PUPy: A Generalized, Optimistic Context Detection Framework

by

Matthew Rafuse

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Masters of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2021

Abstract

In modern life, the usage of smart devices like smartphones and laptops that allow for access to information, communication with friends and colleagues and other indispensable services has become ubiquitous. People have gradually taken to performing more and more of their daily tasks on and through these devices. Therefore, all modern smart devices employ some form of authentication to ensure that access to this confidential data by the wrong person is avoided. This authentication method is usually some form of explicit authentication, which can be detrimental to the user's experience, often leading to users forgoing authentication entirely.

Implicit authentication aims to limit the amount of explicit authentications that are necessary for the user, using passive approaches to authenticate the user instead. Context detection frameworks aim to reduce explicit authentications by disabling explicit authentication entirely when appropriate. Since these two approaches are not mutually exclusive, there exist frameworks that will use the context around them to make decisions when authenticating on which approach to use. This combination of context detection with implicit authentication is the approach taken in this work, though we focus mainly on the context detection part of this hybrid approach.

We aim to build upon existing works through wider applicability, better accuracy through numerous data sources, and most importantly, an optimistic approach to context detection. We build a framework based on the assumption that the absence of data can, in some cases, be taken as a sign the context is safe. This optimistic approach provides a less secure method of determining the context of the device, but simultaneously provides a significantly improved user experience.

In this thesis, we outline a theoretical context detection framework that is based on a novel set of values. These values are called privacy, unfamiliarity and proximity, each describing a different aspect of the current context. Privacy tracks the privacy of the current context, while unfamiliarity tracks how many unfamiliar people are around. Finally, proximity estimates the distance between the device and the user. These values are calculated using a method we devise that better adapts to different contexts. We provide an Android implementation of the framework, including an API that allows other developers to contribute modules to the system. These modules can provide additional input data for PUPy, or build functionality that uses the calculated values. Finally, we evaluate the theoretical framework, using two datasets - Cambridge/Haggle and the MDC dataset. We conduct visual and statistical analysis of how the system functions using data from the datasets. Through this analysis, we find that PUPy compares favourably to existing works, permitting a 77% reduction on average in the number of explicit authentications.

Acknowledgements

I would like to thank my supervisor, Urs Hengartner, for all the support through the duration of my program - their help was instrumental in getting here. I'd also like to thank my committee members Yousra Aafer and N. Asokan for all feedback provided on this thesis. Additionally, I'd like to thank everyone in the CrySP lab for helping all the way through my research program, be it academic support or just hanging out. Also, thank you for not kicking me out despite me being a constant distraction. I'd specifically like to thank Connor, Thomas, Lindsey, Miti, Simon, John and Rasoul for being great colleagues and even better friends. You guys rock. This work benefitted from the use of the CrySP RIPPLE Facility at the University of Waterloo. (Portions of) the research in this paper used the MDC Database made available by Idiap Research Institute, Switzerland and owned by Nokia. We gratefully acknowledge the support of the Waterloo-Huawei Joint Innovation Laboratory for funding this research.

Dedication

This is dedicated to my parents, for enabling me to undertake this endeavor in the first place, and for always being there for me, regardless of any mistake. Also, my cat Freida, for existing.

Table of Contents

List of Figures

List of Tables

It’s a dangerous business, Frodo, going out your door. You step onto the road, and if you don’t keep your feet, there’s no knowing where you might be swept off to.

— J.R.R. Tolkien, The Lord of the Rings

Chapter 1

Introduction

In modern life, the usage of smart devices like smartphones and laptops that allow for access to information, communication with friends and colleagues and other indispensible services has become ubiquitous. Due to the massive utility and convenience of such devices, people have gradually taken to performing more and more of their daily tasks on and through these devices. This increase in usage has led to these devices containing vast quantities of confidential user data, from private correspondence to banking statements. Therefore, all modern smart devices employ some form of authentication to ensure that access to this confidential data by the wrong person is avoided.

Modern authentication methods are generally knowledge-based in the form of PINs, pattern locks or typed passwords, or biometric in the form of face recognition, fingerprint sensors or iris sensors. These authentication methods can be an annoyance to users, as they impede immediate use of the device while authenticating. Instead, users will often forgo any authentication for the sake of convenience [1, 11, 16, 17].

In response to this trend, the field of implicit authentication has arisen. Implicit authentication aims to limit the amount of explicit authentications that are necessary for the user, using passive approaches to authenticate the user instead. These approaches take many forms but generally revolve around authenticating the user passively through the use of context data. To learn about the context the device is in, it can employ the wide range of sensors available on the device. The context information collected can take many forms - accelerometer data, network and Bluetooth scans, ambient temperature, proximity readings, GPS locations and location estimates, and many others, depending on the sensor suite built into the device in question.

Implicit authentication frameworks generally aim to use this sensor data to identify the user of the device. Existing frameworks exist that authenticate the user based on breath [8], gait [9], voice [38, 20, 21], and many others [28, 46, 13]. Often, these implicit authentication frameworks fall back on active authentication if they cannot satisfactorily authenticate the user. Another approach to limiting explicit authentications through the use of the device context has more recently arisen in the form of context detection frameworks.

In contrast to implicit authentication, context detection frameworks aim to reduce explicit authentications by disabling explicit authentication entirely (or offering an easier or implicit alternative form) when appropriate in contexts deemed “safe” [10, 15]. These frameworks often make the argument that in contexts that are designated “safe”, it is significantly less likely that an unauthorized user will attempt to use the device, and therefore authentication is redundant. The key difference between these approaches is that implicit authentication will use context data to examine the *user*, while context detection frameworks use it to examine the *environment or device*. Some frameworks, however, do both.

Since these two approaches are not mutually exclusive, there exist frameworks [20, 46] that will use the context around them to make decisions when authenticating on which approach to use. In low risk, trusted contexts, implicit authentication provides some security while allowing for a more convenient user experience. In less trustworthy, high risk contexts, the framework can enable explicit authentication providing the user with security when it is most needed [18, 20]. This combination of context detection with implicit authentication is the approach taken in this work.

In this thesis, we introduce a new context detection framework, PUPy. PUPy generates a large amount of context information through a simple interface, by taking in sensor data and condensing it into three values - privacy, unfamiliarity, and proximity, which each describe a different aspect of the context. Privacy tracks the privacy of the current context, while unfamiliarity tracks how many unfamiliar people are around. Finally, proximity estimates the distance between the device and the user. The first letters of these values form the basis of the name, PUPy. With the introduction of PUPy, we aim to improve on existing approaches in the following key ways:

Wider Applicability Most existing context detection frameworks [36, 15, 32, 18] do not provide a more general and extensible approach to context detection, and often aim only to achieve specific goals, most frequently deciding on which authentication method best fits the current context. We aim to provide a system that provides a more generalized description of the context, which can be harnessed both for authentication and other applications.

To this end, the approach taken in this work aims to be modular in nature, and aims to provide context information that is widely applicable to many functions. The resulting context parameters can be propagated through a subscription system to many different functionality modules. In addition, input will also be made modular, to allow for input modules, so that later advancements in context detection can be easily retrofitted. This modular approach will ensure long-term compatibility with future works.

Better Accuracy By allowing many different input modules to provide data to the framework, we can draw from many different sources for context data. By combining many different sources, the description of the current context can be made more accurate than if we rely on a single source.

Additionally, we also aim to allow functionality modules to more accurately respond to contexts by expanding the number of data points made available to them. While existing works tend to condense context data into a single value [38] (or occasionally two [20, 15]), we will expose three variables to describe the context, allowing for a fuller picture of the various facets of the user context. Furthermore, calculation of these values is further adapted to fit the context, so the framework can react more accurately to different situations.

Optimistic Context Detection The fundamental belief espoused in existing works is a pessimistic one [15, 32]. They tend to take the absence of data as a sign that the context is unsafe. This work builds on the opposite assumption - that the absence of data can, in some cases, be taken as a sign the context is safe. For example, an empty street will have no nearby people to classify as trusted or familiar, and no data to process, leading to some frameworks marking the context as unsafe. In fact, the lack of people can tell us that it is perfectly safe to disable authentication, as there is no one around that could attempt to access the device. This optimistic approach provides a slightly less secure method of determining the context of the device, but simultaneously provides a significantly improved user experience, a tradeoff often made in implicit authentication.

1.1 Structure and Contributions

In Chapter 2, we look at some of the prerequisite concepts and context for the contributions later on. In Chapter 3, we discuss the fundamental motivation for our contributions, and what drives our differences to existing works. In Chapter 4, we discuss some existing works and how they differ from our system. Chapter 5, Chapter 6 and Chapter 7 examine our main contributions.

Contribution 1: Chapter 5 outlines a novel theoretical framework that improves on existing works through wider applicability, better accuracy and optimistic context detection. These improvements are based on a novel system of values describing a context, calculated using a novel method that better adapts to different contexts.

Contribution 2: Chapter 6 discusses our Android implementation. This application implements the theoretical framework for use on modern Android devices, including an API that allows other users to contribute modules to the system.

Contribution 3: Chapter 7 evaluates PUPy. We evaluate these contributions against existing works, finding that PUPy allows for a 77% reduction in user authentications, comparing favourably to existing works. This evaluation is done using two datasets - the first is a set of Bluetooth sightings collected over a few weeks in Cambridge [39], and the second is a large database of sensor data detected by devices over several years [25, 27].

This work concludes with conclusion and future work in Chapter 8.

Chapter 2

Background

In this chapter, we will discuss some concepts and definitions vital to properly understanding this work. We will focus on high level definitions, leaving more narrow and complex information to be introduced when it is used instead.

2.1 Device Sensors

Modern mobile devices have a large suite of sensors at their disposal for learning about the world around them and to provide functionality. We will quickly enumerate and describe some of the most important sensors for our purposes.

GPS GPS-enabled devices permit users to use the device to get a highly accurate estimate of their current location. For our purposes, this is useful when identifying the context the device is in. At a high level, simply knowing if the device is moving or not gives us important clues. Additionally, if the device is stationary, location data could be cross-referenced with data describing what is at that location, be it whether the area is commercial or residential, in a dense city or empty country.

WiFi Network All modern smartphones can connect to nearby WiFi networks in order to access the Internet. However, of interest to us is the ability (particularly on Android devices) to discern a rough location of a WiFi network through third-party services. This allows a significantly more power-efficient method of determining where the device is located.

Mobile Networks Similarly to WiFi networks, mobile networks allow devices to connect to the wider Internet, but from much further away, covering more area. Similarly to WiFi, the current tower connected to can give a very rough idea of where on earth the device is located, albeit in a much less accurate manner than other approaches. The absence of a strong mobile signal can also give context clues as to where the device is. Since most dense areas have strong mobile reception, the lack thereof can be seen as a hint the device is either underground or in a very rural area.

Bluetooth Network Scanning for nearby Bluetooth devices allows the device to find other Bluetooth devices to connect to, or to get a sense of what type and how many devices there are. This is useful when getting a sense of how crowded the context is, by assuming most Bluetooth devices are attached to a unique user. At a high level, contexts with no other Bluetooth devices are more likely to be less crowded and more private than a context chock full of devices. In this work, we operate on the assumption that everyone carries at least one detectable device, so that we can assume a context with no bluetooth devices is a context without any other people.

Proximity Sensor Many modern smartphones have a proximity sensor installed. This allows the device to sense if it is close (within roughly ten centimetres) of another object. It is often used for sensing if the device is in a pocket or bag, which can be useful contextual information.

Microphone The microphone built into smartphones primarily allow users to place voice calls or talk through video chat, but are often used for other purposes as well. In our use case, the microphone can be used to sample the ambient noise level of a context, giving us an idea as to the crowdedness of the context. Additionally, microphones can be used as implicit authentication through the user of voice recognition or other approaches [8].

2.2 Context Detection

Before discussing context detection, first let us define what is meant by context. The device context is a shorthand for the situation the device is in. This includes any number of factors - how many people are around, how close it is to the owner, where the device is physically located, is the location private or public, is the device mobile or stationary, et cetera. The context of the device is primarily determined through device sensors such as the microphone or GPS, but could also be inferred through past user behaviour or general user statistics. Examples of contexts would include at home on the coffee table, in the owner's backpack, or on the seat next to the owner on public transportation.

Context detection can have many applications, such as sensing the crowdedness of public transportation [35], or in implicit authentication and improving the user experience. In our case, we will be focusing mainly on using context detection in order to adapt the behaviour of various applications the user interacts with. Chief among these usages will be authentication - namely, on adapting the behaviour of implicit authentication frameworks.

2.3 Implicit Authentication

Implicit authentication is a fairly mature field of research. It focuses on the use of device sensors and biometrics to authenticate users passively, without the use of explicit modes of authentication like a PIN or pattern lock. It is aimed at everyday users, who are generally more concerned with ease of use over security. As a compromise, implicit authentication makes the process as seamless as possible, at the cost of accuracy and overall security. The argument put forward is that some security is better than an alternative of no security. While the proposed framework is not specific to authentication by design, it is one application we examine in depth in our evaluation, and one we aim to improve upon.

In this work, we use a slightly broader definition of implicit authentication. While traditional implicit authentication is limited to exploiting biometric or behavioural differences between users, we take implicit authentication to mean any method through which context information is used to reduce the number of explicit authentications a user must perform.

In this thesis, we reconsider the fundamental compromise implicit authentication makes between security and usability, and we examine how we can further push the envelope. We aim to make a further trade between security and usability through the use of an optimistic approach to context detection.

Chapter 3

Motivation

In general, the motivation for PUPy is threefold. Our first aim is to provide a context detection and authentication system that, in contrast to existing works, is fundamentally *optimistic* instead of *pessimistic*. In existing works, authors discuss how the point of these systems is to provide a more user-friendly approach to authentication. As a whole, existing works aim to provide some measure of protection against device misuse or theft without compromising the user experience. This comes at the cost of effectiveness, though. By making a system more user friendly, it often reduces the effectiveness of the protection, leading to situations where devices can be accessed by malicious actors.

However, since some protection is better than no protection, this tradeoff makes sense. If a module foils 90% of adversaries with a small impact on user experience, it is more likely the user will use it. In contrast, a 100% effective form of authentication that ruins the user experience may be rejected, leading to the user using no protection at all, allowing 100% of attacks through. The correct choice in this tradeoff is obvious.

Despite this logic, though, existing approaches all tend to stick to a pessimistic threat model - assume the device is always in danger, unless there is evidence to the contrary. Conversely, this work aims to provide the opposite approach - assume the device is safe, unless there is evidence to the contrary. This allows for a far better user experience, and one that better matches a user's own perception of the situation than a fundamentally pessimistic model.

To expand on this, consider how people react to various contexts. In general, we do not constantly assume we are in danger all the time. Instead, we tend to assume we are safe, unless we are convinced otherwise through our perception of the context. For example, when walking alone in the woods, we are not constantly on edge, ready to respond to any threat. That response is only triggered when we sense there may be a threat - perhaps the sighting of a bear, or the discovery of wolf tracks. The approach taken by existing works is exactly opposite this - the device is constantly locked, unless the given system decides it is safe to relax protections. Our approach better aligns device behaviour with user behaviour.

The second aim of this work is to allow a more modular approach to context detection. Existing context detection systems tend to be monolithic - they collect, process and act on the data alone, without making the data known for the use of other applications, making extensibility of the system difficult. PUPy attempts to take a more distributed approach to context detection, allowing extension of the system through a modular structure.

This modular structure allows developers to incorporate the latest advancements in context detection. Coupled with the abstract nature of the values the system creates, any new advancements can easily be adapted to work with the system. This structure also allows us to draw from multiple data points when learning about the context.

The third aim of this work is to bring a more nuanced approach to context detection. Many existing works tend to operate in binary - instead of providing a description of the context, they collect and process the data, make a decision, and act on it. Instead of allowing applications to react to the context themselves, the judgement is done by the system. This often comes in the form of locking the device, or a binary safe/unsafe result.

This work aims to expose more information about the context that will allow for a more nuanced approach to various situations (see next section), as well as allowing a better description of the sort of context the device is in. However, this must be done carefully. We cannot allow the raw data to be provided to untrusted custom modules, as they can learn too much about the user this way. Instead, the data provided must describe the context accurately, but in general terms so as to not allow privacy leakage.

3.1 Motivating Examples

In this section, we will outline and discuss some different example scenarios, which will hopefully help further reinforce the motivation for this work. We will first discuss how implicit authentication and context detection can be used in unison. We will then explore examples of how optimistic context detection can provide a better user experience than pessimistic detection. Finally, we will discuss some failing examples, which will help elucidate the limits or drawbacks of our approach.

3.1.1 Context Detection and Implicit Authentication

First we will look at some examples of how context detection and implicit authentication are complementary.

Backpack To start off, let us consider a really basic example of how context can complement implicit authentication, without stepping on its toes. Imagine a scenario in which an existing implicit authentication method is being employed - say, proximity sensor based authentication. If the owner is authenticated, uses the phone, and then places the phone in their backpack, we can assume that as long as the backpack stays with the owner, the next person to use the phone will also be the owner.

If our module can report to the authentication module how close the device is to the user, the module can use that information to refine its approach to authentication. When the distance between the user and the device (and thus backpack) increases, it becomes increasingly unlikely that the next user will be the owner. This is an example where providing some sense of the device's context can improve the user experience. Instead of replacing the authentication module entirely, we provide it with additional contextual data, allowing it to adapt. This is what we aim to accomplish with the context engine.

Taxi vs. Personal Car Consider a case where the owner gets into some sort of vehicle, while the owner is authenticated. Authentication in a vehicle is fairly difficult, and there may not be an authentication method that can continue to authenticate the user. However, consider the difference in contexts where the user is in their personal vehicle, and in a Taxi or Uber. The likelihood of the device falling into the hands of a malicious actor is far higher in a Taxi or Uber, and we should modulate our approach to authentication accordingly.

If the device is able to detect when the owner is entering a taxi versus their personal vehicle, the authentication module can adapt to that particular environment when determining what authentication method is necessary. In the taxi, we should continue to authenticate the users to a high degree of certainty due to the high threat level. Conversely, consider the owner's personal vehicle. If they are the only person in the vehicle, it is unnecessary to waste time or energy on strong authentication methods.

The trick, of course, is deciding what type of vehicle is being entered. This could be done through tracking user behaviour, ambient noise, or perhaps recent application usage - use of Uber/Lyft prior to entering the vehicle can give us important clues about the context the device is in. Currently, this example is not something we aim to do, but PUPy is certainly built in a way that would make it quite possible to implement without issues.

Continuous Changes in Context Imagine if the owner of the device works a normal desk job, and frequently leaves their desk to talk to coworkers nearby and in other parts of the building about various subjects. When they do this, they often leave their device at their desk.

Without the use of context, we can still implement a fairly compelling authentication module. If the user is nearby, turn off explicit authentication. If the user has left their desk, enable it. But bringing context into the equation allows us to construct a far richer system. To illustrate this, let us consider a normal day and how context can interact with authentication.

Let us start in the morning. Perhaps the owner is a bit of an early bird, and often arrives before many of their coworkers. In this case, we can use the low level of crowding to modulate the strength of authentication required - perhaps, if the owner leaves for a short amount of time (getting coffee, a quick chat, etc.) the device can stay unlocked without further authentication methods being activated, conserving battery life. As more coworkers file in, the device registers the increased crowding and requires a higher standard of authentication to be met in order for the device to bypass active authentication methods.

Conversely, if the owner often works late into the night, the opposite may happen. As workers file out and the building empties, the device can require less and less rigorous authentication.

3.1.2 Context Detection and Other Applications

Context is useful beyond simply extending authentication or deauthenticating users. Since context is, at its core, interested in tracking the relationship between the device and the owner with regards to the environment they are in, it can be used for scenarios other than authentication.

Public Transport Public transport is another interesting case where context detection could be stretched to its absolute limit. Imagine the owner is on a commuter train, headed from the end of the line into downtown. When the owner first boards, the train is essentially completely empty - their car has no other person on it. However, at each stop, the train picks up more and more passengers, until the train is eventually standing room only as it arrives downtown.

Existing methods of authentication would treat the entire train ride exactly the same way. The framework may consider this a public area, and require strong authentication methods to be used, limiting the usability of the framework. However, the context does not remain the same throughout the trip, and we could adapt the authentication method used throughout.

At the beginning, the context is functionally identical to the owner's personal vehicle - the owner is alone, with no other people in the immediate area. The device can safely be left alone. However, as the train continues along the line, the context becomes more and more dangerous, with device intrusion and theft attempts more and more likely. Unlike a trip in the owner's vehicle or a taxi, throughout the duration of the trip the context changes. Is it possible for the device to recognize this change in context, considering how gradually it occurs?

This type of slow increase in threat level is pervasive throughout our lives. Imagine if a party is being hosted at the owner's apartment - the chance of intrusion would increase as guests arrive, and decrease as guests depart. When the owner is out on the town, as the night progresses they may be more and more likely to forget their device as they become more inebriated. This sort of continuous change in context is a difficult problem to solve, and this project aims to address this problem in some fashion.

Backpack - Device Loss Consider again the case where the device is in the backpack of the user. However, this time, let us imagine the user has a smartwatch, and sits down at a park bench.

Using the Bluetooth connection to the smartwatch, the device can roughly estimate the proximity of the device to the owner. If the owner were to get up and leave without the backpack, the connection would get weaker. When the device senses this, a notification on the smartwatch could let the owner know the backpack is being left behind, prompting them to retrieve their belongings.

However, this behaviour is also contextual. If the owner leaves their backpack in their apartment when heading out for a run, a notification is unnecessary. We can harness the same context functionality to disable the notification if the device is left in a trusted context.

3.1.3 Optimistic Context Detection

In general, improvements in user experience can be expressed in a single example, which we will name the "empty streets" scenario. By understanding the difference in reaction between various systems, one can gain a better understanding of how optimistic context detection can improvement upon existing works.

