 2013. http://www.ecma-international.org/publications/standards/Ecma-340.htm.

[20] Manuel Egele, David Brumley, Yanick Fratantonio, and Christopher Kruegel. An empirical study of cryptographic misuse in android applications. *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security - CCS '13*, pages 73--84, 2013.

[21] Glenn Elert. Frequency range of human hearing, September 2014. http://hypertextbook.com/facts/2003/ChrisDAmbrose.shtml.

[22] William Enck, Peter Gilbert, Byung-Gon Chun, Landon P. Cox, Jaeyeon Jung, Patrick McDaniel, and Anmol N. Sheth. Taintdroid: An information-flow tracking system for realtime privacy monitoring on smartphones. In *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, OSDI'10, pages 1--6, Berkeley, CA, USA, 2010. USENIX Association.

[23] Yves Eonnet and Hervé Manceron. Tagattitude, April 2011. http://www.tagattitude.fr/en/products/technology.

[24] Micha Kalfon Evyatar Hemo and Roey Lehman. Wimbeep, 2012. https://sites.google.com/site/wimbeep/.

[25] Elli Fragkaki, Lujo Bauer, Limin Jia, and David Swasey. Modeling and enhancing android's permission system. In *ESORICS*, Sara Foresti, Moti Yung, and Fabio Martinelli, editors, volume 7459 of *Lecture Notes in Computer Science*, pages 1--18. Springer, 2012.

[26] V. Gerasimov and W. Bender. Things that talk: using sound for device-to-device and device-to-human communication. *IBM Syst. J.*, 39(3-4):530--546, July 2000.

[27] A. Giani, V. H. Berk, and G. V. Cybenko. Data exfiltration and covert channels. In *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, volume 6201 of *SPIE*, page 620103, May 2006.

[28] Michael Grace, Yajin Zhou, Zhi Wang, and Xuxian Jiang. Systematic detection of capability leaks in stock android smartphones. *Proceedings of the 19th Network and Distributed System Security Symposium (NDSS 2012)*, 2012.

[29] Glenn Greenwald and Wewn MacAskill. Edward snowden: the whistleblower behind the nsa surveillance revelations. `http://www.theguardian.com/world/2013/jun/09/edward-snowden-nsa-whistleblower-surveillance`, June 2013.

[30] Sven Grundeberg. Smartphone makers aim at emerging markets with low end devices, February 2014.

[31] Marco Gruteser and Dirk Grunwald. Anonymous usage of location-based services through spatial and temporal cloaking. In *Proceedings of the 1st international conference on Mobile systems, applications and services*, MobiSys '03, pages 31--42, New York, NY, USA, 2003. ACM.

[32] Jeremy J. Gummeson, Bodhi Priyantha, Deepak Ganesan, Derek Thrasher, and Pengyu Zhang. Engarde: Protecting the mobile phone from malicious nfc interactions. In *Proceeding of the 11th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '13, pages 445--458, New York, NY, USA, 2013. ACM.

[33] Michael Hanspach and Michael Goetz. On Covert Acoustical Mesh Networks in Air. *Journal of Communications*, 8(11):758--767, November 2013.

[34] Tian Hao, Ruogu Zhou, and Guoliang Xing. Cobra: color barcode streaming for smartphone systems. In *Proceedings of the 10th international conference on Mobile systems, applications, and services*, MobiSys '12, pages 85--98, New York, NY, USA, 2012. ACM.

[35] Michael Herrmann, Alfredo Rial, Claudia Diaz, and Bart Preneel. Practical Privacy-preserving Location-sharing Based Services with Aggregate Statistics. In *Proceedings of the 2014 ACM Conference on Security and Privacy in Wireless &#38; Mobile Networks*, WiSec '14, pages 87--98, New York, NY, USA, 2014. ACM.

[36] Stephan Heuser, Adwait Nadkarni, William Enck, and Ahmad-Reza Sadeghi. Asm: A programmable interface for extending android security. In *23rd USENIX Security Symposium (USENIX Security 14)*, pages 1005--1019, San Diego, CA, Aug 2014. USENIX Association.

[37] Peter Hornyack, Seungyeop Han, Jaeyeon Jung, Stuart Schechter, and David Wetherall. These aren't the droids you're looking for: Retrofitting android to protect data from imperious applications. In *Proceedings of the 18th ACM Conference on Computer and Communications Security*, CCS '11, pages 639--652, New York, NY, USA, 2011. ACM.