Empty Streets Consider an example in which the owner is walking down an empty street. There is no one around, and thus no devices are around. If we consider how conventional (pessimistic) works will react, they will often decide the device must be kept locked, due to a variety of reasons - not in a well known location, in public, no familiar people or devices nearby. This will continue to be the case whenever the user walks along this street, since many of the factors leading to it being called unsafe persist and cannot be ameliorated.

Conversely, consider an optimistic system. The absence of devices and noise suggests there are few people around, which tells us that the user is likely alone, accurately reflecting the situation. However, if there are a group of people walking along the street, the system can detect these people, and accurately reflect the situation. This approach provides a much better reflection of the reality of the situation, simply by making the default safe instead of unsafe.

3.1.4 Failing Examples

There are some cases where context information can be useless, harmful or just out of scope. In this thesis, there are some cases we do not aim to cover. However, due to the modularity of the system, it is entirely possible many of these cases may eventually be covered, as future contributions are made.

Snooping Partner In this scenario, an owner has been in a long-term relationship with their partner, and through some means (user input, past behaviour, etc.) the device is familiar with them and considers them safe around to be around. However, the owner has not given their partner permission to access the device.

Now, imagine the partner has some reason to believe the owner is cheating on them. They could take advantage of the trust the device has in them to open the device and read private information about the owner.

The issue here stems from an inability from the device to determine a sudden breakdown in trust between the owner and the partner. Not even the owner is aware of such a drastic change, and thus cannot account for it. The only solution is to not trust the partner at all, which is not feasible as the partner would likely spend a large amount of time in proximity with the device, often without any malicious intentions. On a more general level, PUPy cannot easily deal with interpersonal relationships or social context clues.

Coercion In the example of context given here, detecting the owner being coerced by a malicious actor is not helped by the context the device is in. Since coercion comes in many different shapes and sizes, detecting coerced behaviour is not something PUPy is built for. However, a module could be built that does detect it or provides methods for the owner to signal they are being coerced - but such functionality is not augmented by the use of PUPy.

Device Loss in Abnormally Empty Locations One context in which PUPy might break down in is locations that are traditionally crowded, but are currently empty for one reason or another. Since the device loss system will be enabled based mainly on the privacy on the situation, any context that appears to be a private context when, in reality, is generally a public context will cause a failure. For example, an empty coffee shop. If we are in a coffee shop and no one is detected, the device loss module may not engage, which is not the intended behaviour.

There are certainly workarounds - the device loss module could use location alongside the context information to further improve how it works, but this will not be built into the context system.

Chapter 4

Related Work

This chapter consists of an overview and discussion of a number of existing works and systems that have been devised to achieve similar goals as ours. We will discuss similarities and differences between these works and ours in order to get a sense of where our contributions sit among existing literature. We will begin with a discussion of existing implicit authentication techniques. Next will be an exploration of context detection frameworks and the contrasting approaches taken to context detection. The following section will discuss frameworks that have combined context detection and implicit authentication in similar manners to our work, and what conclusions and insights can be drawn from their approaches. Finally, we will discuss approaches to device identification.

4.1 Implicit Authentication

Implicit authentication forms an important use case for our system, and many existing works hold important insights and concepts for understanding context detection. In this section, we will look at a number of existing works in the field of implicit authentication, and discuss their relevance to PUPy.

Implicit authentication was first coined by Shi et al. [41], who investigate using user behaviour as a way to reduce the number of explicit authentications. They generate an authentication score by comparing a user's recent actions with their previous behaviour - behaviour matching previous behaviour (habitual behaviour) increases the score, while behaviour unlike past behaviour reduces the score. This approach is limited in that it only halts usage of the device by the adversary after repeated uses (minimum 2, maximum 16). Follow-up approaches would build upon the concept of implicit authentication, incorporating new advancements that improved performance.

One of the first fully realized implicit authentication frameworks to combine data from multiple sensors into a single framework was progressive authentication by Riva et al [38]. While prior works were mainly about individual implicit authentication schemes, Riva et al. looked at how one could combine multiple implicit authentication methods to provide a better user experience. Their framework used several methods, which included voice and face recognition and proximity to trusted devices in order to authenticate the user. These methods are used to calculate a confidence level that the current user is the owner of the device. Depending on that confidence level, certain functionality of the device would be permitted, while other functions would be limited pending explicit authentication. The basic concept of combining many inputs to get a better sense of the context the framework is after can be seen in many following works, be it in the realm of implicit authentication or context detection. In fact, this concept is central to PUPy, as discussed in Chapters 5 and 6. While we use it for context detection as opposed to authentication, many of the works we build upon can trace their approach back to Riva et al.

However, Riva et al. do not generally concern themselves with the wider context the device is in. For example, while their framework used device sensors to learn about the device and where it is (i.e. on a table, in a pocket, etc.) it did not seek to learn anything about the context it resides in - it did not attempt to find any other information about the context, such as crowding, macro location, etc. This led to the framework looking at context in a very narrow manner.

This is in contrast with our system, which aims to allow implicit authentication modules to adapt based on the wider device context, through detection of how many people are nearby, and what sort of relationship those people have with the device owner. This focus on learning about the context itself and not just the device's place in it allows for a far more dynamic approach to authentication and our other applications.

While not exactly implicit authentication, SnapApp by Buschek et al. [6] investigates ways of lowering the barrier to entry for users by using an optimistic approach to authentication - use a swipe lock on the lock screen to allow users to access the device for some short amount of time (e.g. 30 seconds). This swipe lock could be used up to a certain (user specified) limit, at which point the user must enter their PIN. This is an optimistic approach as it is assuming this will mainly help the owner. SnapApp does not respond to any of the main motivations we lay out for our system, since it is not an implicit authentication system nor does it rely on device context in any way. However this application does dovetail nicely with our approach, which aims to provide an optimistic approach to context detection. Combining the two applications could allow a far better approach to deciding when SnapApp can be used, without substantially impacting the goals of SnapApp.

In addition, both approaches can complement the use of implicit authentication alongside it. The approach taken in SnapApp can be permitted only when implicit authentication has a certain level of confidence in the user being the owner of the device. Similarly, our approach can allow an authentication module to decide if implicit authentication is sufficient given the context the device is in.

To conclude this section, we will look at various authentication schemes that have been proposed, that are either implicit authentication frameworks or closely related to it. While none of these approaches attempt to solve any of the issues our system aims to resolve, a general survey of these can provide an idea of what sort of methods can be used alongside our system. SilentSense by Bo et al. [5] and Touchalytics by Frank et al. [13] both investigate using user interaction (typing, swiping, gestures, etc.) with the device to authenticate the user. Such approaches show high accuracy, but have the drawback of only being applicable after the device is unlocked and is being used. Chauhan et al. [8] investigate to authenticate users based on aspects of the breathing, through the corresponding audio signature. While not implicit authentication, of note is Pico by Stajano [44], which investigates the use of external hardware tokens for authenticating users. This approach is useful due to the fact that accidental authentication is only possible in the case where the token is stolen, but requires the use of such a token. Similarly, Cola et al. [9] investigate using gait-based authentication via a wrist worn device (such as a smartwatch). Such an approach also requires an external device, but one that, unlike in Pico, has other uses. MULE by Studer et al. [45] investigated using location as a method on authentication, which many other implicit authentication schemes later investigated, such as Google's Smart Lock [14], which allows Android users to leave their devices unlocked in locations they know well, such as their home. While location plays an important part in how PUPy calculates the context values, it is not directly used for authentication. This approach of identifying contexts that are frequently visited is not a form of authentication in itself, and thus using location as a method of authentication is orthogonal to our work.

4.2 Context Detection

Context detection as a whole is an extremely wide field, not limited to mobile devices or standard sensors. In this section, we will look at a number of existing works in context detection that focus on context detection using standard mobile phone sensors, focusing mainly on the security of the context.

Rajput et al. [35] investigate using the device accelerometer and GPS to detect crowding of public transit by detecting if the user is able to find a seat. Their aim of detecting the crowdedness of a context is similar to the interest our system takes in the privacy of the context. Sadly, the nature of their approach means it cannot be generalized to other contexts, as it is geared specifically toward detecting crowding on public transportation by sensing if the owner of the device found a seat, or was forced to stand. It also requires dedicated server hardware and functionality added to the busses. This means that their approach could not be used directly in our work. Solutions such as this, however, reinforce how gathering privacy data from a large assortment of methods can improve overall estimates. While their approach was not integrated into our system, it could be adapted fairly easily alongside our devised approach, if their approach was more widely implemented. As mentioned, though, it would only be relevant when on public transit properly equipped for this approach.

Ramakrishnan et al. [36] aim to provide a context-based approach to locking the device, based on the use of a policy-driven framework in their PRISM framework. This focus on using policies to guide unlocking is similar to they approach taken in our system, where rules are used to govern the actions taken by listening applications when changes in context are taken. However, their approach puts these policies at its core. This makes extensibility impossible, as their policies, once set, are not added to and exposed only internally to the system. There is no method of extending these policies, either. In contrast, our approach uses rules to interpret the core values of the framework. It allows developers of functionality modules (authentication, device theft, etc.) to define the rules to fit the behaviour of the application, and does not rely on the rules being built into the framework, unlike in PRISM. Another similarity between PRISM and our system is that they aim mainly to enable or disable authentication based on context, instead of directly choosing what type of authentication to do. However, unlike PRISM, our approach allows our system to interface with implicit authentication methods, improving the overall user experience.

Gupta et al. [15] form an important part of the literature built off of in this work. Their approach of detecting nearby devices and developing a sense of familiarity with them is integral to the calculation of our context values, and therefore in determining context. Indeed, our work is so closely related, we build upon future work mentioned in their paper - that of marking unclassified contexts as safe, instead of unsafe. Due to the close nature of the two frameworks, a further comparison between PUPy and the system proposed by Gupta et al. will be discussed in Section 7.2.4, instead of here.

Another framework to build off the concepts developed by Gupta et al. is ConXSense, developed by Miettinen et al. [32] They have a similar approach in that they categorize contexts (location and WiFi based) into private, work or public contexts, either safe or unsafe. To evaluate the efficacy of their approach, they use a data collection app, into which subject input their own read of the context. The result from the framework's context profiler is then compared against the user's own feedback for the given context to obtain metrics of efficacy.

On a high level, ConXSense's aims to provide two use cases - a usable device lock, and resisting sensory malware. We will focus on the first use case, as the second is out of scope for our work. In order to meet this use case, ConXSense aims to modulate the use of a lock screen based on the context the device and owner find themselves in. It accomplishes this through collection of context data using device sensors, and through this data detecting the location and social context of the user. This, on its surface, is very similar to PUPy - however, it is how this data is used where the approaches sharply differ.

In order to make a decision about a given context, ConXSense extracts context features from the data, and feeds this data into a classifier, which decides whether protections can be relaxed. Crucially, this approach is seen as binary - either a context is *safe* or *unsafe*, without any grey areas. In contrast, PUPy allows some latitude in what sort of approach is taken. By exposing three values that simply report aspects of the current context, PUPy can allow for a more measured response to various contexts. This descriptive approach allows the context data collected through PUPy to be used in varying use cases. Instead of focusing only on safe versus unsafe, providing more information to the various functionality modules allows them to adapt their behaviour, providing more nuance in a given situation.

ConXSense shows some interest in considering the level of privacy exposure a context allows. Privacy exposure is used to delineate whether device usage in a given context is likely to contain information that the user wants to keep private. There are two types - private and confidential. These are defined as information about the user themselves, and information not directly related to the user. However, this is not further considered when classifying contexts - the classifier simply falls back on enabling or disabling the lock screen. In contrast, the approach taken in PUPy would allow applications that fall into these various exposure categories to react differently to the given context, allowing for a better user experience.

Overall, the approach taken by ConXSense does not allow the same level of granularity when dealing with contexts as PUPy. Instead of allowing for different reactions to varying contexts, the data collected and features extracted are used only to decide whether to relax protections, instead of allowing a more nuanced approach to the current context.

4.3 Implicit Authentication and Context Detection

While context detection and implicit authentication alone are interesting fields, combining the two can lead to extremely interesting results. While we do not fully investigate using various types of implicit authentication depending on context in this thesis, on of the primary goals of PUPy is to allow that sort of functionality.

Wójtowicz and Joachimiak [46] investigate combining context detection and biometric authentication in order to provide a more accurate authentication system. Since various approaches to authentication are effective only in a particular context (e.g., gait analysis), using the context the device is in to determine the most effective method of authentication allows for a better user experience.

To this end, their work describes a framework that takes in a wide range of information from device sensors, and uses that data to decide which authentication methods to exclude, allowing it to choose the most accurate method. Their work tends to focus on aspects of the device and user to extrapolate context information, instead of using sensors to directly learn about the context. As an example, having sound enabled or disabled leads them to believe the context is one where silence is necessary, as opposed to using the microphone to measure ambient noise. This user and device focused approach to context detection is contrasted by PUPy, which focuses on learning about the context by collecting information on the environment the device is in, rather than the settings of the device itself.

Mostly closely related functionally to our system is the CORMORANT framework devised by Hintze et al. [20] Indeed, the Android implementation of CORMORANT formed the basis on which our implementation was built. The modular approach used by Hintze et al. translates well to the theoretical approach we use in this work. However, while the application is structurally similar, the aims and results are very different.

Hintze et al. aim to combine three existing fields into a single authentication system - transparent biometrics, risk estimation and extension of the authentication scope (expanding authentication to be shared across many devices, instead of being limited to a single device). These first two goals closely mirror our own approach, though execution varies significantly. CORMORANT focuses on that third point of using cross-device authentication to achieve their goals, and focuses less on determining the context the device is in. They aim to provide continuous authentication across all devices by providing different methods of authentication across many devices, and sharing authentication results between them.

For example, they consider the scenario of a user with a laptop and a phone. Depending on the context, the type of authentication used varies. If the user is walking down the street, gait analysis is used. When the user is sitting at a table using a laptop, then keystroke authentication via the laptop is used. In both cases, these authentication methods are used for the unlocking of both devices, providing continuous authentication.

While we are also interested in different methods of authentication through our functionality modules, our focus is mainly in the area of context detection for authentication. Instead of authenticating the user, we aim to determine if authentication is necessary at all. Hintze et al. start to investigate this idea through their risk estimation plugin, but use extremely coarse statistics (such as the national crime rate or time of day) to determine risk. We take a far more advanced approach to risk estimation, using realtime data of the context to determine risk.

As mentioned, our Android implementation discussed in Chapter 6 is a heavily modified version of the CORMORANT Android framework, modified to better fit our needs. While the details of these differences will be discussed in Section 6.4, at a high level, three large changes were made. First, we removed the cross-device aspect of CORMORANT, and focused solely on the mobile device. Second, we made fundamental changes to the calculations done in the core of the CORMORANT system, adapting their confidence/risk values to the values outlined in Section 5.3. Finally, we built into the system a location-based context identification system, to track the current location-based context the owner is in.

4.4 Device Identification

An important part of PUPy is the method through which we decide what devices are familiar, and which are not. Our work builds upon that of Gupta et al. [15] in this respect, where Bluetooth addresses are used to identify and track devices. However, there has been an increasing push to make tracking devices across long periods of time in various locations more difficult. This poses an issue to our framework, both technically and ethically. While further discussion of these problems takes place in Section 6.1.1, some investigation has been done into identifying devices using various methods, and some of those works are discussed here.

There are a number of WiFi-based device identification approaches. Xu et al. [47] examine identification of wireless devices using potentially identifying variations across several layers of the network stack. For example, they note that miniscule imperfections in the device hardware can lead to variations in communications that can be used as an identifier for devices. However, this approach is not well suited to the sort of device identification we are interested in for our case, as some identifying features are only collected by the wireless AP. Similarly, Miettinen et al. [33] aim to classify IoT device types using an AP-based approach, in order to limit the damage compromised IoT devices can do to other uncompromised, non-IoT devices on the network.

Redondi et al. [37] aim to rectify the AP requirement by providing passive classification of network devices through packet sniffing. This approach does not require any information that is only available to the AP and device, but does require packet sniffing techniques, which cannot be done on Android devices without rooting the device. Yu et al. [48] also fingerprint devices using packet sniffing techniques, though only on packets they were the legitimate receivers of. Matte [31] provides a wide-ranging survey of fingerprinting attacks and countermeasures, and most approaches described are either far too intrusive to justify or have effective countermeasures. It also shines light on the ongoing push to randomize MAC addresses, making the easiest approaches to device identification more difficult.

A more promising avenue for mobile-based device identification are methods based on Bluetooth Low Energy (BLE) data. Several such approaches have been devised. Celosia et al. [7] take an approach of reading the Generic Attribute Profile, data widely available to any listening device, to fingerprint mobile devices. Similarly, Zuo et al. [49] look at IoT devices that advertise themselves over BLE, identifying devices vulnerable to a number of attacks. This approach is less directly applicable to our use case, as they are mostly concerned in identifying vulnerable devices, instead of all devices.

As we have seen, device identification remains an active area of research. However, existing approaches are either not ethically justifiable for our uses, or not applicable due either to the use of external hardware or take approaches that cannot be implemented on smartphones. For this work, we will take the approach of tracking Bluetooth addresses - when a device changes their address for privacy reasons, we will simply treat it as a new, unfamiliar device. This approach is not particularly satisfactory, but the focus of this work is not on device identification but a novel context detection framework. In testing, we use a dataset that was collected prior to the proliferation of such privacy-preserving approaches, allowing us to determine how the framework would preform assuming device identification was not an issue, or a satisfactory workaround was discovered.

4.5 Summary

In summary, we surveyed many existing works and categorized them into four groups - implicit authentication, context detection, a combination of the two, and finally device identification. While there exists a large body of existing works in these fields, and there are some works that address one or two motivations, none adequately account for all of the motivations simultaneously. However, we will build upon the works discussed here in order to address all motivations, combining new and existing approaches to address all of our concerns.

Chapter 5

System Design

In this chapter, we will discuss the theoretical design of our new context detection system. We will first outline the threat model we operate under, and some basic requirements for PUPy based on our goals. We then get into building the theoretical framework.

5.1 Adversary Model

In this section, we will present the adversary model under which our framework functions. There are three main variants, corresponding to each of the three functionality modules we outline. However, before discussing these variants, there are a number of similarities between the models we will enumerate first.

Generally, our adversary models do not aim to protect against adversaries that are aware of the system. This includes adversaries using strategies to confuse the system into reporting falsified results, such as tampering with Bluetooth, voice or similar signals sensed by the device. Some existing works investigate such attacks, showing they are possible and can often be counteracted [42, 24]. We consider this out of scope for our adversary models. We also assume there is no malware or other applications installed on the device that, accidentally or purposefully, impact the performance of the system, or provide falsified data. We assume that Bluetooth is always enabled (though this is specific to our implementation - it is quite possible other approaches that do not require Bluetooth could be integrated).

Authentication For authentication, the adversary takes the form of a person unknown to the owner (i.e. not a sibling, friend, etc.) that can physically access the device, but does not know about our framework. Their aim is to unlock the device and either interact with the device applications, access sensitive data, or otherwise tamper with the device. The adversary can be malicious, honest-but-curious or clueless, though we focus mainly on the malicious case. The goal of the authentication module is to refuse access to the adversary, while providing a convenient user experience for the owner. This adversary model will be the main model we use to evaluate the system in Chapter 7.

Theft For theft, the adversary takes the form of a person unknown to the owner who can physically access the device. The adversary is malicious, aiming to remove the device from the care of the owner permanently (that is, to steal it). They are unconcerned with accessing the device data. The goal of the system is to either alert the owner to a theft that is in progress, or stop the theft (via an alarm or other deterrent).

Loss For loss, there is no adversary. At least, there is no adversary other than the owner's own memory. The goal of the system in this approach is to alert the owner to potential loss of their device.

With these adversary models outlined, we will now outline some requirements for PUPy.

5.2 Design Principles

Using the motivating examples from Chapter 3, we can begin to specify some desired features of our application.

Less Focus on Authentication Since this will not be used to authenticate users, we do not have to worry about whether data can be used to authenticate a user. This will widen the types of data we can use, which makes PUPy more useful in more cases.

Strong Focus on Contextual Awareness There should be a focus on using the context of the device and its relationship to the owner at a given time. This includes adapting the current context based on both user actions and changes in the environment.

Numerous Data Sources There should be functionality permitting the combination of many different data sources to determine the context of the device, and its impact on how the device should react. These data sources could be of varying reliability (data may vary from very accurate to wildly inaccurate) and availability (a source may only be applicable in specific scenarios e.g. gait analysis).

Extensible API An API that can easily be built on is crucial for allowing a wider application of the system, and allowing for a wide number of data sources. In particular, it should be easy to allow applications to read and react to changes in the context.

5.3 Theoretical Framework

This section will outline what aspects of the device context we will track in PUPy, and how we track these aspects based on input module data. We introduce the three values that track our chosen aspects. Finally, we will outline the equations for calculating these values, and how they interact.

We will use C to represent a particular real-world context, and C_n to represent the nth occurrence of that real-world context. The context will consist of a feature vector, where each value is a value between 0 and 1, representing some part of the context. This feature vector will consist of three values, as described next.

Privacy The measure of how private the context is, based on the number of people detected. In this case, by privacy we refer solely to how crowded the context is. A value of 0 is very public (very crowded), while 1 is very private (not crowded). We denote the privacy of C_n by $\mathcal{P}(C_n)$.

Unfamiliarity The measure of how unfamiliar the context is, based on the number of unfamiliar people detected. A value of 0 is very familiar, while 1 is unfamiliar. We denote the unfamiliarity of C_n by $\mathcal{U}(C_n)$.

Proximity The measure of how close the device is to the owner. A value of 0 is being very far, while 1 is very close. We denote the proximity at C_n by $\mathcal{D}(C_n)$.

The use of these three values, and the functionality they allow for, fulfills our first and second requirements. These values are widely applicable instead of being specific to authentication, and provide crucial contextual information to functionality modules. There will be three stages to the calculation and usage of these values, with each stage being completed by a particular type of module. The first step in this process is gathering information about the context the device is in, and providing some information about a particular part of the context. This step will be completed by Input Modules.

Input modules are modules built into the system by default, or contributed by external developers. An example of an input module is a module that uses Bluetooth scans to estimate the number of people in the context, and uses that estimate to provide a value for use when calculating $\mathcal{P}(C_n)$. This approach of using input modules fulfills our third requirement, allowing us to take inputs from numerous data sources.

The second step will be the aggregation step, which will form the majority of this section. This step will be completed by the context engine. The context engine will take the estimates provided by the range of input modules installed and aggregate them into a small number of values that can be provided to modules in the final stage to adapt their behaviour, these modules are called functionality modules.

Functionality modules will take the values as calculated by the context engine and use them in some way. In the case of authentication, this may result in enabling or disabling explicit authentication based on the context, or changing the type of authentication required to use the device.

5.3.1 Privacy

We will now examine the procedure for calculating privacy based on input module data. The set P will denote the set of values returned by the input modules reflecting the number of people in the current context.

Each $p_i(C_x) \in P$ is an estimate of the number of people nearby, which is weighted based on the confidence the module has in its estimate (in order to account for accuracy, etc.), denoted by $w_i \in W_{\mathcal{P}}$. This gives us the following equation for the number of people, estimated across all applicable input modules:

$$L(C_n) = \sum_{i \in 1 \ldots |P|} \frac{p_i(C_n) w_i}{|P| \sum W_{\mathcal{P}}} \tag{5.1}$$

We then convert this unbounded number of people into the instantaneous privacy value, $\mathcal{P}(C_n)$:

$$\mathcal{P}(C_n) = 1 - \frac{\alpha_{\mathcal{C}}^{1 - \frac{1}{L(C_n)}}}{\alpha_{\mathcal{C}}} \tag{5.2}$$

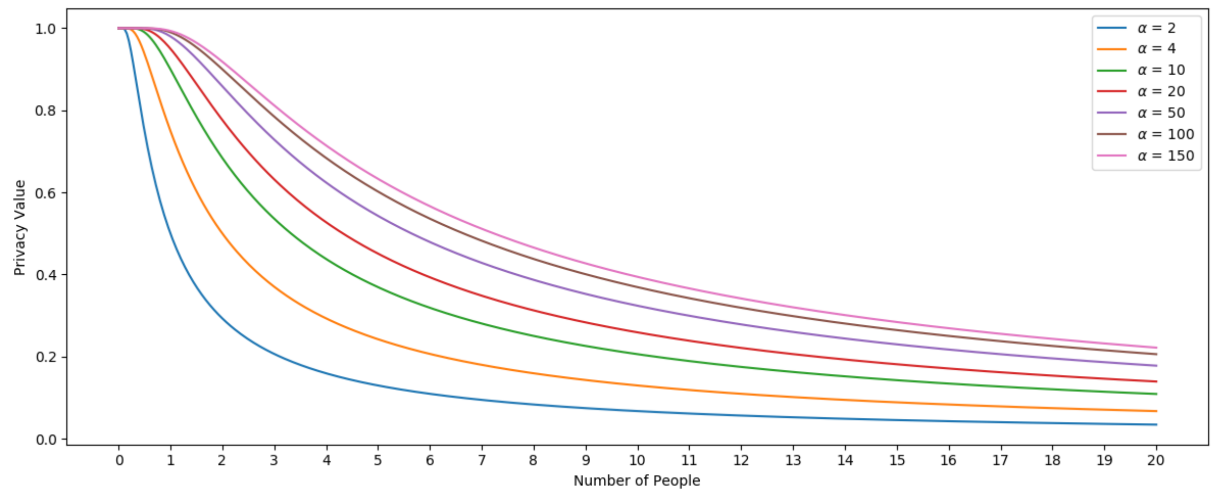


Figure 5.1: Examples illustrating the effect of different values of $a_{\mathcal{C}}$ on the decay of the value for $\mathcal{P}(C_n)$.

$\alpha_{\mathcal{C}}$ controls how quickly the value decays - if $\alpha_{\mathcal{C}}$ is small, the value falls below .75 if more than one non-owner interaction is possible, falling to .25 by 10 interactions. The effects on values can be seen in Figure 5.1. $\alpha_{\mathcal{C}}$ can be set to different values in different locations, allowing us to use it to express different levels of trust in different areas. $\alpha_{\mathcal{C}}$ is discussed more thoroughly in Section 5.3.4. $W_{\mathcal{P}}$ is currently not used (i.e., all modules have $w_i = 1$) in our implementation or evaluation, due to either relying on a single module, or having similar accuracy across modules.

This formulation means that $\mathcal{P}(C_n)$ starts at 1 when the device owner is alone, and decays as the number of people increases. This fact is part of what makes the system optimistic - as long as the input modules cannot detect anyone nearby, we assume the owner is alone, rather than that we are in an unsafe context.

5.3.2 Unfamiliarity

We will now examine the procedure for calculating the unfamiliarity of a context. In order to calculate this, we will need to use a measure of *familiarity*. For this, we will build upon the work of Gupta et al., using their method of instantaneous familiarity [15].

We first define device familiarity, as follows:

$$F_d(d, C_n) = \alpha_F * occ(d, C_n) + (1 - \alpha_F) * F_d(d, C_{n-1}) \tag{5.3}$$

where

$$occ(d, C_n) = \begin{cases} 1 & \text{if } d \text{ is observed in } C_n \\ 0 & \text{if } d \text{ is not observed in } C_n\text{, and} \\ & (n - N_{last}) < N_0 \\ F_d(d, C_{n-1}) & \text{otherwise} \end{cases}$$

where N_0 controls how many observations pass before they are disregarded, and N_{last} (defaulting to 0) is the ordinal number representing the last sample of C in which d was seen. That is, if d was last seen in C_{12} (the 12th time the device has been in this context), then $N_{last} = 12$. α_F is a suitably chosen constant, controlling how quickly the system learns. In our implementation and evaluation, a value of $\alpha_F = .05$ is used. The structure of this method ensures that $F_d(d, C_n)$ is a value between 0 and 1.

Using this definition of device familiarity, we go on to define instantaneous familiarity:

$$\mathcal{F}(C_n) = \frac{1}{|D_{C_n}|} \sum_{d \in D_{C_n}} F_d(d, C_n) \tag{5.4}$$

where D_{C_n} is the set of devices in the context C_n. This defines instantaneous familiarity as the average familiarity of all devices in C_n. Since this is the average of values between 0 and 1, the overall average will be between 0 and 1.

We will now build upon the existing theoretical work we have described so far, and expand on this concept in a novel method that converts this previously pessimistic measure to an optimistic approach. To accomplish this, we calculate the novel value we name device unfamiliarity, denoted as $\mathcal{U}(C_n)$. First, we define the following equation:

$$U(C_n) = L(C_n) * (1 - \mathcal{F}(C_n)) \tag{5.5}$$

This equation takes the number of people detected ($L(C_n)$, from Equation 5.1), and multiplies this by the inverse of the instantaneous familiarity of the context (since $\mathcal{F}(C_n) \in [0, 1]$). This gives us an estimate for the number of unfamiliar people nearby. We then use the similar conversion as in Equation 5.2 to convert this unbounded number to a value between 0 and 1:

$$\mathcal{U}(C_n) = \frac{\alpha_{\mathcal{C}}^{1 - \frac{1}{U(C_n)}}}{\alpha_{\mathcal{C}}} \tag{5.6}$$

thus giving us our final equation for calculating unfamiliarity. As mentioned, we can modify $\alpha_{\mathcal{C}}$ to control the rate at which $\mathcal{U}(C_n)$ increases. The effects on values can be seen in Figure 5.2. $\alpha_{\mathcal{C}}$ can be set to different values in different locations, allowing us to use

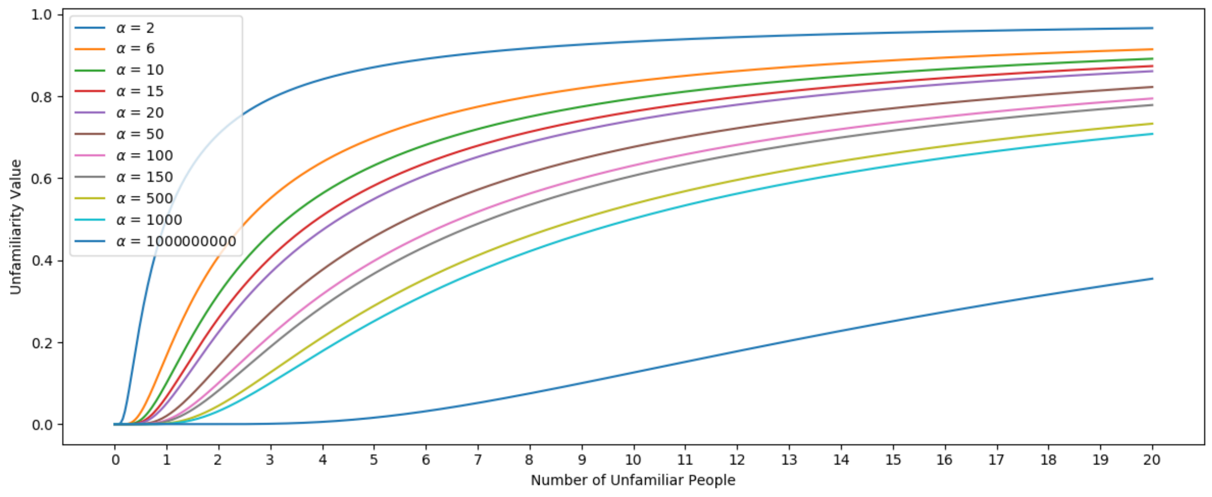


Figure 5.2: Examples illustrating the effect of different values of $a_{\mathcal{C}}$ on the growth of the value for $\mathcal{U}(C_n)$.

it to express different levels of trust in different areas. $\alpha_{\mathcal{C}}$ is discussed more thoroughly in Section 5.3.4.

In a similar manner to $\mathcal{P}(C_n)$, $\mathcal{U}(C_n)$ is a value between 0 and 1. However, unlike $\mathcal{P}(C_n)$, $\mathcal{U}(C_n)$ starts at 0 when the device owner is alone, and grows as the number of unfamiliar people increases. So unlike $\mathcal{P}(C_n)$, unfamiliarity starts at 0 and increases.

The reason for this discrepancy is that changes in privacy and unfamiliarity mean different things. As privacy changes from its default, we move from *high* privacy (i.e. a very private context) to *low* privacy. In contrast, as unfamiliarity changes, we move from *low* unfamiliarity (there are no unfamiliar people around) to *high* unfamiliarity. So it does not really make sense for these values to both decrease when they mean opposite things.

5.3.3 Proximity

We will now examine the procedure for calculating the strength of the relationship between the owner and the device based on input module data. Each $r_i(C_x) \in R$ is an estimate of the distance, which is weighted based on the confidence the module has in its estimate (in order to account for accuracy, etc.), denoted by $w_i \in W_{\mathcal{R}}$. This gives us the following

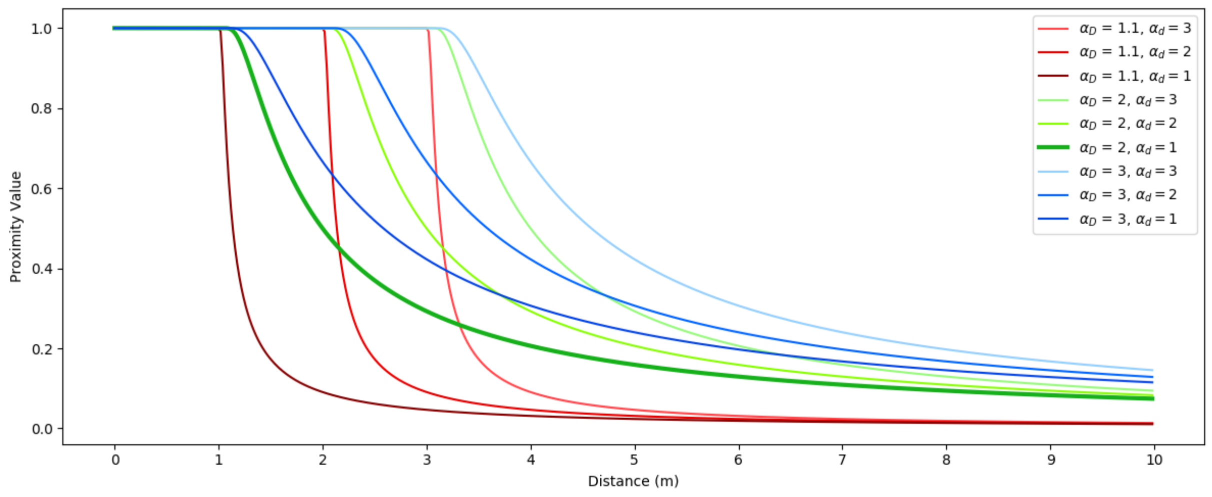


Figure 5.3: Examples illustrating the effect of different values of $a_{\mathcal{D}}$ and a_d on the decay of the value for $\mathcal{D}(C_n)$. The highlighted line plots the values selected for $a_{\mathcal{D}}$ and a_d in our Implementation.

equation for the distance, estimated across all applicable input modules:

$$d(C_n) = \sum_{i \in 1 \ldots |R|} \frac{r_i(C_n) w_i}{|R| \sum W_{\mathcal{R}}} \tag{5.7}$$

We then convert this unbounded distance value into proximity, $\mathcal{D}(C_n)$:

$$\mathcal{D}(C_n) = \begin{cases} 1 & d(C_n) \leq \alpha_d \\ 1 - \frac{\alpha_{\mathcal{D}}^{1 - \frac{1}{d(C_n) - \alpha_d}}}{\alpha_{\mathcal{D}}} & d(C_n) > \alpha_d \end{cases} \tag{5.8}$$

This allows the relationship to be strong while the device is within α_d of the owner, and we can use $\alpha_{\mathcal{D}}$ to decide how quickly the relationship decays as distance increases. The effects on values can be seen in Figure 5.3. Unlike the previous two, we do not want proximity to change in different contexts, so these two values will be set based on experimentation and will not change after being set. $W_{\mathcal{R}}$ is currently not used (i.e., all modules have $w_i = 1$) in our implementation or evaluation, due to either relying on a single module, or having similar accuracy across modules.

Similarly to $\mathcal{P}(C_n)$, $\mathcal{D}(C_n)$ starts at 1 when the device is close to the owner, and decays as distance increases. This is due to the same logic that governs the changes in $\mathcal{P}(C_n)$. As privacy changes from its default, we move from *high* privacy to *low* privacy. In the case of proximity, we move from *high* proximity to *low* proximity.

5.3.4 Context Familiarity

Both of the equations for privacy and unfamiliarity have a value governing how quickly the value changes based on changes to the environment, denoted as $\alpha_{\mathcal{C}}$. We call this value the *Context Familiarity* value, which will track the familiarity of the context we are in rather than the people or devices in it. The context will currently be limited to location, but can easily be expanded to other contexts through functionality modules with a small amount of work.

There is the potential to use this value as a way to incorporate functionality similar to that of the familiarity system into the context engine. There are a few attributes context familiarity should exhibit:

1. Slow growth
2. Trivial to calculate
3. High value in familiar contexts, low value in unfamiliar contexts

We use device sensor data we collect to change context familiarity. This allows context familiarity to be higher in more familiar areas. This causes privacy to decay slower, and unfamiliarity to grow slower. This approach allows us to build in some tolerance for a certain number of unfamiliar devices in historically familiar locations, such as your home and apartment, or a frequent coffee shop.

In order to gather historical data on a context, it is necessary to track instances of this context. Using the number of times a given context has been detected, we can change context familiarity to be more forgiving. For example, the first time a context is entered, context familiarity defaults to 3. From there, every subsequent detection of this context causes the value for context familiarity to increase by some amount (potentially .1, or 1). As the user is in that context more and more frequently, context familiarity grows larger, causing the device to feel more "comfortable" in that context.

Certain contexts will never be safe, and so we want to stop context familiarity from increasing. This could potentially be handled through user interaction, or perhaps using only static contexts by disabling learning and setting a specific value for each context that

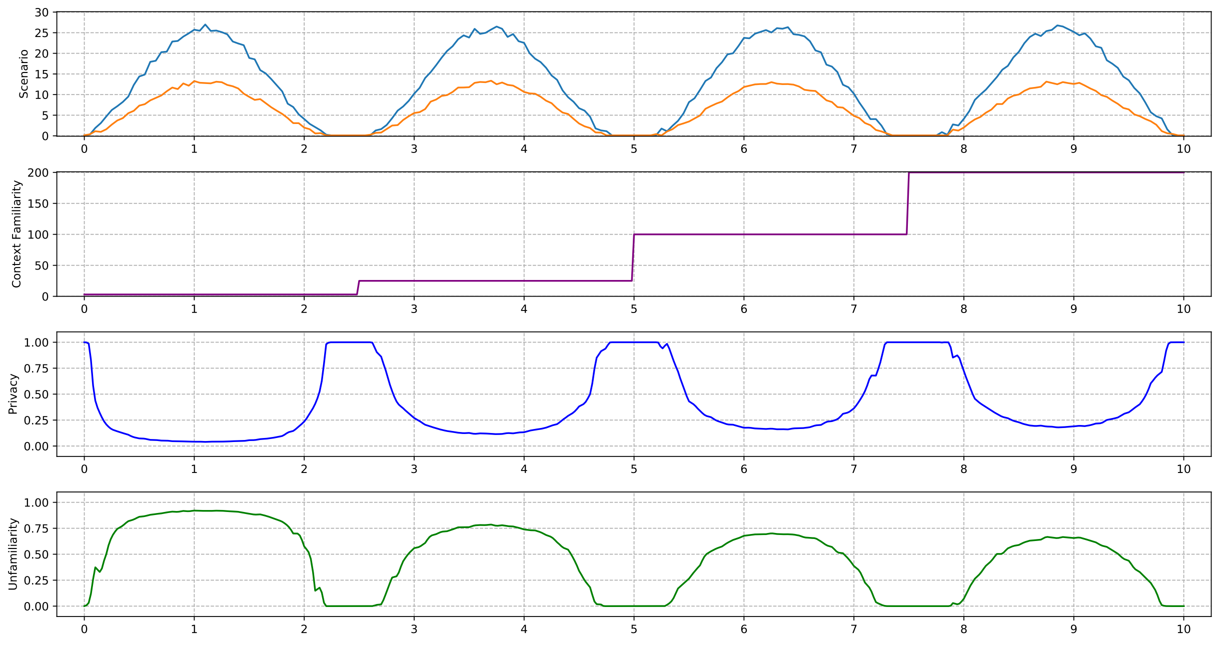


Figure 5.4: A synthetic example, showing how changing $a_{\mathcal{C}}$ changes the resulting values for privacy and unfamiliarity. Light blue denotes the number of people overall $(L(C_n))$, orange denotes the number of unfamiliar people $U(C_n)$. Purple denotes the context familiarity $(\alpha_{\mathcal{C}})$ for values $\alpha_{\mathcal{C}} = 3, 25, 100, 200$. Dark blue denotes the privacy value $(\mathcal{P}(C_n))$, and green denotes unfamiliarity $(\mathcal{U}(C_n))$.

does not change. Barring this, the value of context familiarity for the context C_n could be calculated as:

$$\alpha_{\mathcal{C}} = 2 + n \tag{5.9}$$

$\alpha_{\mathcal{C}}$ has pathological behaviour when set to 1, and below 3 decays too rapidly for realistic scenarios. Therefore, $\alpha_{\mathcal{C}}$ has a default value of 3 for all contexts. As contexts are encountered more and more frequently, this value will continue to increase - however, we cannot increase this value forever, since sufficiently large values of $\alpha_{\mathcal{C}}$ completely disable the system (as can be seen in Figures 5.1 and 5.2, when $\alpha_{\mathcal{C}} = 1\text{e}{+}9$), which is undesirable. In order to maintain functionality in frequently visited contexts, $\alpha_{\mathcal{C}}$ will be capped at 200.

For an idea of how increasing values for context familiarity changes the behaviour of $\mathcal{P}(C_n)$ and $\mathcal{U}(C_n)$, we look to Figure 5.4. Depending on the different values for $\alpha_{\mathcal{C}}$, the decay of $\mathcal{P}(C_n)$ and growth $\mathcal{U}(C_n)$ changes - the larger the value for $\alpha_{\mathcal{C}}$, the slower the values change.

Some user interaction can also be built into the system. For example, we could allow the user to set their home. When the user sets their home, we will set α_C for the corresponding context to the maximum - it is assumed the home is always frequently visited. This could be implemented through an input module that reports high alpha when the user is at their home, and is massively weighted by the context engine. This has not been implemented, as the framework seems to adapt quickly without the need of user interaction.

5.3.5 Interfacing with Functionality Modules

In order for these values to actually interface with functionality modules, we will use a system of rules provided by the installed functionality modules to decide when these modules should be notified. If the functionality module wants to allow the user to tweak these rules, that is entirely up to them, but is not necessary. This allows the context engine to function entirely without user interaction if need be. For the purposes of evaluation and experimentation, three functionality modules have been implemented as outlined in Section 6. The rules governing them are outlined in Tables 6.2, 6.3 and 6.4.

5.4 Justification

The aim of this section is to justify why each of these three values are necessary, and why their combination is sufficient to support a variety or novel scenarios not covered by previous work. To that end, we will discuss the importance of each value, and how combinations of these values lead to a more effective approach than any of the values alone can provide.

5.4.1 Privacy

Privacy allows modules to adapt their behaviour based on the privacy of a given context. Due to the simplicity of calculating the value (how many devices are nearby?), this value will likely frequently be one of the more accurate. By itself, it can be used by modules to respond to contexts where the owner is in a crowded location, or those where the owner is alone. A potential use case for this is device loss, where we want to notify the user if they are leaving their device behind. In this case, we make the assumption that in private contexts, the owner can safely leave their device behind, and therefore this choice may be intentional. Conversely, in public contexts, it is unlikely the owner intentionally leaves the device behind.