[38] Jun Huang, Wahhab Albazrqaoe, and Guoliang Xing. Blueid: A practical system for bluetooth device identification. In *INFOCOM*, pages 2849--2857, 2014.

[39] Google Inc. Android online documentation. http://developer.android.com/guide/index.html, February 2016.

[40] Monsoon Power Inc. Monsoon power monitor, September 2014. `https://www.msoon.com/LabEquipment/PowerMonitor/`.

[41] Suman Jana and Vitaly Shmatikov. Memento: Learning secrets from process footprints. In *Proceedings of the 2012 IEEE Symposium on Security and Privacy*, SP '12, pages 143--157, Washington, DC, USA, 2012. IEEE Computer Society.

[42] Weiwei Jiang, Denzil Ferreira, Jani Ylioja, Jorge Goncalves, and Vassilis Kostakos. Pulse: Low bitrate wireless magnetic communication for smartphones. In *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, UbiComp '14, pages 261--265, New York, NY, USA, 2014. ACM.

[43] Pravein Govindan Kannan, Seshadri Padmanabha Venkatagiri, Mun Choon Chan, Akhihebbal L. Ananda, and LiShiuan Peh. Low cost crowd counting using audio

tones. In *Proceedings of the 10th ACM Conference on Embedded Network Sensor Systems*, SenSys '12, pages 155--168, New York, NY, USA, 2012. ACM.

[44] Michael Kirk and Mike Wiser. Pbs frontline : United states of secrets, 2014.

[45] William Klieber, Lori Flynn, Amar Bhosale, Limin Jia, and Lujo Bauer. Android taint flow analysis for app sets. In *ACM SIGPLAN International Workshop on the State Of the Art in Java Program Analysis (SOAP 2014)*, June 2014. To appear.

[46] Panayiotis Kotzanikolaou, Constantinos Patsakis, Emmanouil Magkos, and Michalis Korakakis. Lightweight Private Proximity Testing for Geospatial Social Networks. *Comput. Commun.*, 73(PB):263--270, January 2016.

[47] Butler W. Lampson. A note on the confinement problem. *Commun. ACM*, 16(10):613--615, October 1973.

[48] Antii S. Lankila. Jni fft implementation in cpp, 2013. url https://github.com/lioncash/droidsound/tree/master/jni/FFT.

[49] Jungmee Lee, Sumitrajit DharRebekah Abel, Renee Banakis, Evan Grolley, Jungwh Lee, Steven Zecker, and Jonathan Siegel. Behavioral hearing thresholds between 0.125 and 20khz using depth-compensated ear simulator calibration. *Ear Hear*, May-June 2012. `http://www.ncbi.nlm.nih.gov/pubmed/22436407`.

[50] Matthew Lentz, Viktor Erdélyi, Paarijaat Aditya, Elaine Shi, Peter Druschel, and Bobby Bhattacharjee. Sddr: Light-weight, secure mobile encounters. In *23rd USENIX Security Symposium (USENIX Security 14)*, pages 925--940, San Diego, CA, Aug 2014. USENIX Association.

[51] Muyuan Li, Haojin Zhu, Zhaoyu Gao, Si Chen, Le Yu, Shangqian Hu, and Kui Ren. All Your Location Are Belong to Us: Breaking Mobile Social Networks for Automated User Location Tracking. In *Proceedings of the 15th ACM International Symposium*

on Mobile Ad Hoc Networking and Computing, MobiHoc '14, pages 43--52, New York, NY, USA, 2014. ACM.

[52] Eric Lichtblau and Katie Benner. Apple fights order to unlock san bernardino gunman's iphone, February 2015.

[53] Chia-Chi Lin, Hongyang Li, Xiaoyong Zhou, and Xiaofeng Wang. Screenmilker : How to milk your android screen for secrets. *NDSS*, pages 23--26, February 2014.

[54] Zi Lin, Denis Foo Kune, and Nicholas Hopper. Efficient private proximity testing with gsm location sketches. In *Financial Cryptography*, pages 73--88, 2012.

[55] Steve Lipner, Trent Jaeger, and Mary Ellen Zurko. Lessons from vax/svs for high-assurance vm systems. *Security Privacy, IEEE*, 10(6):26--35, Nov 2012.