In addition to the standalone case, the combination of privacy and unfamiliarity can provide helpful insights into the current context, which is covered in Section 5.4.2. In general, using the values together vastly improves usefulness and the amount of information one can gather about the context.

5.4.2 Unfamiliarity

Unfamiliarity allows functionality modules to adapt their behaviour based on the number of unfamiliar devices nearby. The clearest use case is when determining how threatening the current context is. It is a fair assumption that a large number of unfamiliar devices nearby is a more threatening context than one without any unfamiliar devices.

This translates to helping us enable and disable functionality modules based on the threat level of the context. In more threatening contexts - say, a crowded coffee shop - authentication methods demanding a stronger level of certainty can be enabled, and a device theft module can be prepared for potential theft.

Unfamiliarity can be made more useful through the use of privacy alongside it. For example, imagine two scenarios. In both, unfamiliarity is hovering around .25. In the first scenario, privacy is relatively high, at .75, while in the second scenario, it is very low.

In the first scenario, the context is significantly more private, implying that it is likely the owner is surrounded with a few unfamiliar people - perhaps they are meeting for the first time. Since there are only a few people around, perhaps we can continue to use weaker forms of authentication, despite there being some unfamiliar devices nearby. It is fairly easy for the owner to keep track of the few unfamiliar people nearby, without many other people to track.

In the second scenario, the owner is in a public setting. But the low value of unfamiliarity implies that most of the people in the context are familiar - perhaps this is a workplace. In this case, the combination of some small number of unfamiliar devices with a fairly public setting may prompt the use of stronger authentication methods, or activation of a device theft module.

5.4.3 Proximity

The use case for this value is fairly obvious - giving the various functionality modules access to the proximity of the device allows them to adapt their behaviour based on this value. This is most useful for device theft and device loss, since both by definition require the device to be a certain distance from the owner, but can also be used for authentication - methods such as voice recognition can function from a fair distance, but we would only want them functional in close proximity to the user.

Combining proximity with privacy and unfamiliarity allows us to get a sense of whether it is a problem when proximity decreases. By itself, proximity cannot distinguish between the case where the owner leaves their device behind in their home, and in a crowded coffee shop. By using privacy and unfamiliarity, such a difference can be determined. Additionally, by comparing privacy and unfamiliarity, we can determine if decreasing proximity between the owner and device is more likely a case of theft, or loss. In a context with low unfamiliarity, regardless of privacy, it is more likely device loss. However, if unfamiliarity is high, theft becomes more likely. Distinguishing behind these approaches allows us to react differently to each case.

Chapter 6

Implementation

In this chapter, we will describe the Android implementation of PUPy, including the implemented structure and the specific purpose of each module. Figure 6.1 shows a general outline of the structure of the system. There are four main parts - the hardware sensors and OS that handles interfacing with the hardware, the input modules that produce estimates for any combination of privacy, unfamiliarity and proximity based on that sensor data, the context engine which aggregates the estimates and tracks the familiarity of a given context, and the functionality modules, that actually act upon the resulting values.

The overall structure of the system leans heavily on the work completed by Hintze et al. in CORMORANT [20, 21], a cross-device authentication system that is built upon a modular system well suited to our needs. The source code for CORMORANT formed the starting point for this implementation. In the course of implementing PUPy, many parts of the framework were fundamentally rewritten, or just removed wholesale. Despite this, many similarities remain.

CORMORANT was built with robust compatibility for plugins using a fairly simple service interface, allowing the structure to be adapted easily with limited modifications necessary. It also has facilities for querying the backend for the current confidence of the user's identity based on the active authentication plugins, which was adapted into the rules module, used as an interface between the context engine and functionality modules.

In Section 6.1, we will outline the currently implemented input modules, including what data they use and which values they provide estimates for. In Section 6.2, we will examine the core of the system, the context engine, that implements most of the theoretical underpinnings of the system previously discussed. In Section 6.3, we will discuss the

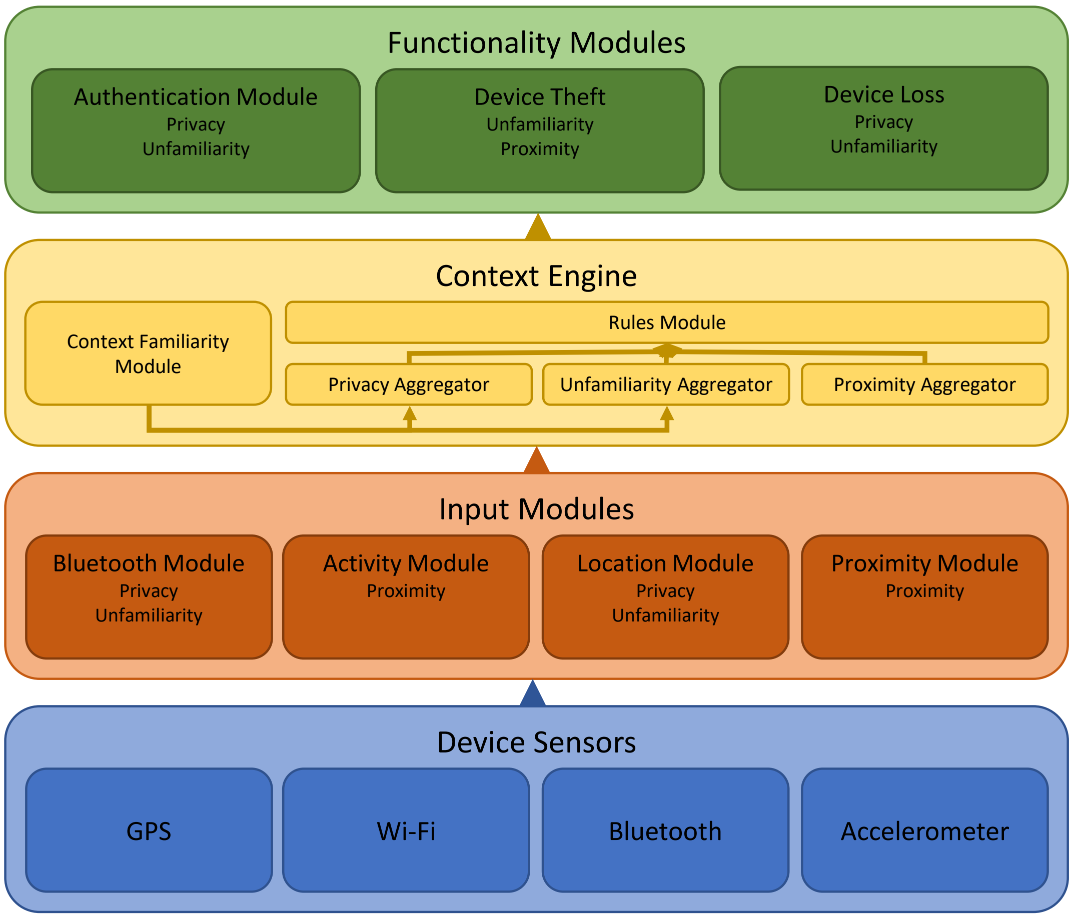


Figure 6.1: System diagram of PUPy.

current implementation of the functionality modules, and due to their fairly basic existing implementations, the resulting envisioned functionality.

6.1 Input Modules

The first step of the system is to collect sensor data from the device, and use it to calculate an estimate for one or more of the values. We will look at the implemented modules - how they work, and what estimates they provide to the context engine.

An important similarity all input modules share is the use of a persistent notification. This notification is required to maintain access to device sensors when an application is in the background on the latest versions of Android. These notifications can be minimized or hidden by the user, but by default this leads to a large number of persistent notifications in the device's notification bar.

6.1.1 Bluetooth Application

The Bluetooth application is the main module for estimating privacy and unfamiliarity. It conducts periodic Bluetooth scans, using this information to generate an estimate for $L(C_n)$ (estimate of the total number of people in the context) and $U(C_n)$ (estimate of the total number of *unfamiliar* people in the context) from Equations 5.1 and 5.5 from Section 5.3. This means there are two main purposes of the application:

Estimating the Number of People When reporting a value for $L(C_n)$, that value is obtained by counting the number of nearby Bluetooth devices, filtering out Bluetooth devices not tied to a particular user (mainly IoT devices). This approach means that users can be counted more than once, if they have multiple devices on their person (e.g. a phone and smartwatch). We also make the assumption that every person carries at least one device we can detect.

Estimating the Number of Unfamiliar People In order to track the number of unfamiliar people ($U(C_n)$), we must calculate $\mathcal{F}(C_n)$ from Equation 5.4. We do this using the same basic approach as Gupta et al. [15], by storing previous scans, and calculating a device familiarity value for each device seen. The average of this gives us $\mathcal{F}(C_n)$, from which we can calculate $U(C_n)$ as in Equation 5.5.

Implementation Details

The application registers two services with the context engine: `PrivacyPluginService` and `UnfamiliarityPluginService`. They rely on using the `BluetoothAdapter` built into Android to interface with the Bluetooth application. They register a `BroadcastReceiver` to receive `BluetoothDevice.ACTION_FOUND` broadcasts, allowing them to listen for when Bluetooth devices are detected. When an address is discovered, they each take different actions:

Privacy The privacy service maintains a list in memory of recently seen Bluetooth devices, and reports this value to the context engine as an estimate of the number of people around when polled by the context engine via the `MSG_POLL_DATA` message, using the `Message` class sent to the `Handler` implementation used by the context engine's plugin manager.

Unfamiliarity The unfamiliarity system is obviously a bit more complex, as it must track devices over a much longer period of time, and calculate the number of unfamiliar devices as outlined in Section 5.3.2. This requires also keeping track of recent devices, functionality that is shared with the privacy service. To store the familiarity value for devices that have not detected recently, a basic SQL database is used, using the built-in Android database libraries. The database simply tracks all sightings of all Bluetooth devices, and this data is used to compute the device familiarity for the current context. Combining this data allows us to obtain an estimate of the number of unfamiliar people nearby, which is published when polled by the context engine via the `MSG_POLL_DATA` message.

Device Identification

Device identification forms a significant challenge to deciding what devices to trust - if one does not know if they have seen the device before, they must assume they are strangers. Due to growing interests in privacy among device users, manufacturers are more frequently implementing measures that make tracking devices across time and space more difficult.

This poses a significant ethical question for our framework - given that these devices do not want to be tracked, should we be tracking them? While we do not use the identification information for any nefarious purposes, the question remains. For now we have decided to leave this an open question, focusing instead on testing the framework assuming devices can be tracked.

Activity	**Distance Estimate**
Walking	0 metres
Running	0 metres
On Foot	0 metres
On Bicycle	0 metres
In Vehicle	2 metres
Still (Not on person)	5 metres

Table 6.1: The mapping for activity to distance estimate used by the activity application.

In the meantime, we simply keep track of Bluetooth addresses without any attempt to counter changes in these addresses. So, this would mean that when the MAC Address resets, we would lose all familiarity with the device. Under our current implementation with a learning rate of 0.05, it takes 200 observations to obtain a familiarity of roughly .85. Using 10 minute intervals, that is approximately 33 hours of time spent with a device.

It is important to note that the sort of privacy-preserving behaviour we see is not uniform across all manufacturers, and exact ramifications will vary on the devices encountered. Additionally, it is possible other methods of identifying people can be used or will be developed, such as voice recognition.

6.1.2 Activity Application

The activity application is the primary method of estimating proximity. It relies mainly on accelerometer data and Google's `ActivityRecognitionClient`. Depending on the type of activity the user is engaged in, a different value is reported to the context engine. In this case, the estimate is the value for $d(C_n)$ (estimate of the distance in metres between the owner and device) in Equation 5.7.

In Table 6.1, we show the mapping between the current activity the user is engaging in and the corresponding distance estimate. This is obviously a fairly inaccurate manner of estimating distance, but the point of the application was not to provide an extremely accurate proximity estimation, but an estimation to test with.

6.1.3 Location Application

The location application is similar to the Bluetooth application, in that it also estimates $L(C_n)$ and $U(C_n)$, and thus shares the same two purposes. However, it obtains that estimate another way. The location application keeps track of specific locations the user visits frequently, and reports a higher value for $L(C_n)$ and $U(C_n)$ when not in proximity to those specified locations.

The location application works in two steps - firstly, the user configures their desired locations through the `LocationConfigurationActivity` shown in Figure 6.2, which as the name implies, allows the user to set and remove locations they deem sufficiently safe. The activity uses a Google map to display the safe locations, and allows the user to select on the map where to place safe locations.

Using the locations dictated through the `LocationConfigurationActivity`, the application uses the `LocationManager` to get the last known location of the user, preferring GPS location over network estimates. The application then calculates the distance between all defined safe locations and the user's current location - it finds the minimum distance and calculates the estimates $p_i(C_N)$ for $\mathcal{P}(C_n)$ and $u_i(C_N)$ for $\mathcal{U}(C_n)$ via the following equations:

$$p_i(C_n) = \text{distance (m)}/50$$

$$u_i(C_n) = \text{distance (m)}/75$$

These values are then reported back to the context engine whenever the engine requests data (through a `MSG_POLL_DATA` message).

6.1.4 Proximity Application

The final input module is another proximity application, which is built around the device's proximity sensor. It uses the proximity sensor to detect if the device is currently in the user's pocket, and reports a low distance (0 metres) if it is. If it is not, it reports a higher distance (5 metres), since it is likely not on person. It reports one of these two values as $d(C_n)$ to the context engine.

6.2 Context Engine

The context engine is what will combine the estimates obtained from the input modules, using the processes outlined in Section 5.3 . It then provides the aggregated values to the

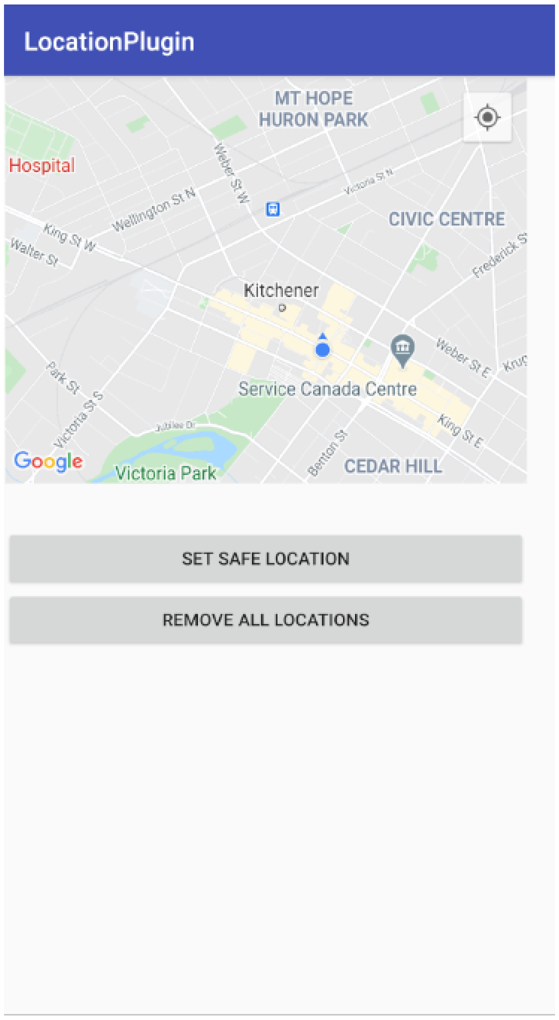


Figure 6.2: Screenshot of the `LocationConfigurationActivity`.

functionality modules. The context engine forms the core of PUPy. It is what aggregates estimates, calculates the context values, and compares these values to rules set by the functionality modules. Alongside the main modules located in Figure 6.1, there is a fair amount of supporting code as well. Overall, the structure of the code can be broken down as follows:

1. User interface
2. Plugin manager
3. Aggregator modules
4. Context familiarity module
5. Rule module

In order to give context for when the higher level modules are discussed, we will first discuss how the supporting code is implemented. We will then proceed to review the implementation of the higher level modules.

User Interface

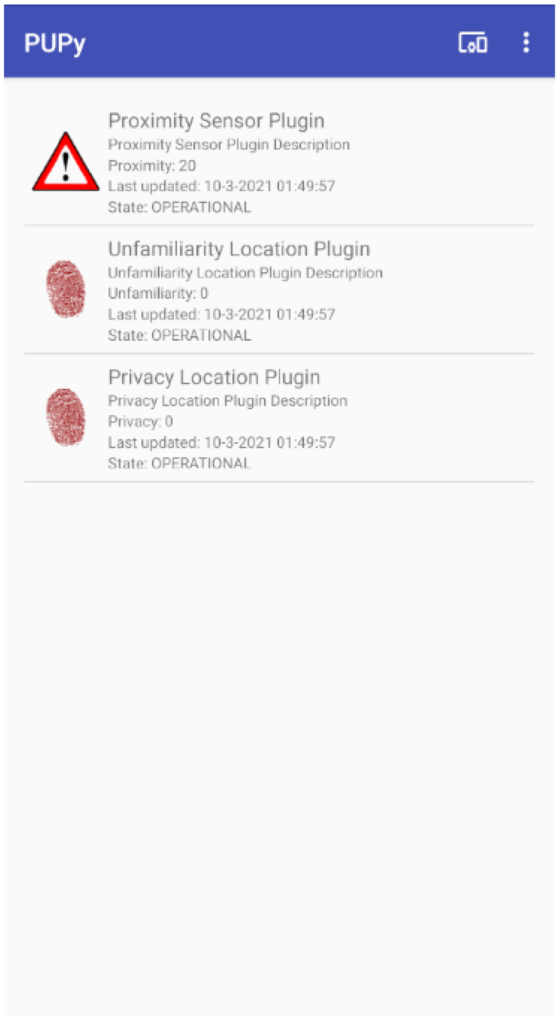
The user interface, as shown in Figure 6.3 is fairly simple - it is mainly used to list active plugins, and give the user a way of seeing the estimate provided by each module. None of the modules interact with the user interface - instead, the data presented is obtained from the plugin manager, which in turn receives it directly from the input modules.

The user interface also allows the user to access the configuration activities (if they exist) for these input modules. If the user taps on any of the entries for the input modules, and these input modules have defined a configuration activity, that activity will be launched, allowing the user to tweak user preferences through the application.

The user interface is also the method through which the context engine asks for its required permissions - namely, `ACCESS_FINE_LOCATION` and `ACCESS_COARSE_LOCATION`.

Plugin Manager

The plugin manager is the part of the context engine through which the engine communicates with the input and functionality modules. To that end, it handles all inter-app communication and the adding and removing of active input/functionality modules. The core



Proximity Sensor Plugin
Proximity Sensor Plugin Description
Proximity: 20
Last updated: 10-3-2021 01:49:57
State: OPERATIONAL

Unfamiliarity Location Plugin
Unfamiliarity Location Plugin Description
Unfamiliarity: 0
Last updated: 10-3-2021 01:49:57
State: OPERATIONAL

Privacy Location Plugin
Privacy Location Plugin Description
Privacy: 0
Last updated: 10-3-2021 01:49:57
State: OPERATIONAL

Figure 6.3: Example of the context engine's user interface, using three placeholder input modules.

of this inter-app communication is implemented using the `Message` and `Handler` classes provided by Android. The `Handler` listens for four specific types of messages:

1. `MSG_ADD_PLUGIN`: Register new input and functionality modules
2. `MSG_PRIVACY`: Handle privacy data updates
3. `MSG_UNFAMILIARITY`: Handle unfamiliarity data updates
4. `MSG_PROXIMITY`: Handle proximity data updates

These messages are parsed and processed by the plugin manager. When a new module is registered, it provides to the plugin manager information that allows the context engine to interface with the new module. This data, which is stored in a `PluginInfo` object, includes (but is not limited to) the type of plugin (privacy, unfamiliarity, or proximity) along with the title and description of the module to show in the module list. It also initializes a default value for the estimate of the value, which will later be updated through a data update. It is important to note that there may be multiple entries in the plugin manager for each input module installed, if the module reports estimates for multiple values.

In the case of a data update, the value corresponding to the type of the plugin is updated in the `PluginInfo`. The process for responding to requests for data is easy for the input modules to implement, partially fulfilling the final requirement from Section 5.2. These values are accessed by the corresponding aggregator module, which processes the data. We will discuss these aggregator modules next.

6.2.1 Aggregator Modules

The aggregator modules aggregate the estimates from the input modules. Essentially, they take in the different estimates that are used to calculate $L(C_n)$, $U(C_n)$ and $d(C_n)$. These estimates are retrieved from the the list of `PluginInfo` objects maintained by the plugin manager by filtering for the corresponding type of module. From there, each of the aggregators perform the calculations for each of the values:

Privacy For calculating privacy, the module takes all estimates $p_i(C_n)$ from the privacy input modules and combines them as per Equation 5.1. It then takes the estimate for context familiarity $\alpha_{\mathcal{C}}$ from the context familiarity module, and calculates $\mathcal{P}(C_n)$ as defined in Equation 5.2.

Unfamiliarity For calculating unfamiliarity, the module takes all estimates $u_i(C_n)$ that are combined to obtain the number of unfamiliar people $U(C_n)$ from Equation 5.5, and aggregates them in a manner similar to that of Equation 5.1, by computing a weighted average. Combining this estimate with the context familiarity module's estimate $\alpha_{\mathcal{C}}$ and proceeding to calculate $\mathcal{U}(C_n)$ via Equation 5.6.

It is important to note that the practical implementation of the context engine does not match the theoretical basis as outlined in Section 5.3.2. To be precise, the aggregation step is performed differently. Instead of reusing the value $L(C_n)$ and multiplying it by $(1-\mathcal{F}(C_n))$ as shown in Equation 5.5, we instead aggregate individual estimates $u_i(C_n) \in V$ and use them to calculate $U(C_n)$:

$$U(C_n) = \sum_{i \in 1 \ldots |V|} \frac{u_i(C_n) w_i}{|V| \sum W_{\mathcal{U}}}$$

$W_{\mathcal{U}}$ is currently not used (i.e., all modules have $w_i = 1$) in our implementation. The theoretical approach is not completely removed - recall it is used by the Bluetooth module as described in Section 6.1.1. This method allows for alternate means of estimating the number of unfamiliar people, making the context engine more adaptable. Instead of relying specifically on estimating the number of familiar devices, input modules can use whatever means they choose to decide upon the number of unfamiliar people nearby. Perhaps they track voices, or use face recognition. By decoupling the theoretical approach and relegating it to a particular input module, we gain flexibility in the implementation.

Proximity For calculating proximity, the module takes all estimates $r_i(C_n)$ for the distance between the device and user and aggregate them into the value $d(C_n)$ as in Equation 5.7.

The proximity module deviates from the other two approaches, in that it does not take any data from the context familiarity module. Instead, $\alpha_{\mathcal{D}}$ (which handles the decay rate of $\mathcal{D}(C_n)$) is static, set to 2. In addition, the proximity cutoff α_d is also static, set to 1. These static values and the aggregate are used as in Equation 5.8 to obtain the calculated value $\mathcal{D}(C_n)$.

Each of these results is then passed to the rule module, which handles applying these values to the rules provided by functionality modules. Before discussing the rule module, however, we will move on to the context familiarity module.

6.2.2 Context Familiarity Module

The context familiarity module is the module that tracks the device's familiarity with a particular context. Currently only locational contexts are supported, but eventually this could be expanded. This module keeps track of the context familiarity value $\alpha_{\mathcal{C}}$ for every context visited, and increments that value on repeated visits to track as the device's familiarity with the context increases. This value is provided to the aggregation modules when calculating $\mathcal{P}(C_n)$ and $\mathcal{U}(C_n)$.

The context familiarity module is implemented as a part of the `AggregatorModule` class, which is mainly responsible for aggregating the values from the input modules via plugin manager. Since the aggregator module is the only location context familiarity is used, it is sensible to include the context familiarity module here.

The module functions by periodically querying the Android OS for the last known location of the device. When the location is returned, the location is converted to a unique key through the following function:

```
fun createKey(lat: Double, lng: Double): String {
    val acc = 10000.0
    val latitude = ((lat * acc).roundToInt().toDouble() / acc)
    val longitude = ((lng * acc).roundToInt().toDouble() / acc)

    return ‘‘$latitude,$longitude"
}
```

This approach simply reduces the accuracy of the latitude and longitude to a roughly 100 metre area. If the location has been seen before, the corresponding familiarity value is updated, as in Equation 5.9. If not, the location is added to the database, with a default value of 3. This default value is explained in Section 5.3.4. However, this rather basic method of calculating context familiarity was not the only proposal.

Proposed Alternate Approach

An alternate approach that may be more useful would be to allow input modules to provide an estimate for the current context familiarity, allowing a synthesis of many different approaches, and allow extension of the system without necessitating any rewriting of the context engine. In this approach, estimates for $\alpha_{\mathcal{C}}$ would be provided by some input modules, and aggregated in a method similar to Equation 5.1:

$$\alpha_{\mathcal{C}} = \sum_{i \in 1 \ldots |P|} \frac{a^i_C(C_n) w_i}{|a_C| \sum W_{a_C}} \tag{6.1}$$

Where each $a_C^i(C_x) \in a_C$ is an estimate for context familiarity, which is weighted based on the confidence the engine has in that module, denoted by $w_i \in W_{a_C}$.

The final decision was to keep calculation of context familiarity within the context engine, as it is more sensitive than the other approaches and requires permissions (i.e. Location) that users are less likely to provide to every individual module. Modifying the framework to support an aggregated approach to estimating context familiarity could be interesting future work, however.

6.2.3 Rules Module

The rules module forms the interface between the aggregators and the functionality modules. Each functionality module defines a set of rules, taking the form of a rule that takes up to three inputs - $\mathcal{P}(C_n), \mathcal{U}(C_n)$ and $\mathcal{D}(C_n)$. These rules allow the functionality module to react to changes in the context as necessary. At fixed intervals, the rules module sends a message of type `MSG_PUBLISH_DATA` with the new values of $\mathcal{P}(C_n), \mathcal{U}(C_n)$ and $\mathcal{D}(C_n)$ to all registered functionality modules, so it may check these values against the rules they set, and adapt accordingly. The API for receiving context updates is easy to use allowing for easy extension of the system, fulfilling the rest of the final requirement from Section 5.2.

6.3 Functionality Modules

In order to give a sense of how the context values could be used, three basic modules were implemented - authentication, device loss, and device theft. While the implementations are mainly stubs designed to showcase use cases of the values, we will describe how they function, and what steps can be taken to improve the implementation. As explained, these modules interface with the context engine via rules. In this section, we will describe these rules for each module, and their justification.

Before continuing to the individual functionality modules, we will first discuss how each functionality module works. Since they are all mainly stubs used mainly as a proof of concept, their structure is extremely similar. They all consist of a single service. When the service starts, the service registers with the context engine. When running, the service listens for messages from the context engine of type `MSG_PUBLISH_DATA`. These messages are handled by their handler, and contain the final three aggregated values generated by the aggregator modules and propagated by the rules module. When the values match a

Rule	Action	Justification
$\mathcal{P}(C_n) - \mathcal{U}(C_n) < .1$	Enable Authentication Module	Presence of unfamiliar devices should trigger the module - in more private locations, the number of unfamiliar devices can be higher.
$\mathcal{P}(C_n) - \mathcal{U}(C_n) \geq .1$	Disable Authentication Module	
$\mathcal{P}(C_n) - \mathcal{U}(C_n) < -.4$	Start Auth High Alert	If a large fraction of people in the current context are unfamiliar, we want to be very careful. This may mean using explicit authentication.
$\mathcal{P}(C_n) - \mathcal{U}(C_n) \geq -.4$	End Auth High Alert	

Table 6.2: Module rules and justification for the authentication module. They are based on experimentation conducted using the Cambridge/Haggle dataset, discussed in Chapter 7, and are meant as proof of concept.

rule set by the functionality module, a message is sent to that corresponding functionality module.

The functionality modules function mainly through notifications - generally, they have a persistent notification display what their current state is (e.g. unlocked, locked) and occasionally post notifications when a certain change in context is detected (e.g. the device is being left behind). They do not currently do much beyond acting upon the context data compiled by the framework - this is left to future work, since it is not really the core of the project.

6.3.1 Authentication

The authentication module aims to enable or disable authentication of the device based on the reported values for $\mathcal{P}(C_n)$ and $\mathcal{U}(C_n)$. The module allows us to test how the additional context data could influence the behaviour of an authentication module, and see when and how the module would enable and disable explicit authentication. Currently, this module is fairly simple, and does not perform any additional implicit authentication. Instead, the authentication module performs two actions, following rules as outlined in Table 6.2:

1. Changes the state of a persistent notification stating whether the module is engaged or disengaged, depending on the variation between $\mathcal{P}(C_n)$ and $\mathcal{U}(C_n)$.
2. Sends a notification declaring the start and end of Auth high alert.

This is a fairly rudimentary implementation, as actual methods of implicit authentication are out of scope for the project. However, we will discuss how a more thoroughly implemented authentication module could behave next. In general, there are three main ways that an authentication module could use the device context to improve the user experience:

Disabling in "Safe" Contexts In this use case, the authentication module can recognize when a context is "safe" by noting when $\mathcal{P}(C_n)$ is high, and $\mathcal{U}(C_n)$ is low. In this case, the authentication module does not need to authenticate the user at all, or use an extremely basic authentication method. However, when these values change, the module can re-enable authentication.

Extending Existing Authentications When extending an authentication, context is used to establish continuity between the device and the owner. For as long as possible, an existing authentication is stretched until continuity cannot be confirmed with a satisfactory confidence level, leading to deauthentication. Consider a user that uses trusted Bluetooth devices to authenticate. However, they leave the house, forgetting their smartwatch, making the authentication no longer apply. However, they carry the phone around in their pocket (which the module uses to extend the authentication). Eventually, they place the device on a table, which finally breaks the authentication.

Deauthentication in Context Changes In the previous case, the authentication module recognizes that a certain authentication method is no longer applicable, and instead switches to relying on continuity to extend the authentication. In this case, the authentication module uses a particular change in the context to recognize that current authentication methods are not sufficient, and deauthenticates *itself*. For an example of this case, consider a context where the authentication module uses a specific location (e.g. their home) to authenticate the user. This continues to work until the device detects the presence of multiple unfamiliar people in this context, meaning that simply being at a particular

Rule	**Action**	**Justification**
$\mathcal{U}(C_n) > .5 \land \mathcal{D}(C_n) < .9$	Enable Device Theft Module	If there are unfamiliar people around, and the device is not on person, enable the device theft module.
$\mathcal{U}(C_n) \leq .1 \lor \mathcal{D}(C_n) \geq 1$	Disable Device Theft Module	
$\mathcal{U}(C_n) > .5 \land \mathcal{D}(C_n) \leq .2$	Start Audio Alarm	It is likely the device is being stolen.

Table 6.3: Module rules and justification for the device theft module. They are based on experimentation conducted using the Cambridge/Haggle dataset, discussed in Chapter 7, and are meant as proof of concept.

location is no longer sufficient to authenticate the user. The authentication module determines that in the current context, location is an insufficient method of authentication, and deauthenticates the user, forcing them to scan their fingerprint to unlock the device.

6.3.2 Device Theft

The purpose of the device theft module is to detect when the device is being stolen by an adversary, and prevent the theft - either by notifying the owner or sounding an alarm to discourage the thief. This is specific to contexts where theft is more likely, so the system should only enable when unfamiliar people are nearby. To this end, it relies on the context engine's estimates of $\mathcal{U}(C_n)$ and $\mathcal{D}(C_n)$.

Currently, the device theft module simply enables or disables itself (via persistent notifications), and starts an audible alarm if the device is too distant, following rules as outlined in Table 6.3. In the future, device theft modules would likely use the device sensors and collect data specific to device theft in order to raise the alarm, instead of simply relying on the context data (which is better suited to enabling and disabling the module, rather than performing the act of detecting device theft).

Rule	Action	Justification
$\mathcal{D}(C_n) \leq .75$	Enable Device Loss Module	Enable the device loss module if the device is not near the owner.
$\mathcal{D}(C_n) > .85$	Disable Device Loss Module	
$\mathcal{D}(C_n) < .3 \land \mathcal{P}(C_n) < .5$	Send Lost Device Notification	If the device is too far, attempt to notify the owner.

Table 6.4: Module rules and justification for the device loss module. They are based on experimentation conducted using the Cambridge/Haggle dataset, discussed in Chapter 7, and are meant as proof of concept.

6.3.3 Device Loss

The device loss module aims to notify the owner when they are likely unintentionally leaving the device behind when leaving a context. Device loss is more likely in public, since if the user leaves the phone behind on purpose, it is not device loss. Since purposefully leaving the device behind is more likely in private contexts, the module limits itself to more public contexts. It thus relies on estimates for $\mathcal{P}(C_n)$ and $\mathcal{D}(C_n)$ from the context engine to make decisions.

Currently, the device theft module simply enables or disables itself (via persistent notifications), and sends a priority notification if the device is too distant, following rules as outlined in Table 6.4. Similarly to the device theft module, a more fleshed out approach would likely gather its own data and rely on the context engine only for deciding when to enable and disable itself. However, unlike the device theft module, a rudimentary device loss module could be implemented using only the context data - all the necessary ingredients are there. This basic module does implement a fairly usable module as is. One method of improving on this could be to monitor the rate of change of proximity, allowing for a clearer picture of when a device is potentially being stolen.

6.4 Changes from CORMORANT

As mentioned before, the codebase for CORMORANT [20] formed the basis of our implementation. However, due to the different use cases of our system when compared to CORMORANT, there were many changes to be made. In this section, we will outline the largest changes made to the codebase. Instead of introducing the overall structure of the CORMORANT framework here, please refer to the work outlining their implementation [21].

Unnecessary Code Removal The CORMORANT codebase has a number of sections irrelevant to our approach. Namely, since the system was built around cross-device authentication, there are a number of different implementations across various platforms. To start, we limited ourselves to working only with the Android applications making up CORMORANT. In addition, there is a large section of the codebase dedicated to inter-device communication using the Signal protocol. All of this code is also unnecessary, as PUPy is confined to a single device. Therefore, all Signal code and inter-device communication was also removed. The next section of the existing codebase removed was the gait authentication plugin, which was one of the authentication plugins used by the system. Since an implicit authentication application was secondary to our system, and the gait analysis application was fairly complex, removing the application was a fairly easy choice.

At this point, all code not necessary for forming the basis of our implementation was gone. What remained was a number of example input plugins CORMORANT has, the central CORMORANT framework, and the CORMORANT API that is used by plugins to interface with the CORMORANT framework.

Conversion to Kotlin The next step was to convert the codebase to Kotlin. Kotlin is the new default for application development on Android, and our past projects that were implemented on Android used Kotlin. Since Android Studio includes a tool for automatically converting Java files to Kotlin, the conversion was fairly easy - manual cleanup took only a few days. The Kotlin conversion made further work on the codebase significantly easier, as Kotlin is now the primary supported language for Android development, and the author is better acquainted with the language.

Framework Overhaul The next step was to overhaul the central framework CORMORANT uses. Since the framework is built around predicting two values, risk and confidence, we needed to modify the code to accept three values instead - privacy, unfamiliarity and proximity estimates. The plugin manager was updated to accept three values, and the fusion and decision modules were overhauled to accept the three values and better fit our theoretical framework. The fusion module was converted to the aggregator modules, and the decision module became the rules module. Finally, while the CORMORANT framework has limited functionality allowing the propagation of the confidence level to other applications, this needed to be nearly entirely overhauled to match our functionality module approach.

Chapter 7

Evaluation

In order to properly evaluate the efficacy of the system we have implemented, we would normally test the Android implementation outlined in Section 6 through a user study. However, due to the COVID-19 pandemic, going to a large number of public places over many months is entirely out of the question, as public health orders restricted such activities. Instead, we take an approach shared by many previous works [15, 20], instead turning to existing datasets we can use to evaluate our system. This chapter will outline the evaluation of the system we performed using these datasets, and is structured as follows: Section 7.1 will outline the two datasets we used for evaluations. Section 7.2 will examine a number of specific scenarios in depth, and perform qualitative comparisons between our system and existing works. In Section 7.3, we will perform a more high level, quantitative analysis of our system and how it varies from existing works.

7.1 Datasets

For running evaluations of the system, we rely on two main datasets. For a high level, day-in-the-life look at how the system would react, we rely on the Cambridge/Haggle dataset [40]. For a more low level simulation of how the system would work when fed raw data, the MDC dataset [25, 27] is used. A description of the two datasets follows.

7.1.1 Cambridge/Haggle Dataset

The Cambridge/Haggle dataset [40] is a dataset containing a number of experiments. Each experiment is the compilation of Bluetooth sightings by a group of users carrying small Bluetooth devices (iMotes) for some period of time. For our testing, we use a single one of these experiments. This particular experiment was a deployment of iMotes among students for roughly a week in Cambridge. The resulting data outlines all sightings of Bluetooth devices, both those known as familiar (other students/participants) and those considered unfamiliar (anyone not in the experiment). This allows us to fairly easily adapt their dataset for our own uses.

For our use, we consider any other participant as *familiar*, and any non-participant as *unfamiliar*. These sightings are then used to calculate our values for privacy and unfamiliarity as outlined in Section 5.3. In addition, some stationary iMotes included in the dataset are set up in the shared laboratory, which was adapted as a location of high context familiarity for the purposes of our evaluation of the system using this dataset. To augment this, the top three Bluetooth addresses were also added as frequently visited (and thus high context familiarity) locations as well, giving us a sufficient amount of sightings to draw from for calculating context familiarity. Finally, proximity data was randomly generated as there was no reasonable way to derive it from the dataset, which is only tracking times of sightings, not proximity to the user or anything else.

The point of evaluation using this dataset is not to evaluate the ability of the system to learn over time, or provide a dense graph to look at, but to serve as a simple proof-of-concept for the system. The lightweight nature of the dataset allowed rapid iteration and testing of the system, and tweaking of the logic and equations.

7.1.2 MDC Dataset

The MDC Dataset [27] is a large dataset based on the Lausanne Data Collection Campaign [25]. It is composed of a massive amount of data collected mainly through sensors and logs of mobile devices used by the participants, for periods as short as a few weeks to more than a year. The vast quantities of data collected allows extremely in-depth analysis of how PUPy would function in the real world, and it or similar datasets have been used by similar projects in the past to simulate long-term usage of such systems [15, 20].

However, there are many datasets such as this one. The reason this dataset in particular was chosen is twofold - it is used by Gupta et al. to test their familiarity system, a work whose contributions formed an important part of the theoretical framework in Section 5.3,

Type	**Quantity**
Bluetooth Addresses	587 000
Bluetooth Sightings	34 000 000
Location (GPS)	12 000 000
Location Inferences (based on network addresses)	12 000 000
Visited Places	493
Wireless Network Sightings	57 000 000

Table 7.1: The type and quantity of records in the MDC dataset that were used in the experiments.

	Mean	Standard Deviation	Median
Overall Time	358 days	149 days	374 days
Proximity Time	194 days	94 days	206 days
Safe Visit Time	171 days	91 days	159 days
Authentications	11882	8523	10025

Table 7.2: Statistics on the MDC dataset.

and it used a viral marketing approach to recruit participants. This approach ensured that many of the participants regularly interacted with each other, ensuring we would have a solid basis for learning what devices are familiar.

The dataset is comprised of data collected from nearly 200 participants. The dataset crucially includes location, network and Bluetooth data, all data points that are extremely helpful in calculating values for alpha, privacy and unfamiliarity. Acceleration data and inferences on user behaviour based on said acceleration data is also available, which is used to estimate proximity as well. Table 7.1 outlines the quantity and type of the various records we used when calculating the values.

In order to gain a better understanding of the sort of quantity of data the dataset contains, we can look to Table 7.2. This table shows some statistics on the average timespan of each user in the dataset. On average, the user data recorded spans roughly a year, with standard deviation of roughly half a year. This gives us plenty of time to analyze how the framework learns and reacts. However, in order for the proximity calculations to work, it is necessary that proximity data is also included - often, that data does not span the entire length of a user's timespan. On average, there is roughly 190 days of proximity data, with a standard deviation of 94 days. This is generally the section of the dataset our graphs

Safe Context	**Unsafe Context**	**Unclassified Context**
My home	Holiday resort or vacation spot	Home of a friend
My free time home	Shop or shopping center	My other workplace
My main workplace	Location related to transportation	My main school or college
	Place for indoor sports	Other
	Place for outdoor sports	I do not know

Table 7.3: How different contexts were mapped to safe or unsafe contexts.

focus on, instead of the entire timespan.

A subset of the users in the dataset also collected data pertaining to their current context periodically throughout the experiment. This data is what we use to extrapolate the ground truth data for evaluation later on. On average, there is around 170 days of context data from "safe" contexts as determined by the mapping in Table 7.3, which matches the mapping used by Gupta et al. in their evaluation [15]. In general, there was very little "unsafe" context data, and not every user collected unsafe ground truth data. Those who did had around two to three days of unsafe context to evaluate on.

The final statistic of interest is the number of detected authentications. To calculate the number of detected authentications, application usage logs from the MDC dataset were used, which log any interaction between the user and any application. We assume that an authentication happens when application usage takes place 310 seconds after the last usage. This number is based on the work of Harbach et al. [16], which shows that on average, users use their device for 70 seconds, with a standard deviation of 240 seconds. We assume the device must reauthenticate if it has not been used for more than one standard deviation from this average. On average, there are just under 12 000 authentications per user, with a standard deviation of 8 523 authentications. These authentications are what form the backbone of our evaluations.

By comparing the classification of a given authentication to the ground truth context, we can get a sense of if the system is properly classifying contexts. That is, if our ground truth data tells us a given context is usually considered safe, we would expect a higher proportion of the authentications in that context to be skipped, or marked as safe. Conversely, we would expect a higher proportion of authentications in unsafe contexts to require authentication, or be marked as unsafe.

7.2 Qualitative Analysis

In this section, we will perform a qualitative analysis of the framework, as well as outline steps taken to test the framework and investigate its efficacy. This section will focus mainly on examples and less on statistical analysis, which we will leave for Section 7.3. We will begin with outlining the first steps taken in evaluating the framework via the Cambridge/Haggle dataset, then discuss the bulk of the evaluation, completed using the MDC dataset.

7.2.1 Visual Analysis Based on the Cambridge/Haggle Dataset

In order to get a sense of if PUPy would actually function as expected, before working with the far more complex MDC dataset, we first created graphs based on the Cambridge/Haggle dataset, which is significantly simpler and smaller, allowing for rapid iteration and testing. In contrast with the MDC Dataset, the Cambridge/Haggle dataset spans only a few weeks at most. This means that we could not really allow the framework to learn over time - instead, we used the Cambridge/Haggle dataset as a simulation of how the system would function after several months of use.

In order to accomplish this, we broke up the Bluetooth sightings used into a number of categories. The Cambridge/Haggle dataset assigns a numeric id to each unique Bluetooth address - we harnessed this by using the numeric ids linked to other students as familiar devices, and most other devices as unfamiliar. This allowed us to obtain estimates for $\mathcal{P}(C_n)$ and $\mathcal{U}(C_n)$ by counting the number of familiar and unfamiliar devices in a given context. However, since the Cambridge/Haggle dataset is only Bluetooth sightings, we did not have any data through which to calculate $\mathcal{D}(C_n)$,. To get around this, we simply generated random estimates for proximity.

With these measures complete, the only value we did not yet have any data for was context familiarity. To obtain estimates for context familiarity, we combined two approaches:

1. Using stationary iMotes scattered around Cambridge. Depending on where the iMote was located, it was matched to a particular context familiarity value. For example, iMotes in public spaces mapped to low values for context familiarity, while the iMotes at the entrance to the computer lab were marked as locations with high context familiarity. There were 18 iMotes scattered across the city.