[56] Benjamin Livshits and Jaeyeon Jung. Automatic mediation of privacy-sensitive resource access in smartphone applications. In *Proceedings of the 22Nd USENIX Conference on Security*, SEC'13, pages 113--130, Berkeley, CA, USA, 2013. USENIX Association.

[57] Long Lu, Zhichun Li, Zhenyu Wu, Wenke Lee, and Guofei Jiang. Chex: Statically vetting android apps for component hijacking vulnerabilities. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, pages 229--240, New York, NY, USA, 2012. ACM.

[58] Ingrid Lunden. 6.1b bsmartphone users globally by 2020, overtaking basic fixed phone subscriptions. `http://techcrunch.com/2015/06/02/6-1b-smartphone-users-globally-by-2020-overtaking-basic_-fixed-phone-subscriptions/`, June 2015.

[59] Anil Madhavapeddy, David Scott, and Richard Sharp. Context-aware computing with sound. In *UbiComp 2003: Ubiquitous Computing*, AnindK. Dey, Albrecht

Schmidt, and JosephF. McCarthy, editors, volume 2864 of *Lecture Notes in Computer Science*, pages 315--332. Springer Berlin Heidelberg, 2003.

[60] Anil Madhavapeddy, David Scott, Alastair Tse, and Richard Sharp. Audio networking: The forgotten wireless technology. *IEEE Pervasive Computing*, 4, 2005.

[61] Claudio Marforio, Aurélien Francillon, Srdjan Capkun, Srdjan Capkun, and Srdjan Capkun. Application collusion attack on the permission-based security model and its implications for modern smartphone systems. Technical Report 724, Department of Computer Science, ETH Zurich, 2011.

[62] Hosei Matsuoka, Yusuke Nakashima, and Takeshi Yoshimura. Acoustic communication system using mobile terminal microhphones. In *NTT DoCoMo Technical Journal Vol. 8 No. 2*, 2006.

[63] Jonathan K. Millen. Covert channel capacity. *IEEE Symposium on Security and Privacy 1987*, 1987.

[64] Pongsin Poosankam Min Gyung Kang, Stephen McCamant and Dawn Song. Dta++: Dynamic taint analysis with targeted control-flow propagation. *NDSS'11*, 2011.

[65] mobiThinking. Global mobile statistics 2014 part a: Mobile subscribers; handset market share; mobile operators, June 2014.

[66] Mohamed F. Mokbel, Chi-Yin Chow, and Walid G. Aref. The new casper: query processing for location services without compromising privacy. In *Proceedings of the 32nd international conference on Very large data bases*, VLDB '06, pages 763--774. VLDB Endowment, 2006.

[67] Moni Naor and Benny Pinkas. Efficient oblivious transfer protocols. In *Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '01,

pages 448--457, Philadelphia, PA, USA, 2001. Society for Industrial and Applied Mathematics.

[68] Arvind Narayanan, Narendran Thiagarajan, Mugdha Lakhani, Michael Hamburg, and Dan Boneh. Location privacy via private proximity testing. In *NDSS*. IEEE, The Internet Society, 2011.

[69] A.E. Nergiz, M.E. Nergiz, T. Pedersen, and C. Clifton. Practical and secure integer comparison and interval check. In *Social Computing (SocialCom), 2010 IEEE Second International Conference on*, pages 791 --799, aug. 2010.

[70] Jakob Nielsen. Response times: The 3 important limits. `http://www.nngroup.com/articles/response-times-3-important-limits/`, January 1993.

[71] E. Novak and Qun Li. Near-pri: Private, proximity based location sharing. In *INFOCOM, 2014 Proceedings IEEE*, pages 37--45, April 2014.

[72] Ed Novak. Near-pri online code repository. `https://github.com/deadmund/Nearby`, November 2012.

[73] Ed Novak, Yutao Tang, Zijiang Hao, Qun Li, and Yifan Zhang. Physical media covert channels on smart mobile devices. In *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, UbiComp '15, pages 367--378, New York, NY, USA, 2015. ACM.

[74] Alexandra-Mihaela Olteanu, Kévin Huguenin, Reza Shokri, and Jean-Pierre Hubaux. *Privacy Enhancing Technologies: 14th International Symposium, PETS 2014, Amsterdam, The Netherlands, July 16-18, 2014. Proceedings*, chapter Quantifying the Effect of Co-location Information on Location Privacy, pages 184--203. Springer International Publishing, Cham, 2014.