2. Using the most seen numeric ids by each user as highly familiar locations. This does not reflect reality (as we have no guarantee these addresses were actually stationary),

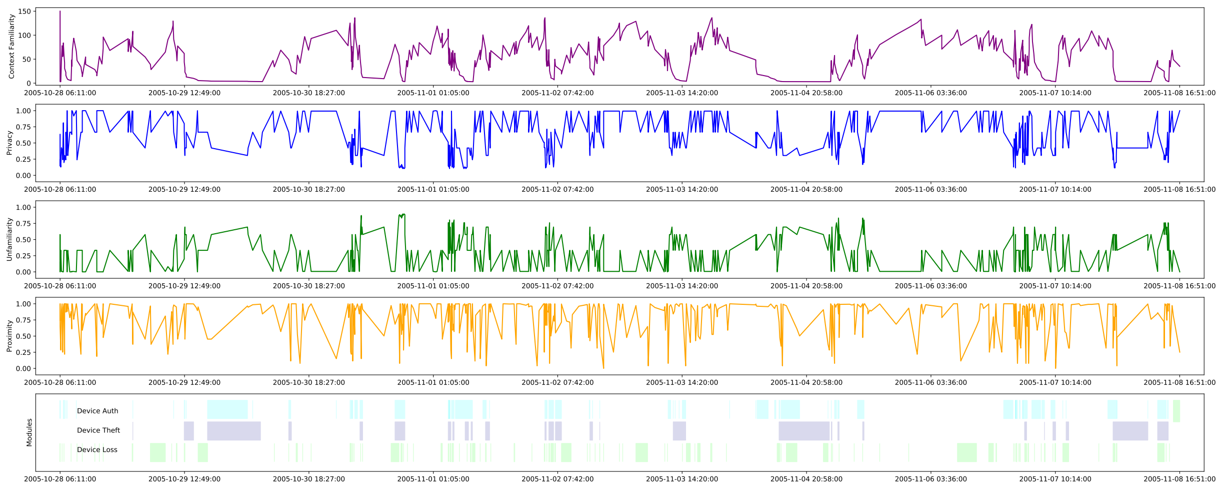


Figure 7.1: An example of the sort of graph generated using the Cambridge/Haggle dataset.

but allows us to have different contexts with high context familiarity across different users, adding additional variation to each user's graph.

The combination of these two measures allows us to simulate context familiarity. With all four values now accounted for, we can create graphs of them. An example graph is shown in Figure 7.1. The graph contains five sections:

1. The context familiarity section. This section marks the context familiarity value $\alpha_{\mathcal{C}}$ for each location, and how it fluctuates over time in purple.

2. The privacy section. This section shows the privacy value, $\mathcal{P}(C_n)$, over time in blue.

3. The unfamiliarity section. This section shows the unfamiliarity value, $\mathcal{U}(C_n)$, over time in green.

4. The proximity section. This section shows the proximity value, $\mathcal{D}(C_n)$, over time in yellow.

5. The actions section. This section shows when each of the three functionality modules we simulate are enabled. The top row, in cyan, is the authentication module. The middle row, in purple, is the device theft module. The bottom row, in lime, is the device loss module. If the above values for $\mathcal{P}(C_n)$, $\mathcal{U}(C_n)$, and $\mathcal{D}(C_n)$ match the rule that enables a functionality module, that section of the row is highlighted in its

corresponding colour. On the other hand, if the values match the rule that disables any of the modules, that part of the row is left empty.

Looking at the resulting graphs, it was clear that PUPy works as expected, with modules getting enabled and disabled depending on the values $\mathcal{P}(C_n)$, $\mathcal{U}(C_n)$ and $\mathcal{D}(C_n)$. Heartened by this rudimentary implementation, we moved on to the MDC dataset, and creating graphs from it.

7.2.2 Visual Analysis Based on the MDC Dataset

In order to get a good sense of how PUPy reacts to device sensor data, we created a script that builds graphs of the values resulting for each user's data, and showed how those values affected the enabling and disabling of the modules. We will look at a few specific users in this section, investigating how the system reacts in various situations.

To start off, we will take a bird's eye view of what the system looks like over the total lifetime of a given user's data. Some example graphs have been included in Figure 7.2. These examples were included instead of others due to a number of factors. Many users in the dataset did not regularly collect data, so there are only three or four data points for any given day. Others participated only for a few days or weeks. The examples selected here span a large amount of time, and have a consistently dense amount of data to draw from.

We will break down these graphs and explain what they represent. In each graph, there are six sections:

1. The scenario section. This is split into two parts - the top shows the ground truth data (if available for the given user), and the bottom shows each authentication of the device, as described in Section 7.1.2. If the ground truth data tells us the current context is safe, it is shown as green. If it is unsafe, it is shown in red. For authentication attempts, if the authentication module was disabled when the authentication takes place, the authentication is marked as a green line. If the module is enabled, the authentication is marked as a red line.

2. The context familiarity section. This section marks the context familiarity value $\alpha_{\mathcal{C}}$ for each location in purple, and tracks how it fluctuates over time.

3. The privacy section. This section shows the privacy value, $\mathcal{P}(C_n)$, over time in blue. If there is a significant gap between the values of $\mathcal{P}(C_n)$ and $\mathcal{U}(C_n)$ at a particular time, that time is highlighted in gold.

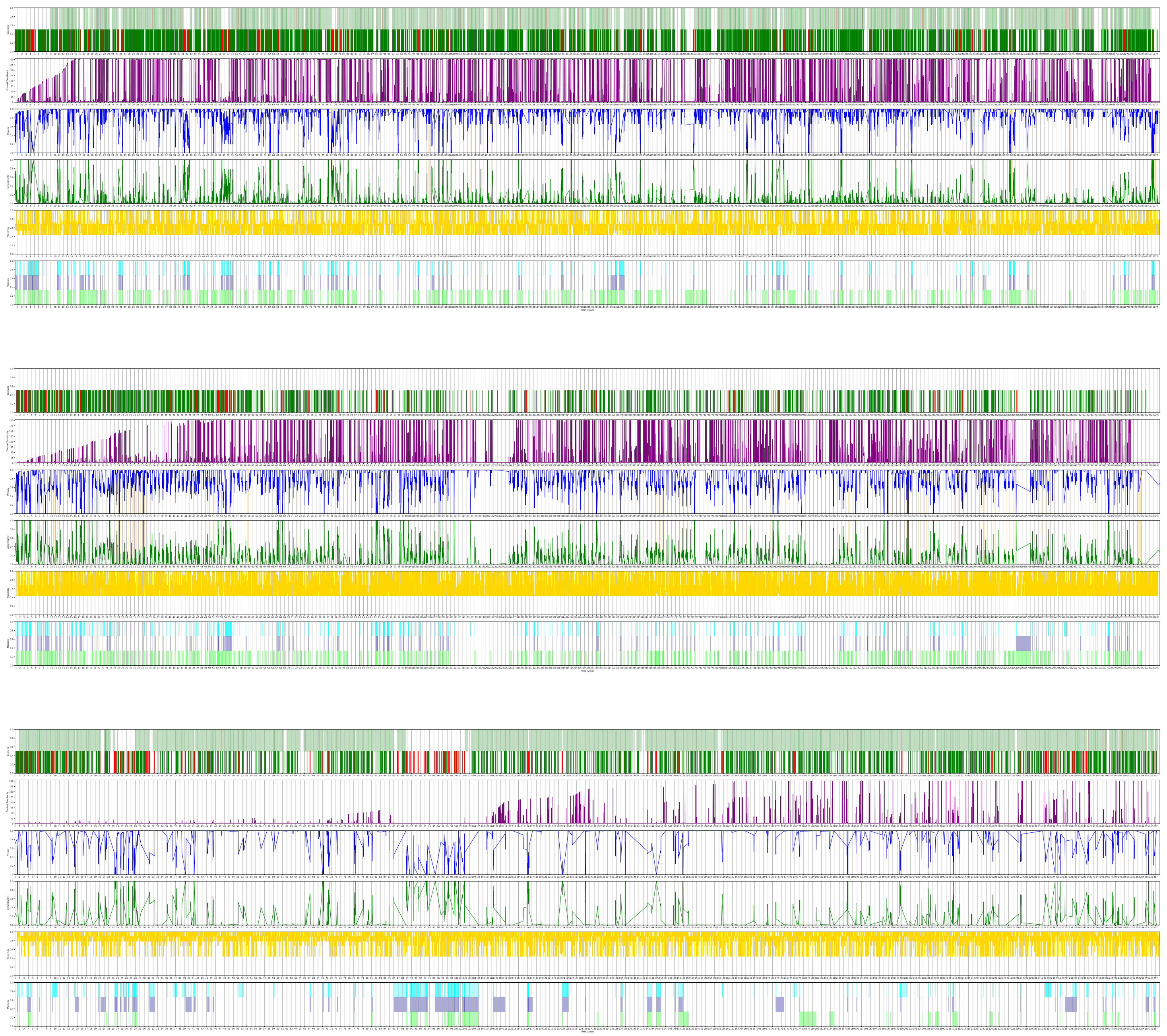


Figure 7.2: A selection of graphs from some users in the MDC dataset. Depicts the entire duration of the user data.

4. The unfamiliarity section. This section shows the unfamiliarity value, $\mathcal{U}(C_n)$, over time in green. If there is a significant gap between the values of $\mathcal{P}(C_n)$ and $\mathcal{U}(C_n)$ at a particular time, that time is highlighted in gold.

5. The proximity section. This section shows the proximity value, $\mathcal{D}(C_n)$, over time in yellow.

6. The actions section. This section shows when each of the three functionality modules we simulate are enabled. The top row, in cyan, is the authentication module. The middle row, in purple, is the device theft module. The bottom row, in lime, is the device loss module. If the above values for $\mathcal{P}(C_n)$, $\mathcal{U}(C_n)$, and $\mathcal{D}(C_n)$ match the rule that enables a functionality module, that section of the row is highlighted in its corresponding colour. On the other hand, if the values match the rule that disables any of the modules, that part of the row is left empty.

The gold highlights in the third and fourth sections are based on a large gap between $\mathcal{P}(C_n)$ and $\mathcal{U}(C_n)$. If a context has entirely unfamiliar devices, we would expect that $\mathcal{P}(C_n)$ would be equal to $1 - \mathcal{U}(C_n)$. This is because, per Equation 5.5:

$$\mathcal{U}(C_n) = \frac{\alpha_{\mathcal{C}}^{1-\frac{1}{U(C_n)}}}{\alpha_{\mathcal{C}}}$$

and since $U(C_n) = L(C_n) * (1 - \mathcal{F}(C_n))$, when $\mathcal{F}(C_n) = 0$, we have that $U(C_n) = L(C_n)$. This makes intuitive sense - if there is no one around you who is familiar, then the number of unfamiliar people is exactly equal to the total number of people. Substituting, we get:

$$\mathcal{U}(C_n) = \frac{\alpha_{\mathcal{C}}^{1-\frac{1}{L(C_n)}}}{\alpha_{\mathcal{C}}} \iff \mathcal{F}(C_n) = 0.$$

Finally, recall that, by Equation 5.2:

$$\mathcal{P}(C_n) = 1 - \frac{\alpha_{\mathcal{C}}^{1-\frac{1}{L(C_n)}}}{\alpha_{\mathcal{C}}}$$

and we can see that:

$$\mathcal{P}(C_n) = 1 - \mathcal{U}(C_n) \iff \mathcal{F}(C_n) = 0.$$

Why is this of interest? Because if $\mathcal{P}(C_n) \neq 1 - \mathcal{U}(C_n)$, it tells us that $\mathcal{F}(C_n) > 0$. When there is a large gap, that means $\mathcal{F}(C_n)$ is likely very high, and thus we are surrounded by a large number of familiar people. This sort of event, when privacy is very low but unfamiliarity is also low, is where the additional context data our system provides allows for functionality modules to better adapt to the context. For example, there is clearly a very big difference in a context when you are at a dinner party and everyone is a stranger, and when you host a dinner party with your ten closest friends. Therefore, we keep an eye out in our calculations for such situations. If such situations are widespread, it means that there are many contexts in which our additional context data is particularly useful over other existing works.

7.2.3 Visual Analysis of Familiarity Improvements

One of the important parts of the framework is the improvements made to how it adapts over time, through its use of familiarity. The usage comes in two types - context familiarity and unfamiliarity. The latter of these two types aims to improve on the familiarity system devised by Gupta et al. [15]. In order to get a better idea of how the system is adapting, we examined narrow slices of the previous graphs. Since they often span hundreds of days, zooming in and looking at particular points in time along during the length of the experiment allows us to look, in a more fine grained manner, how the devices respond to specific scenarios. In this section, we will look at a few of these particular scenarios, and discuss how the system reacted.

Adaptation through Context Familiarity

One area of particular interest was how the context familiarity system learned over time. To examine how quickly it learns about new contexts, we examine the start of the graph. As seen in Figure 7.3 the familiarity value for a particular context is incremented once every visit. So as time goes on, and the number of visits to a particular context increases, so does context familiarity. In the case of our example user, it took a particular location (likely their home, or office) about a month and a half of consistent visits to hit the maximum value of 200. While this could be improved through the use of user feedback, there is not a reason to - the system functions perfectly well without immediately setting the home to 200. It may be a bit too sensitive, but otherwise works well. This is due to the role context familiarity plays in calculating privacy and unfamiliarity - it influences only how quickly or slowly the value changes, and does not provide input data. This approach is key to being able to limit the need for user feedback.

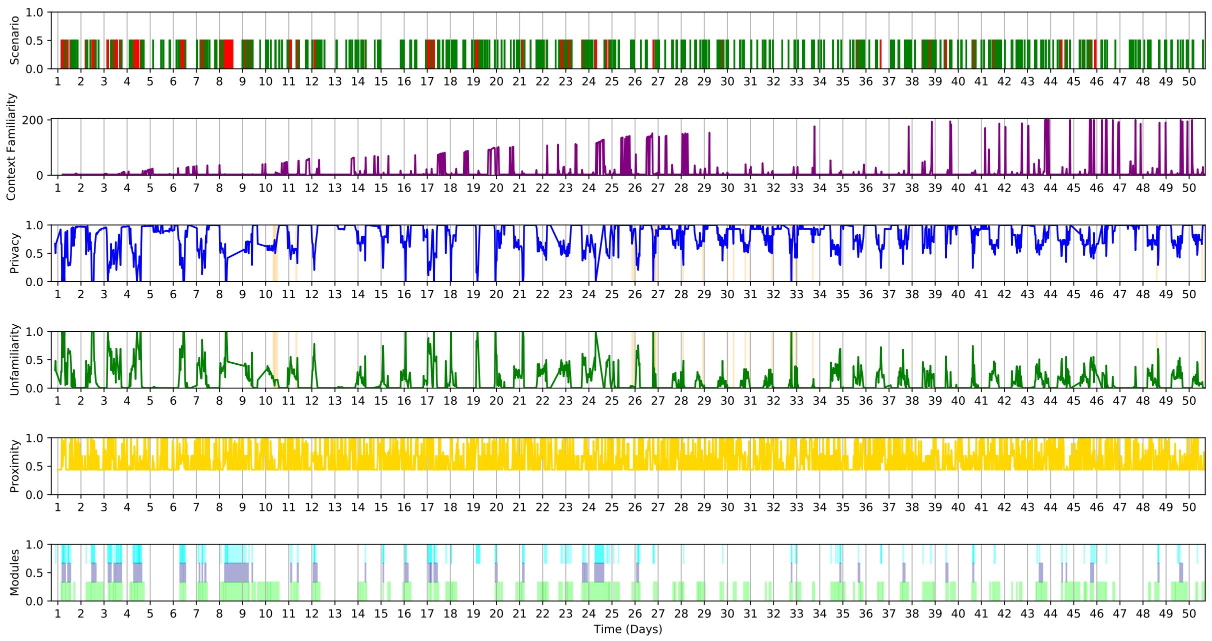


Figure 7.3: An example of learning behaviour of the context familiarity module.

Adaptation through Privacy and Unfamiliarity

The gold highlights (as discussed earlier) single out contexts where there is a large gap between privacy and unfamiliarity, which are of particular interest to us. As will be discussed more thoroughly in Section 7.2.4, these are contexts where our system provides better detail and nuance than the original familiarity system. To understand what gives rise to these contexts, we will examine a close up of one such example. In Figure 7.4, we see the graphs from before, focused on a 6 day timespan. As can be seen in this closer view, the highlights occur mainly when privacy is somewhat low (i.e., $\mathcal{P}(C_n) \approx .5$), but unfamiliarity is even lower. This tells us that these sorts of contexts occur mainly when the user is in a group of people, some or all of which are familiar to them.

7.2.4 Comparison with Gupta et al.

The approach taken in PUPy carries a number of strong similarities to the work of Gupta et al., who consider the metrics instantaneous familiarity and aggregate familiarity [15]. Indeed, the method for calculating instantaneous familiarity has an important influence

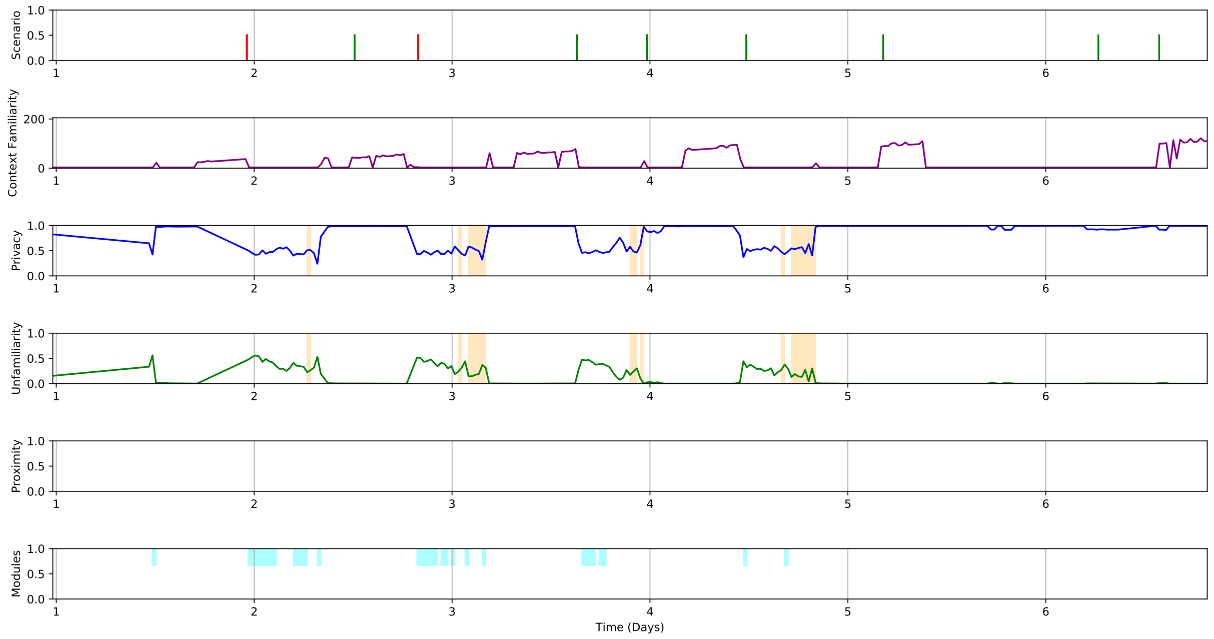


Figure 7.4: A closer look at some highlighted scenarios.

on how unfamiliarity is calculated. However, the approach taken by Gupta et al. and the approach taken by PUPy have some key differences, which allows it to better reflect a wider set of circumstances.

Before continuing, let us outline the definitions of instantaneous and aggregate familiarity. For instantaneous familiarity, we already defined it in Section 5.3, in Equation 5.4:

$$\mathcal{F}(C_n) = \frac{1}{|D_{C_n}|} \sum_{d \in D_{C_n}} F_d(d, C_n)$$

where $F_d(d, C_n)$ is device familiarity (Equation 5.3), and D_{C_n} is the set of devices in the context C_n. Aggregate familiarity is the aggregate of all previous values of instantaneous familiarity for the current context:

$$\text{aggFam}(C_n) = \alpha_C \mathcal{F}(C_n) + (1 - \alpha_C)\mathcal{F}(C_{(n-1)})$$

where α_C is a suitably chosen constant. We have adapted the notation from their paper to better match ours.

Gupta et al.'s system is based on geolocational contexts and the devices therein. Removing this requirement and focusing instead merely on the people in a given context frees us from the location-based restrictions of their system. As long as we can obtain an estimate for how many people are around, PUPy can remain functional - moving or otherwise. Since the only value that relies on location is context familiarity, simply using the default while moving allows PUPy to continue to provide estimates while moving between various stationary contexts.

Consider the method of calculating privacy. At its root, the only value it depends on is the number of people around, $L(C_n)$. This can be detected through any number of methods - detecting the number of distinct voices, the number of nearby phones, etc. Calculating this value does not necessarily depend on there being Bluetooth devices nearby that we can categorize as familiar or unfamiliar, and is not hampered by linking the context to a particular location. This stands in contrast to the Gupta et al. system, which relies on calculating the familiarity of devices seen in a given location.

To highlight the issues caused in non-static contexts by the Gupta et al. system's dependence on detecting similar devices, imagine a context where the owner is walking down a quiet street. There is no one around, and thus no devices are around. How do the actions of the Gupta et al. system and PUPy diverge? In the case of the Gupta et al. system, the framework reports low instantaneous familiarity, since it requires the presence of devices to increase this value. This locks the phone down, requiring an authentication. This will persist, no matter how frequently the owner walks down this street - instantaneous familiarity will always be low.

Conversely, consider privacy. The absence of devices and noise suggests there are few people around, meaning that $\mathcal{P}(C_n)$ is likely fairly high, accurately reflecting the situation. However, if there are a group of people walking along the street, the device can detect these Bluetooth devices, and perhaps the conversation happening in the group, and decrease $\mathcal{P}(C_n)$ to accurately reflect the situation. Using these different values for $\mathcal{P}(C_n)$, we can modulate how we authenticate more effectively in this scenario than the Gupta et al. system.