[75] M. Ongtang, S. McLaughlin, W. Enck, and P. McDaniel. Semantically rich application-centric security in android. *2009 Annual Computer Security Applications Conference*, 2009.

[76] Clemens Orthacker, Peter Teufl, Stefan Kraxberger, Günther Lackner, Michael Gissing, Alexander Marsalek, Johannes Leibetseder, and Oliver Prevenhueber. Android security permissions - can we trust them? In *MobiSec*, pages 40--51, 2011.

[77] Gultekin Ozsoyoglu, David A. Singer, and Sun S. Chung. Anti-tamper databases: Querying encrypted databases. In *In Proc. of the 17th Annual IFIP WG 11.3 Working Conference on Database and Applications Security, Estes Park*, pages 4--6, 2003.

[78] Stavros Papadopoulos, Spiridon Bakiras, and Dimitris Papadias. Nearest neighbor search with strong location privacy. *Proc. VLDB Endow.*, 3(1-2):619--629, September 2010.

[79] Constantinos Patsakis, Panayiotis Kotzanikolaou, and Mélanie Bouroche. *Security and Trust Management: 11th International Workshop, STM 2015, Vienna, Austria, September 21-22, 2015, Proceedings*, chapter Private Proximity Testing on Steroids, pages 172--184. Springer International Publishing, Cham, 2015.

[80] Brett Paulson. Zoosh, 2011. `http://www.naratte.com/`.

[81] Nicholas J. Percoco and Sean Schutle. Adventures in bouncerland. https://www.youtube.com/watch? v=-Kcy-Idh5h0, July 2012.

[82] Benny Pinkas, Thomas Schneider, Gil Segev, and Michael Zohner. Phasing: Private Set Intersection Using Permutation-based Hashing. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 515--530, Washington, D.C., August 2015. USENIX Association.

[83] Benny Pinkas, Thomas Schneider, and Michael Zohner. Faster Private Set Intersection Based on OT Extension. In *Proceedings of the 23rd USENIX Conference on Security Symposium*, SEC'14, pages 797--812, Berkeley, CA, USA, 2014. USENIX Association.

[84] Sebastian Poeplau, Yanick Fratantonio, Antonio Bianchi, Christopher Kruegel, and Giovanni Vigna. Execute this! analyzing unsafe and malicious dynamic code loading in android applications. *NDSS*, pages 23--26, February 2014.

[85] Tatiana Pontes, Marisa Vasconcelos, Jussara Almeida, Ponnurangam Kumaraguru, and Virgilio Almeida. We know where you live: privacy characterization of foursquare behavior. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*, UbiComp '12, pages 898--905, New York, NY, USA, 2012. ACM.

[86] Manoj Prabhakaran. Homomorphic encryption, lecture 15. `courses.engr.illinois.edu/cs598man/fa2011/slides/ac-f11-lect15.pdf` , Fall 2011.

[87] Farzana Rahman, Md. Endadul Hoque, Ferdaus Ahmed Kawsar, and Sheikh Iqbal Ahamed. Preserve your privacy with pco: A privacy sensitive architecture for context obfuscation for pervasive e-community based applications. In *Proceedings of the 2010 IEEE Second International Conference on Social Computing*, SOCIALCOM '10, pages 41--48, Washington, DC, USA, 2010. IEEE Computer Society.

[88] Venkata N. Padmanabhan Rajalakshmi Nandakumar, Krishna Kant Chintalapudi and Ramarathnam Venkatesan. Dhwani : Secure peer-to-peer acoustic nfc. In *Proceedings of ACM SIGCOMM 2013*, Sigcomm '13, New York, NY, USA, 2013. ACM.

[89] Siegfried Rasthofer, Steven Arzt, and Eric Bodden. A machine-learning approach for classifying and categorizing android sources and sinks. *NDSS*, pages 23--26, February 2014.

[90] Bill Ray. Crafy app lets phones send data by ultrasound with speakers, mics, November 2012. http://www.theregister.co.uk/2012/11/08/ultrasonic_bonking/.

[91] Marguerite Reardon. Nfc mobile payments disappoint while money transfers boom, June 2013. http://www.cnet.com/news/nfc-mobile-payments-disappoint-while-money_transfers-boom/.