In a similar vein, imagine a context in which there are no devices to be found. In the case of privacy, this can be used as evidence that $\mathcal{P}(C_n)$ is high, using the logic that few devices means there are few people. The Gupta et al. system, however, run into trouble. Since the whole system is based on detected familiar devices, no devices to detect means instantaneous familiarity cannot be ascertained. The Gupta et al. system is built to work only when there are plenty of devices around to learn about. PUPy (and in particular $\mathcal{P}(C_n)$), with its focus on the number of people, does not have this issue.

Similarly to $\mathcal{P}(C_n)$, $\mathcal{U}(C_n)$ responds well to cases where there are few people/devices around. Since it is also rooted in counting the number of unfamiliar people, if there is no one around it also reports a more accurate value. Since there is no one around, there are necessarily no unfamiliar people around, either. But because the value is based on the same familiarity definition used by Gupta et al., it responds similarly in contexts where there are plenty of people around, familiar or unfamiliar.

Taking a broader angle, the system described in this paper is fundamentally optimistic - it assumes the context is private and safe, until evidence to the contrary is provided. This allows it to function in more contexts than the Gupta et al. system, which is fundamentally pessimistic - it assumes all locations are unsafe, over time building up a sense of trust for particular locations. This can cause trouble when used in many contexts. Our optimistic approach goes against common practice in security - a pessimistic approach is far more safe. The tradeoff being made here is one of security and usability - while the system described in this paper is less secure, it is more usable, a tradeoff that makes sense for many.

There are, of course, still examples where such a system is less secure - one particular example is when there is a change in context too rapid for the device to detect. Due to hardware limitations, we can only collect sensor data at a certain frequency. If the system is in a private location with no unfamiliar devices when sensor data is polled, and the context rapidly changes to a more unsafe context, there is a window where the device incorrectly identifies this high threat context as a low threat one. For a practical example, a carefully planned (or particularly unlucky) mugging may be a vulnerability of the system. This can be mitigated with high frequency polling, which naturally would have an effect on the device's battery. It is an interesting challenge to balance these issues, one that has not been addressed in this work. The Gupta et al. system, however, does not have this issue. Since it defaults to a location being unsafe, unless the mugging took place in a familiar location (i.e., your home), the device will be locked, since that is the default.

7.3 Quantitative Analysis

In this section, we will look at various metrics and statistics, to see how the amount of data and length of use impacts the result of our system. We will then look at a number of performance statistics, which will allow us to evaluate the efficacy of PUPy.

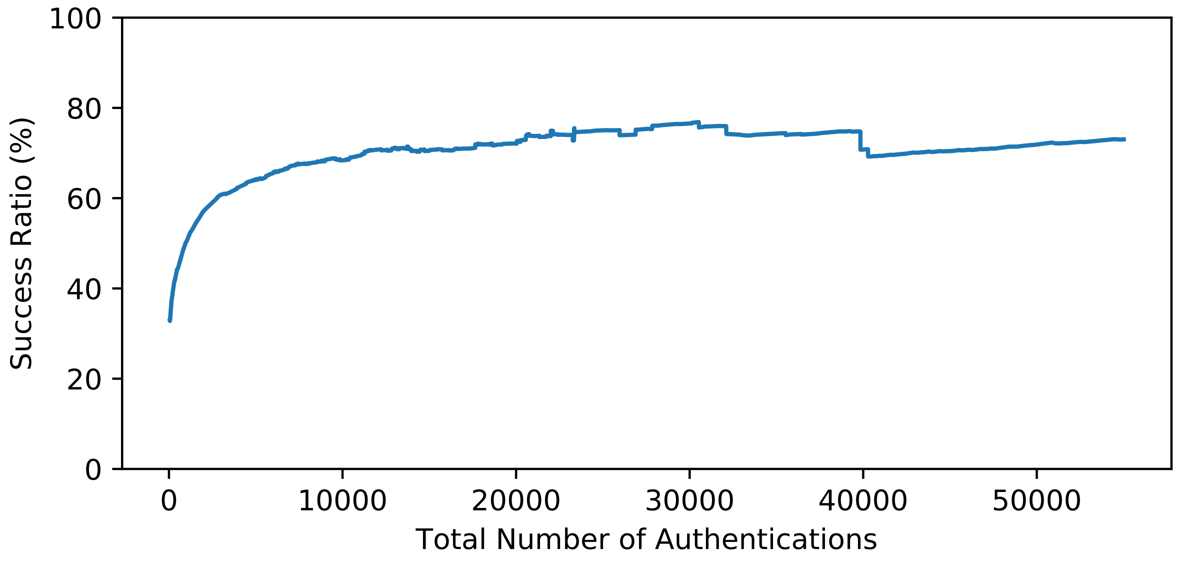


Figure 7.5: The cumulative success rate of the system in allowing authentications over the total number of authentications.

7.3.1 How Performance Changes Over Time

There are two sections of PUPy that can benefit from having a large amount of time and data to learn from when determining the context. The first of these two sections is the calculations we perform for unfamiliarity, which learns what devices are familiar and which are not over time. The second is the value of context familiarity, which increases as contexts are routinely visited. To examine the impact time has on the performance of the system, we will look at a number of metrics.

Figure 7.5 plots the cumulative success ratio over time. The success ratio is defined as follows:

$$\frac{\text{number of authentications when the authentication module was disabled so far}}{\text{total number of authentications so far}} \times 100$$

As the system learns, we would expect this ratio to increase, as more and more unlocks do not require an explicit authentication. That is exactly what we see - as the number of authentications increases, the success ratio also increases, eventually stabilizing around 80% success.

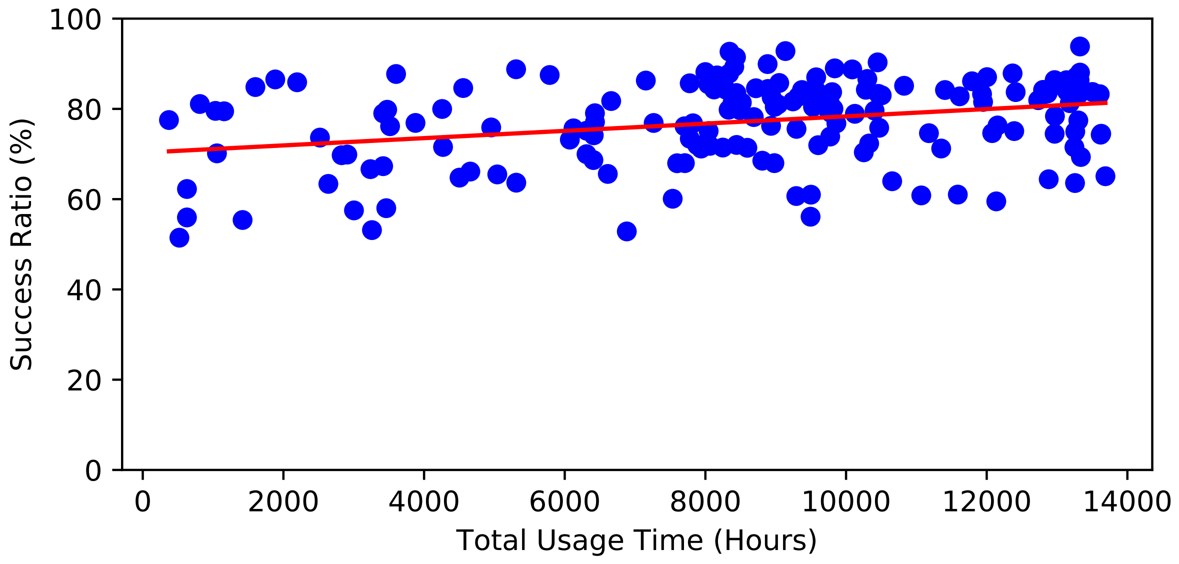


Figure 7.6: The success ratio for different users, with different total usage time.

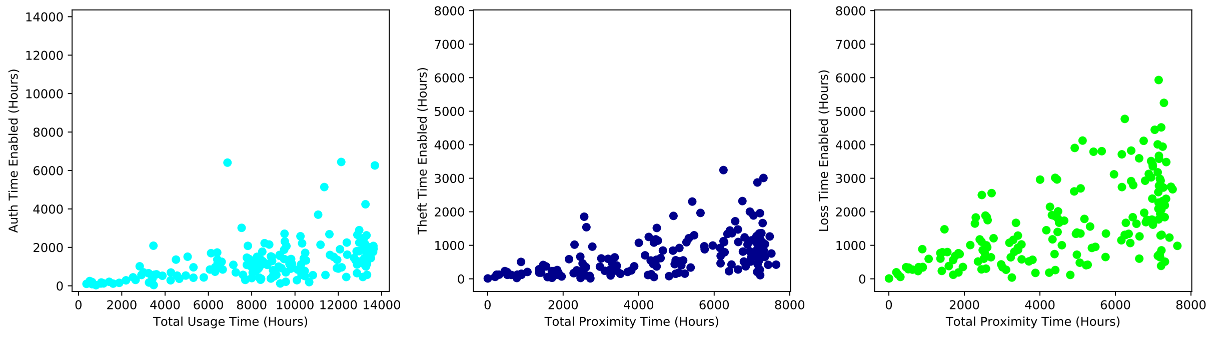


Figure 7.7: The relationship between the total usage time and the amount of time each module is enabled.

Figure 7.6 shows the overall success ratio for each user, depending on the total amount of time spent with the device. The red line represents the best fit line, which shows us that while there is a weak correlation between the total time a user uses the device and the success ratio, the system is able to learn fairly quickly. If the device is used for any reasonable length of time, there is a success ratio above 50%, meaning half of all authentications are implicit using PUPy.

Figure 7.7 shows the relationship between the total time the device is used and the amount of time each of the three modules are enabled. The graphs are square, so if the module was enabled 100% of the time, the dot would lie on the diagonal. If the system did not learn over the life of the device, we would expect to the see the amount of time enabled grow roughly linearly. Instead, barring a few outliers, the amount of time enabled grows slower for authentication and theft, the two modules that rely on data that adapts over time. Proximity, however, does not have any inputs that learn over time, hence the steeper growth pattern. This behaviour is expected. While both device theft and device loss rely on proximity to detect when the owner and device are growing further apart, device theft also relies on unfamiliarity, which adapts over time.

7.3.2 Overall Performance

In this section, we will break down some overall performance statistics of how PUPy performs. Firstly, we will discuss the success ratio. As discussed previously, the success ratio is the ratio of the number of authentications that happened when the authentication module was disabled, over the total number of unlocks. Essentially, it tells us how frequently the user does not need to authenticate at all. In Table 7.4, the ratio is expressed as a value in the range $[0, 1]$. On average, the ratio was 0.77, or a 77% reduction. This means that only around 23% of authentications occurred while the authentication module was enabled. It is important to note that, with a fully featured authentication module building on this, this ratio could likely be even higher, as some authentications will take place implicitly.

Despite PUPy being mainly a context detection framework and not directly authenticating the user, PUPy compares nicely to a number of existing works - Progressive authentication [38] saw a reduction of 42%. CORMORANT [20] was able to achieve a reduction of 97.82%, but requires a much larger framework overall - it combines authentication data from multiple devices, and requires the application to be installed on all devices. ConXSense [32] achieves similar performance to us, relaxing security in roughly 70% of contexts.

The next statistic of interest is the total number of highlighted contexts, as discussed in

	Mean	Standard Deviation	Median
Success Ratio	0.77	0.09	0.8
Total Highlighted Contexts	25	44	8
Auth Time	52 days	44 days	46 days
Theft Time	29 days	25 days	23 days
Loss Time	69 days	51 days	56 days
Auth Ratio	0.1	0.1	0.1
Theft Ratio	0.2	0.2	0.1
Loss Ratio	0.4	0.3	0.4
High Context Familiarity	28	16	25
Medium Context Familiarity	44	29	36

Table 7.4: Additional statistics outlining the performance of PUPy.

Section 7.2, that the user encounters on average. The users encountered 25 such contexts on average (with standard deviation 44) where the privacy and unfamiliarity values diverged, showing us that our system provided better context information at these points. As we can see with the high standard deviation, some users benefitted far more from this approach, so it seems to be a feature only a smaller subset of users will use.

The next set of metrics is the total amount of time each of the three modules were enabled (to be compared against Overall Time and Proximity Time from Table 7.2), and their corresponding ratios. Comparing these, we see that the device loss module was by far the most active module, activated roughly 40% of the time. This makes sense, as it has the most general rules governing it. On the other hand, the authentication and theft modules were only enabled roughly 10% and 20% of the time, respectively.

The final two metrics of interest are the number of locations that, by the end of the timespan, had high and medium context familiarity values, as discussed in Section 5.3. This denotes the number of places the user visited over their timespan with enough frequency to increase context familiarity beyond 175 for high familiarity locations, and 100 for medium familiarity locations. The high context familiarity value is somewhat surprising - on average, 28 locations had a context familiarity value greater than 175. This seems high, though it is important to note this is generally spanning nearly a year of data. If we account for the fact each location is a roughly 100m area, simply working in a large workplace or occasional GPS inaccuracy could artificially inflate this number. However, if this value is, in fact, too high, a simple fix is to decrease the rate at which context familiarity grows. Another approach could be to slow down the growth of context familiarity as it approaches the maximum. The value for medium context familiarity is more in line with what one would expect. On average, 44 contexts are marked as medium familiarity.

7.3.3 Comparison with Gupta et al.

While qualitative arguments comparing PUPy to other works can be useful, it is equally if not more convincing to see how frameworks react to similar inputs. To that end, an attempt has been made to simulate related works in a similar manner to how PUPy is simulated, and compare the results.

The most obvious existing work to compare PUPy against is that of Gupta et al. [15]. Portions of the system outlined by Gupta et al. form the basis for the unfamiliarity calculation in PUPy. However, the implementation of the familiarity system varies with regard to our system, along with what metrics are used for evaluation. In order to compare the two, we will re-implement their theoretical framework, using the computations and statistics generated through this project.

Before continuing, it is best to define a number of terms we will use in our evaluation. Firstly, precision, which is defined as:

$$\text{Precision} = \frac{|G_{safe} \cap C_{GREEN}|}{|C_{GREEN}|}$$

where G_{safe} is the number of safe contexts in the ground truth dataset, and C_{GREEN} is the number of contexts classified as safe/green by the system. Precision essentially tracks how often a context we classify as safe/green is also safe in the ground truth data. Another important metric is recall, defined as:

$$\text{Recall} = \frac{|G_{safe} \cap C_{GREEN}|}{|G_{safe}|}$$

Recall is used to track how often we classify a safe context from the ground truth dataset as safe/green. The final metric of interest is fallout:

$$\frac{|G_{unsafe} \cap C_{GREEN}|}{|G_{unsafe}|}$$

Fallout tracks what proportion of a given set (in this case, ground truth unsafe contexts) are classified as safe/green. We use this to track how likely a system is to misclassify various types of contexts. Similar metrics for precision, recall and fallout can be calculated by using G_{unsafe} and C_{RED} as well, to measure the performance of the system when classifying unsafe contexts and other cases. We will now move on to discussing the new evaluation method.

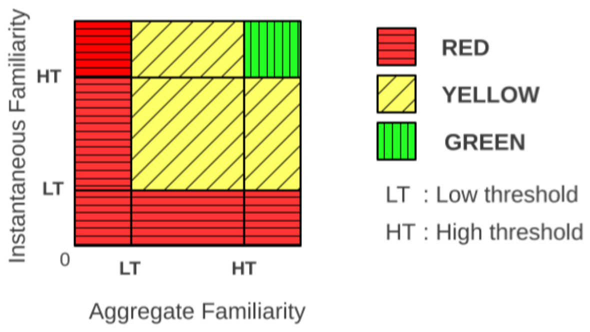


Figure 7.8: Security mapping for familiarity used by Gupta et al.

Gupta et al.'s Evaluation Method

The Gupta et al. system is evaluated in their paper using ground truth, which is created from the place data included in MDC dataset. This dataset is generated through a subset of the MDC dataset, which categorizes various contexts the participants visit via labels - home, friend's house, restaurant, et cetera. Gupta et al. categorize these labels into safe, risky or uncategorized, and compares the result from the familiarity system to this ground truth.

Their system is mainly interested in classifying various locations into one of three modes - red, yellow or green, denoting the predicted safety of the location as shown in Figure 7.8. The aim of their evaluation is for the system to evaluate "safe" contexts from the ground truth dataset as green, and "unsafe" contexts as red. In order to do this, they define two thresholds, delineating the boundaries between the three modes. These thresholds are 0.4 (marks the boundary between red and yellow), and 0.85 (marks the boundary between yellow and green). They calculate a number of different metrics based on this evaluation, including precision, recall and fallout.

Unfortunately, their ground truth dataset cannot be directly used for our system. While there is a number of different potential metrics for PUPy, none of them overlap with this context profiling evaluation done in the Gupta et al. paper. The metrics currently used to evaluate PUPy are mainly concerned with the amount of time a module is engaged, the number of authentications successfully bypassed. These are all far more applicable to the stated goal of our system than the context profiling done by Gupta et al.

Due to these differences, a direct quantitative comparison of the two is not feasible. However, a hybrid approach where the theoretical framework underpinning the context profiling system used by Gupta et al. could be tested against our framework. We will outline such a hybrid approach next.

Our Evaluation Method

In order to compare the two theoretical frameworks on metrics better suited to describing the performance of PUPy, we will examine a new evaluation method. We begin by limiting PUPy to just enabling the authentication module, through comparing privacy and unfamiliarity only. These values are drawn from the same data as is used to calculate instantaneous and aggregate familiarity in the Gupta et al. system, and so provides a comparison between the different results obtained from the underlying location and Bluetooth data.

To fairly compare the Gupta et al. system, it is repurposed to enabling and disabling the authentication module, based on rules adapted from their work. This allows the two system to be compared based on the total time each enables the authentication module for. The logical basis for these rules comes from the low and high thresholds from earlier (0.4 and 0.85, respectively). The module can be enabled when the low threshold is met, and anything above the high threshold can put it into high alert/require a PIN unlock.

While this approach does not really compare the full Gupta et al. system to our full proposed system as we have implemented it, it allows us to compare the relative performance of the theoretical framework underpinning the Gupta et al. system versus the theoretical framework underpinning PUPy.

Results

Across every case tested, the authentication module spent significantly more time enabled using the familiarity system compared to our system. A representative example of this can be seen in Figure 7.9. This figure shows the values calculated from the sensor data for PUPy and our custom implementation of the Gupta et al. system, which we will refer to as the custom familiarity system. There are six graphs:

1. The context familiarity over time is shown in purple.
2. The value for $\mathcal{P}(C_n)$ over time is shown in blue.
3. The value for $\mathcal{U}(C_n)$ over time is shown in green.

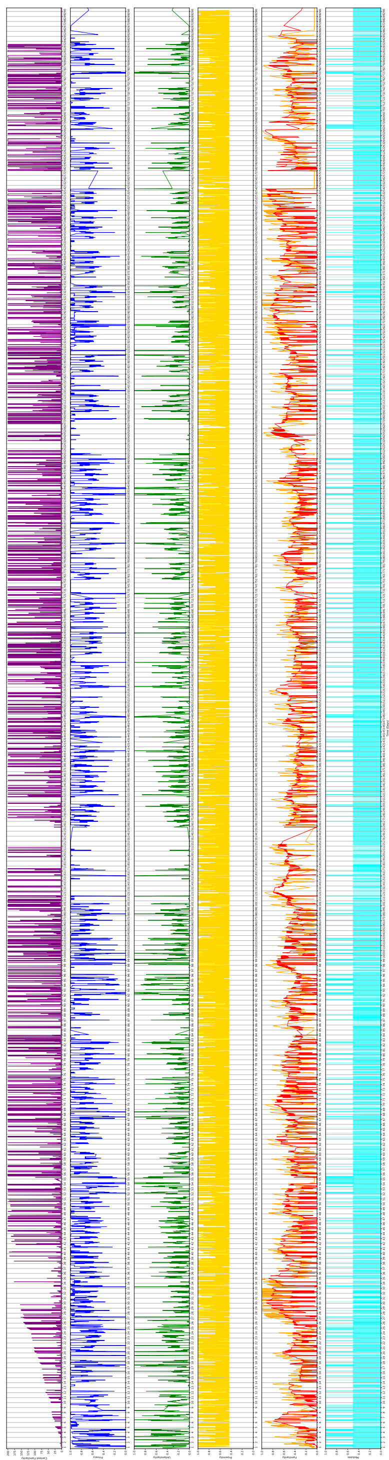


Figure 7.9: Example comparison graph between PUPy and the Gupta et al. system.

4. The value for $\mathcal{D}(C_n)$ over time is shown in yellow.

5. The two values calculated for the familiarity implementation over time are shown. In orange we plot instantaneous familiarity, and in red we plot aggregate familiarity.

6. When the authentication module was enabled under our system and under the familiarity system - PUPy on top, the custom familiarity system on the bottom.

As can be seen, the authentication module is enabled significantly more using the custom familiarity system.

In addition to these graphs, we calculated a wide array of statistics used in the original Gupta et al. paper, for both PUPy and our custom familiarity system. These results can be reviewed in Table 7.5. There are four groups of interest - PUPy, PUPy (Strict), Custom Familiarity and Original Familiarity. We will quickly discuss the differences between each before continuing to the results.

The first two groups are evaluations of our system. In the first, we consider any time the authentication module is enabled, whether it is in low or high alert mode, to be an "unsafe" classification. On the other hand, in the strict version, we consider a classification to be unsafe only if the authentication module was in high alert. This better matches the original familiarity evaluation by Gupta et al., which disregarded yellow classifications during evaluation.

For the familiarity system, the custom familiarity system evaluation is simply the results we get when we compare our custom familiarity system to PUPy using our evaluation framework. So instead of ignoring yellow classifications, we instead look at whether it enabled or disabled the authentication module at all. This is very different from the original evaluation completed by Gupta et al., and should be treated as such. In general, for a fair comparison it is best to compare PUPy (Strict) and Original Familiarity, and to compare PUPy and Custom Familiarity. We will now continue to the results.