[92] Hubert Ritzdorf. Analyzing covert channels on mobile devices. Master's thesis, ETH Zurich, 2012.

[93] Franziska Roesner, Tadayoshi Kohno, Alexander Moshchuk, Bryan Parno, Helen J. Wang, and Crispin Cowan. User-driven access control: Rethinking permission granting in modern operating systems. *2012 IEEE Symposium on Security and Privacy*, pages 224--238, 2012.

[94] Nirupam Roy and Romit Roy Choudhury. Ripple ii: Faster communication through physical vibration. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, pages 671--684, Santa Clara, CA, March 2016. USENIX Association.

[95] Roman Schlegel, Kehuan Zhang, and Xiaoyong Zhou. Soundcomber: A stealthy and context-aware sound trojan for smartphones. *Proceedings of the 18th Annual Network and Distributed System Security Symposium (NDSS)*, pages 17--33, 2011.

[96] Reza Shokri, George Theodorakopoulos, Jean-Yves Le Boudec, and Jean-Pierre Hubaux. Quantifying location privacy. In *Proceedings of the 2011 IEEE Symposium on Security and Privacy*, SP '11, pages 247--262, Washington, DC, USA, 2011. IEEE Computer Society.

[97] Reza Shokri, Georgios Theodorakopoulos, Carmela Troncoso, Jean-Pierre Hubaux, and Jean-Yves Le Boudec. Protecting Location Privacy: Optimal Strategy

against Localization Attacks. In *19th ACM Conference on Computer and Communications Security*, 2012.

[98] Laurynas Siksnys, Jeppe Rishede Thomsen, Simonas Saltenis, and Man Lung Yiu. Private and flexible proximity detection in mobile social networks. In *Proceedings of the 2010 Eleventh International Conference on Mobile Data Management*, MDM '10, pages 75--84, Washington, DC, USA, 2010. IEEE Computer Society.

[99] Aaron Smith. U.s. smartphone use in 2015, April 2015.

[100] Vadim Sokolovsky. Illiri, July 2013. `http://www.illiri.com/`.

[101] BSB Mobile Solutions. decibel application. url= https://play.google.com/store/apps/details?id=bz.bsb.decibel&hl=en.

[102] E.M. Sozer, M. Stojanovic, and J.G. Proakis. Underwater acoustic networks. *Oceanic Engineering, IEEE Journal of*, 25(1):72--83, 2000.

[103] Ethem Mutlu Sözer and Milica Stojanovic. Reconfigurable acoustic modem for underwater sensor networks. In *Proceedings of the 1st ACM International Workshop on Underwater Networks*, WUWNet '06, pages 101--104, New York, NY, USA, 2006. ACM.

[104] Mudhakar Srivatsa and Michael Hicks. Deanonymizing mobility traces: Using social network as a side-channel. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, October 2012.

[105] J. A. Stankovic. Research directions for the internet of things. *IEEE Internet of Things Journal*, 1(1):3--9, Feb 2014.

[106] Wai-Tian Tan, Mary Baker, Bowon Lee, and Ramin Samadani. The sound of silence. In *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems*, SenSys '13, pages 19:1--19:14, New York, NY, USA, 2013. ACM.

[107] Go Launcher Dev Team. Go launcher ex application. https://play.google.com/store/apps/details?id=com.gau. go.launcherex&hl=en, May 2015.

[108] Robert Templeman, Zahid Rahman, David Crandall, and Apu Kapadia. Placeraider: Virtual theft in physical spaces with smartphones. *NDSS*, September 2012.

[109] Rafael Tonicelli, Bernardo Machado David, and Vinícius de Morais Alves. Universally composable private proximity testing. In *Proceedings of the 5th international conference on Provable security*, ProvSec'11, pages 222--239, Berlin, Heidelberg, 2011. Springer-Verlag.

[110] Baltimoore County University of Maryland. Umbc implementation of paillier encryption in java. `http://www.csee.umbc.edu/~kunliu1/research/Paillier.html`, September 2012.

[111] Tielei Wang, Kangjie Lu, Long Lu, Simon Chung, and Wenke Lee. Jekyll on ios: When benign apps become evil. In *Proceedings of the 22Nd USENIX Conference on Security*, SEC'13, pages 559--572, Berkeley, CA, USA, 2013. USENIX Association.

[112] Yifei Wang, Srinivas Hariharan, Chenxi Zhao, Jiaming Liu, and Wenliang Du. Compac: Enforce component-level access control in android. In *Proceedings of the 4th ACM Conference on Data and Application Security and Privacy*, CODASPY '14, pages 25--36, New York, NY, USA, 2014. ACM.

[113] Wei Wei, Fengyuan Xu, and Qun Li. Mobishare: Flexible privacy-preserving location sharing in mobile online social networks. In *INFOCOM*, Albert G. Greenberg and Kazem Sohraby, editors, pages 2616--2620. IEEE, 2012.

[114] Wikipedia. Paillier encryption. `http://en.wikipedia.org/wiki/Paillier_encryption`, September 2012.

[115] Wikipedia. Wikipedia hamming code article, November 2013. `http://en.wikipedia.org/wiki/Hamming_code`.

[116] Wikipedia. Magnetic stripe card, September 2014. `http://en.wikipedia.org/wiki/Magnetic_stripe_card`.

[117] "wikipedia". Wikipedia 3g article. `http://en.wikipedia.org/wiki/3G`, January 2014.

[118] Jack Wills, Wei Ye, and John Heidemann. Low-power acoustic modem for dense underwater sensor networks. In *Proceedings of the 1st ACM International Workshop on Underwater Networks*, WUWNet '06, pages 79--85, New York, NY, USA, 2006. ACM.

[119] Chiachih Wu, Yajin Zhou, Kunal Patel, Zhenkai Liang, and Xuxian Jiang. Airbag : Boosting smartphone resistance to malware infection. *NDSS*, pages 23--26, February 2014.

[120] Liang Xiao, Qiben Yan, Wenjing Lou, Guiquan Chen, and Y. Thomas Hou. Proximity-Based Security Techniques for Mobile Users in Wireless Networks. *Trans. Info. For. Sec.*, 8(12):2089--2100, December 2013.

[121] Rubin Xu, Hassen Saïdi, and Ross Anderson. Aurasium: Practical policy enforcement for android applications. In *Proceedings of the 21st USENIX Conference on Security Symposium*, Security'12, pages 27--27, Berkeley, CA, USA, 2012. USENIX Association.

[122] M. Xue, Y. Liu, K. W. Ross, and H. Qian. I know where you are: Thwarting privacy protection in location-based social discovery services. In *Computer Communications Workshops (INFOCOM WKSHPS), 2015 IEEE Conference on*, pages 179--184, April 2015.

[123] yahoo. Yahoo weather application. https://play.google.com/store/apps/details?id= com.yahoo.mobile.client.android.weather&hl=en, May 2015.

[124] Jie Yang, Yingying Chen, W. Trappe, and J. Cheng. Detection and localization of multiple spoofing attackers in wireless networks. *IEEE Transactions on Parallel and Distributed Systems*, 24(1):44--58, 2013.

[125] Lan Zhang and Xiang-Yang Li. Message in a sealed bottle: Privacy preserving friending in social networks. *CoRR*, abs/1207.7199, 2012.

[126] Yuan Zhang, Min Yang, Bingquan Xu, Zhemin Yang, Guofei Gu, Peng Ning, X. Sean Wang, and Binyu Zang. Vetting undesirable behaviors in android apps with permission use analysis. In *Proceedings of the 20th ACM Conference on Computer and Communications Security (CCS)*, November 2013.

[127] Xiuyuan Zheng, Hongbo Liu, Jie Yang, Yingying Chen, R.P. Martin, and Xiaoyan Li. A study of localization accuracy using multiple frequencies and powers. *Parallel and Distributed Systems, IEEE Transactions on*, 25(8):1955--1965, Aug 2014.

[128] Ruogu Zhou and Guoliang Xing. nshield: A noninvasive nfc security system for mobiledevices. In *Proceedings of the 12th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '14, pages 95--108, New York, NY, USA, 2014. ACM.

[129] Xiaoyong Zhou, Soteris Demetriou, Dongjing He, Muhammad Naveed, Xiaorui Pan, XiaoFeng Wang, Carl A. Gunter, and Klara Nahrstedt. Identity, location, disease and more: Inferring your secrets from android public resources. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, CCS '13, pages 1017--1028, New York, NY, USA, 2013. ACM.