The main metric of interest for comparison is that of precision with regard to safe contexts, as defined earlier. One of our primary goals was to take an optimistic approach to context detection, under the assumption that such an approach would result in a significantly improved user experience. If this worked as expected, we would expect to see a higher proportion of safe contexts correctly identified as safe. Indeed, this is exactly what we see in the calculated statistics. Precision regarding safe contexts is 0.978, a roughly 15% increase in correct classifications over the Gupta et al. system. While this seems like a fairly small increase, this means that in safe contexts, only 15% of the explicit authentications performed under the Gupta et al. system would be explicit authentications under our system, a significant improvement in the user experience.

Safe Contexts	Formula	PUPy	PUPy (Strict)
Precision	$\frac{\lvert G_{safe} \cap C_{GREEN} \rvert}{\lvert C_{GREEN} \rvert}$	0.978	0.978
Recall	$\frac{\lvert G_{safe} \cap C_{GREEN} \rvert}{\lvert G_{safe} \rvert}$	0.900	0.964
Unsafe Fallout	$\frac{\lvert G_{unsafe} \cap C_{GREEN} \rvert}{\lvert G_{unsafe} \rvert}$	0.677	0.780
Unclassified Fallout	$\frac{\lvert G_{UC} \cap C_{GREEN} \rvert}{\lvert G_{UC} \rvert}$	0.658	0.798
Unsafe Contexts			
Precision	$\frac{\lvert G_{unsafe} \cap C_{RED} \rvert}{\lvert C_{RED} \rvert}$	0.048	0.070
Recall	$\frac{\lvert G_{unsafe} \cap C_{RED} \rvert}{\lvert G_{unsafe} \rvert}$	0.323	0.220
Safe Fallout	$\frac{\lvert G_{safe} \cap C_{RED} \rvert}{\lvert G_{safe} \rvert}$	0.100	0.036
Unclassified Fallout	$\frac{\lvert G_{UC} \cap C_{RED} \rvert}{\lvert G_{UC} \rvert}$	0.342	0.202

Safe Contexts	Formula	Custom Familiarity	Original Familiarity
Precision	$\frac{\lvert G_{safe} \cap C_{GREEN} \rvert}{\lvert C_{GREEN} \rvert}$	1.000	0.854
Recall	$\frac{\lvert G_{safe} \cap C_{GREEN} \rvert}{\lvert G_{safe} \rvert}$	0.003	0.917
Unsafe Fallout	$\frac{\lvert G_{unsafe} \cap C_{GREEN} \rvert}{\lvert G_{unsafe} \rvert}$	0.000	0.152
Unclassified Fallout	$\frac{\lvert G_{UC} \cap C_{GREEN} \rvert}{\lvert G_{UC} \rvert}$	0.001	0.755
Unsafe Contexts			
Precision	$\frac{\lvert G_{unsafe} \cap C_{RED} \rvert}{\lvert C_{RED} \rvert}$	0.000	0.311
Recall	$\frac{\lvert G_{unsafe} \cap C_{RED} \rvert}{\lvert G_{unsafe} \rvert}$	0.003	0.341
Safe Fallout	$\frac{\lvert G_{safe} \cap C_{RED} \rvert}{\lvert G_{safe} \rvert}$	0.997	0.019
Unclassified Fallout	$\frac{\lvert G_{UC} \cap C_{RED} \rvert}{\lvert G_{UC} \rvert}$	0.999	0.096

Table 7.5: Various statistics on the performance of PUPy and the Gupta et al. system. The Original Familiarity numbers are results directly from Gupta et al. [15]

PUPy	Mean	Standard Deviation	Median
Auth Time	52 days	44 days	46 days
Auth Ratio	0.2	0.1	0.1
Auth Time Overlap	10 days	10 days	7 days
Auth Overlap Ratio	0.06	0.06	0.05
Custom Familiarity System	**Mean**	**Standard Deviation**	**Median**
Auth Time	352 days	150 days	372 days
Auth Ratio	0.99	0.02	1.00
Auth Time Overlap	168 days	90 days	158 days
Auth Overlap Ratio	0.99	0.04	1.00

Table 7.6: Additional comparison statistics between PUPy and our custom familiarity system.

In addition, a larger proportion of unsafe contexts are marked as safe (unsafe fallout is .677 vs .152), and a smaller proportion are marked as unsafe (unsafe context precision is .048 vs .311). This could mean one of two things - either our system is significantly better at detecting when a purported unsafe context is actually safe, or it is significantly worse at detecting actually unsafe contexts. This will depend heavily on how dangerous the context actually was, which is not covered by ground truth data. In general, we would believe we are doing a better job of detecting when a purportedly unsafe context is safe, but there is no way to verify this using this dataset.

Comparing their implementation to ours, the statistics are essentially entirely different. While on the surface, precision of 1 seems to point to the fact that their system performs even better in our evaluation, there are a number of important caveats. Firstly, recall is extremely low, telling us that the system is only correctly identifying a small fraction of the number of safe contexts. For example, on average the familiarity system implementation only marks 29 authentications as safe, out of 11 881 average user authentications. Comparing this to our system, there are on average 9 440 green authentications. What is happening here is that the familiarity system is being extremely cautious in what it marks as safe, to the point that the system almost never unlocks the device.

This is made more apparent when we look at how often each framework has the authentication module enabled. In Table 7.6, we have outlined a number of statistics relating to the amount of time the authentication module was enabled, and when. There are four metrics of interest:

1. Auth Time measures the absolute amount of time the authentication module was enabled across the life of the experiment. In this case, on average the familiarity system enabled the authentication module for almost exactly 300 more days than PUPy. Keeping in mind that in Table 7.2 we show that the average length of the experiment data is roughly 358 days, and we discover that authentication is disabled for only six days total out of the year, on average.

2. Auth Ratio just provides a ratio for the above. In this case, we can see that PUPy enabled the authentication module for only 20% of the life of the experiment, compared to 99% for the familiarity system.

3. Auth Time Overlap looks at the intersection between Auth Time, and the amount of time the user spent in a safe context (Safe Visit Time from Table 7.2). It is not just one over the other, it looks at the overlap between the two - that is, a period of time is only added to this amount if the user is in a safe context, and the authentication module is on. In general, we want this to be as low as possible. Under PUPy, the authentication module is enabled in a safe context for only 9 days, out of the 170 days on average a user spends in safe contexts. The familiarity system is enabled for 168 days.

4. Auth Overlap Ratio is similar to the Auth Ratio, in that it just provides a ratio for the above - in this case, the amount of time the authentication module was enabled, over the total safe visit time.

Overall, our custom implementation of the familiarity system significantly underperforms both our system, and the results obtained by Gupta et al. in their work. We theorize there are two main reasons for this discrepancy:

Implementation Differences It is likely that our implementation of the familiarity system varies from the implementation they use. Their approach uses significantly different metrics, and so attempting to perfectly imitate it is difficult. In addition, there is a total absence of user feedback for our custom familiarity system, which is not the case for the actual Gupta et al. system.

Evaluation Criteria While Gupta et al. classify contexts into three types, C_{GREEN}, C_{YELLOW} and C_{RED} that each correspond to a specific expected safety, they consider only C_{RED} and C_{GREEN}, not C_{YELLOW}, when evaluating. Conversely, we consider any case where the authentication module is enabled as an “unsafe” classification. This means that for our evaluation (both for PUPy and our implementation of the familiarity system), all C_{YELLOW} classifications essentially count as C_{RED}. Our implementation of the familiarity system has a much smaller number of C_{GREEN} classifications, and a significantly larger proportion of C_{YELLOW} classifications.

It is likely these two differences explain the sharply different results we see between our implementation and their reported values. It is best to see our implementation of familiarity as an adapted evaluation of their approach, and not a stand-in. Both sets of metrics give us interesting insight into how the familiarity system compares to our system - one is their best case evaluation, the other their worst case.

Chapter 8

Conclusion and Future Work

Throughout this work, we propose, implement and evaluate a novel context detection framework called PUPy that breaks with existing works in several ways. We aimed to provide a context detection framework with a wider applicability than existing approaches, better accuracy through aggregating multiple data sources, and taking a fundamentally optimistic approach to context detection to improve the user experience.

First, we outlined the theoretical framework underpinning PUPy. We discussed the three values used to describe the context, outlining how they are calculated and what they could be used for. The proposed framework works without necessitating user input, and is extensible by other works, allowing it to adapt to advances in the field. Second, we created an implementation of the theoretical framework on Android, showing how such an application could realistically function. Finally, we devised a method of evaluating the framework based on existing datasets, and analyzed its performance, comparing it to existing works in the field. The framework showed significant promise in improving the user experience, albeit at the cost of easier access for adversaries, a tradeoff common in implicit authentication frameworks.

There are plenty of potential avenues to continue work on this project. In particular, the functionality modules implemented to interface with the context engine are rudimentary, and could be further expanded. Additionally, due to the constraints caused by the COVID-19 pandemic, performing a user study and soliciting feedback was not possible. Modifying the framework to better cater to user needs based on feedback and real-world experience could improve the system further.

The question of access control and security of the system is also one not considered. While some rudimentary attempts are made to secure the system from malicious applications, these protections are not sufficient and should be expanded to ensure malicious applications cannot impede the performance or accuracy of the framework. This may include tightening the requirements for new input modules, and ensuring functionality modules cannot misuse the values generated by the context engine.

Additionally, one large issue mostly set aside in this work is the question of device identification. While it forms an important part of our system, we did not find an adequate solution that balances performance with protecting the privacy of others. Finding a solution or workaround for this approach is crucial for the long term viability of the system, as privacy-preserving measures spread to all major phone manufacturers.

Finally, adding additional input modules to further expand the types of inputs used to calculate the three values will allow for even better system performance. Potential avenues include using voice or face recognition, or implementing a large number of context-specific approaches to improve accuracy in specific contexts. Also in this vein is changing the method through which context familiarity is calculated, from being location-based and done in the context engine, to an aggregated approach similar to the other three estimates.

Overall, we hope that the proposed framework furthers interest in context detection and implicit authentication, and perhaps stimulates more research into optimistic context detection to improve the user experience.

References

[1] Yusuf Albayram, Mohammad Maifi Hasan Khan, Theodore Jensen, and Nhan Nguyen. "...better to use a lock screen than to worry about saving a few seconds of time": Effect of fear appeal in the context of smartphone locking behavior. In *Thirteenth Symposium on Usable Privacy and Security (SOUPS 2017)*, pages 49–63, 2017.

[2] Noah Apthorpe, Danny Yuxing Huang, Dillon Reisman, Arvind Narayanan, and Nick Feamster. Keeping the smart home private with smart(er) IoT traffic shaping. *Privacy Enhancing Technologies*, 2019(3):128–148, 2019.

[3] Noah Apthorpe, Dillon Reisman, and Nick Feamster. Closing the blinds: Four strategies for protecting smart home privacy from network observers. *arXiv preprint arXiv:1705.06809*, page 6, 2017.

[4] Patricia Arias-Cabarcos, Christian Krupitzer, and Christian Becker. A survey on adaptive authentication. *ACM Computing Surveys*, 52(4):1–30, 2019.

[5] Cheng Bo, Lan Zhang, Xiang-Yang Li, Qiuyuan Huang, and Yu Wang. Silentsense: silent user identification via touch and movement behavioral biometrics. In *19th annual international conference on Mobile computing & networking*, pages 187–190, 2013.

[6] Daniel Buschek, Fabian Hartmann, Emanuel Von Zezschwitz, Alexander De Luca, and Florian Alt. Snapapp: Reducing authentication overhead with a time-constrained fast unlock option. In *2016 CHI Conference on Human Factors in Computing Systems*, pages 3736–3747, 2016.

[7] Guillaume Celosia and Mathieu Cunche. Fingerprinting Bluetooth-low-energy devices based on the generic attribute profile. In *2nd International ACM Workshop on Security and Privacy for the Internet-of-Things - IoT S&P'19*, pages 24–31. ACM Press, 2019.

[8] Jagmohan Chauhan, Yining Hu, Suranga Seneviratne, Archan Misra, Aruna Seneviratne, and Youngki Lee. BreathPrint: Breathing acoustics-based user authentication. In *15th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '17, pages 278–291. ACM, 2017.

[9] Guglielmo Cola, Marco Avvenuti, Fabio Musso, and Alessio Vecchio. Gait-based authentication using a wrist-worn device. In *Proceedings of the 13th International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services*, MOBIQUITOUS 2016, pages 208–217. ACM, 2016.

[10] Mauro Conti and Chhagan Lal. Context-based co-presence detection techniques: A survey. *Computers & Security*, 88:101652, 2020.

[11] Serge Egelman, Sakshi Jain, Rebecca S. Portnoff, Kerwell Liao, Sunny Consolvo, and David Wagner. Are you ready to lock? In *2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 750–761. ACM, 2014.

[12] Alain Forget, Sonia Chiasson, and Robert Biddle. Choose your own authentication. In *Proceedings of the 2015 New Security Paradigms Workshop*, NSPW '15, page 1–15, New York, NY, USA, 2015. Association for Computing Machinery.

[13] Mario Frank, Ralf Biedert, Eugene Ma, Ivan Martinovic, and Dawn Song. Touchalytics: On the applicability of touchscreen input as a behavioral biometric for continuous authentication. *IEEE Transactions on Information Forensics and Security*, 8(1):136–148, 2013.

[14] Google. Smart lock. https://get.google.com/smartlock/. Accessed on 2021-02-05.

[15] Aditi Gupta, Markus Miettinen, N. Asokan, and Marcin Nagy. Intuitive security policy configuration in mobile devices using context profiling. In *2012 International Conference on Privacy, Security, Risk and Trust and 2012 International Confernece on Social Computing*, pages 471–480, 2012.

[16] Marian Harbach, Alexander De Luca, Nathan Malkin, and Serge Egelman. Keep on lockin' in the free world: A multi-national comparison of smartphone locking. In *2016 CHI Conference on Human Factors in Computing Systems*, pages 4823–4827. ACM, 2016.

[17] Marian Harbach, Alexander De Luca, and Matthew Smith. It's a hard lock life: A field study of smartphone (un)locking behavior and risk perception. In *10th Symposium On Usable Privacy and Security (SOUPS 2014)*, pages 213–230, 2014.

[18] Eiji Hayashi, Sauvik Das, Shahriyar Amini, Jason Hong, and Ian Oakley. CASA: context-aware scalable authentication. In *Proceedings of the Ninth Symposium on Usable Privacy and Security*, pages 1–10, 2013.

[19] Alexander Heinrich, Milan Stute, and Matthias Hollick. BTLEmap: Nmap for bluetooth low energy. In *13th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, pages 331–333. ACM, 2020.

[20] Daniel Hintze, Matthias Füller, Sebastian Scholz, Rainhard D. Findling, Muhammad Muaaz, Philipp Kapfer, Eckhard Koch, and René Mayrhofer. CORMORANT: Ubiquitous risk-aware multi-modal biometric authentication across mobile devices. *ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 3(3):85:1–85:23, 2019.

[21] Daniel Hintze, Matthias Füller, Sebastian Scholz, Rainhard D. Findling, Muhammad Muaaz, Philipp Kapfer, Wilhelm Nüßer, and René Mayrhofer. CORMORANT: On implementing risk-aware multi-modal biometric cross-device authentication for android. In *17th International Conference on Advances in Mobile Computing & Multimedia*, pages 117–126. ACM, 2019.

[22] Hassan Khan, Aaron Atwater, and Urs Hengartner. A comparative evaluation of implicit authentication schemes. In Angelos Stavrou, Herbert Bos, and Georgios Portokalidis, editors, *Research in Attacks, Intrusions and Defenses*, volume 8688, pages 255–275. Springer International Publishing, 2014.

[23] Hassan Khan, Aaron Atwater, and Urs Hengartner. Itus: an implicit authentication framework for android. In *20th annual international conference on Mobile computing and networking - MobiCom '14*, pages 507–518. ACM Press, 2014.

[24] Hassan Khan, Urs Hengartner, and Daniel Vogel. Targeted mimicry attacks on touch input based implicit authentication schemes. In *14th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '16, pages 387–398. ACM, 2016.

[25] Niko Kiukkonen, Jan Blom, Olivier Dousse, Daniel Gatica-Perez, and Juha Laurila. Towards rich mobile phone datasets: Lausanne data collection campaign. *Proc. ICPS, Berlin*, 68:7, 2010.

[26] Aleksandra Korolova and Vinod Sharma. Cross-app tracking via nearby bluetooth low energy devices. In *Eighth ACM Conference on Data and Application Security and Privacy*, pages 43–52. ACM, 2018.

[27] Juha K Laurila, Daniel Gatica-Perez, Imad Aad, Jan Blom, Olivier Bornet, Trinh Minh Tri Do, Olivier Dousse, Julien Eberle, and Markus Miettinen. From big smartphone data to worldwide research: The mobile data challenge. *Pervasive and Mobile Computing*, 9(6):752–771, 2012.

[28] Wei-Han Lee and Ruby Lee. Implicit sensor-based authentication of smartphone users with smartwatch. *arXiv:1703.03523 [cs]*, 2017.

[29] Tao Li, Yimin Chen, Jingchao Sun, Xiaocong Jin, and Yanchao Zhang. iLock: Immediate and automatic locking of mobile devices against data theft. In *2016 ACM SIGSAC Conference on Computer and Communications Security - CCS'16*, pages 933–944. ACM Press, 2016.

[30] Xinyu Liu, David Wagner, and Serge Egelman. Detecting phone theft using machine learning. In *2018 International Conference on Information Science and System - ICISS '18*, pages 30–36. ACM Press, 2018.

[31] Célestin Matte. *Wi-Fi Tracking: Fingerprinting Attacks and Counter-Measures*. Theses, Université de Lyon, 2017.

[32] Markus Miettinen, Stephan Heuser, Wiebke Kronz, Ahmad-Reza Sadeghi, and N. Asokan. ConXsense - automated context classification for context-aware access control. *9th ACM symposium on Information, computer and communications security - ASIA CCS '14*, pages 293–304, 2014.

[33] Markus Miettinen, Samuel Marchal, Ibbad Hafeez, N. Asokan, Ahmad-Reza Sadeghi, and Sasu Tarkoma. IoT sentinel: Automated device-type identification for security enforcement in IoT. *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pages 2177–2184, 2017.

[34] Oyindamola Oluwatimi and Elisa Bertino. A multi-enterprise containerization approach with an interoperable position-based system. In *Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy*, CODASPY '18, pages 256–266. Association for Computing Machinery, 2018.

[35] Pruthvish Rajput, Manish Chaturvedi, and Vivek Patel. Opportunistic sensing based detection of crowdedness in public transport buses. *Pervasive and Mobile Computing*, 68:101246, 2020.

[36] Arun Ramakrishnan, Jochen Tombal, Davy Preuveneers, and Yolande Berbers. PRISM: Policy-driven risk-based implicit locking for improving the security of mobile

end-user devices. In *13th International Conference on Advances in Mobile Computing and Multimedia - MoMM 2015*, pages 365–374. ACM Press, 2015.

[37] Alessandro Enrico Cesare Redondi, Davide Sanvito, and Matteo Cesana. Passive classification of wi-fi enabled devices. In *19th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, pages 51–58. ACM, 2016.

[38] Oriana Riva, Chuan Qin, Karin Strauss, and Dimitrios Lymberopoulos. Progressive authentication: deciding when to authenticate on mobile phones. In *21st USENIX Security Symposium*. USENIX, 2012.

[39] James Scott, Richard Gass, Jon Crowcroft, Pan Hui, Christophe Diot, and Augustin Chaintreau. *CRAWDAD dataset Cambridge/Haggle (v. 2009-05-29)*. CRAWDAD, 2009. Published: Downloaded from https://crawdad.org/Cambridge/Haggle/20090529.

[40] James Scott, Richard Gass, Jon Crowcroft, Pan Hui, Christophe Diot, and Augustin Chaintreau. CRAWDAD dataset cambridge/haggle (v. 2009-05-29). Downloaded from https://crawdad.org/cambridge/haggle/20090529, May 2009.

[41] Elaine Shi, Yuan Niu, Markus Jakobsson, and Richard Chow. Implicit authentication through learning user behavior. In *International Conference on Information Security*, pages 99–113. Springer, 2010.

[42] B. Shrestha, N. Saxena, H. T. T. Truong, and N. Asokan. Sensor-based proximity detection in the face of active adversaries. *IEEE Transactions on Mobile Computing*, 18(2):444–457, 2019.

[43] Frank Stajano. One user, many hats; and, sometimes, no hat: Towards a secure yet usable PDA. In Bruce Christianson, Bruno Crispo, James A. Malcolm, and Michael Roe, editors, *Security Protocols*, volume 3957, pages 51–64. Springer Berlin Heidelberg, 2006.

[44] Frank Stajano. Pico: No more passwords! In Bruce Christianson, Bruno Crispo, James Malcolm, and Frank Stajano, editors, *Security Protocols XIX*, volume 7114, pages 49–81. Springer Berlin Heidelberg, 2011.

[45] Ahren Studer and Adrian Perrig. Mobile user location-specific encryption (MULE): Using your office as your password. In *Third ACM conference on Wireless network security*, pages 151–162, 2010.

[46] Adam Wójtowicz and Krzysztof Joachimiak. Model for adaptable context-based biometric authentication for mobile devices. *Personal and Ubiquitous Computing*, 20(2):195–207, 2016.

[47] Qiang Xu, Rong Zheng, Walid Saad, and Zhu Han. Device fingerprinting in wireless networks: Challenges and opportunities. *arXiv:1501.01367 [cs]*, 2015.

[48] Lingjing Yu, Bo Luo, Jun Ma, Zhaoyu Zhou, and Qingyun Liu. You are what you broadcast: Identification of mobile and iot devices from (public) wifi. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 55–72, 2020.

[49] Chaoshun Zuo, Haohuang Wen, Zhiqiang Lin, and Yinqian Zhang. Automatic fingerprinting of vulnerable BLE IoT devices with static UUIDs from mobile apps. In *2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 1469–1483. ACM, 2019.