PUPy: A Generalized, Optimistic Context Detection Framework for Implicit Authentication

Matthew Rafuse and Urs Hengartner
Cheriton School of Computer Science
University of Waterloo
{matthew.rafuse, urs.hengartner}@uwaterloo.ca

***Abstract*—Devices like smartphones and laptops employ some form of user authentication to ensure that access to confidential data by the wrong user is avoided. Implicit authentication aims to limit the number of explicit authentications that a user is subjected to by using passive approaches to authenticate the user. Context detection frameworks aim to reduce explicit authentications by disabling explicit authentication entirely when appropriate. Since explicit and implicit authentication are not mutually exclusive, we can also use context detection frameworks to decide whether explicit or implicit authentication should be used when authentication is required.**

We present a novel context detection framework, PUPy, that uses sensed context data to infer and make available three values—privacy, unfamiliarity, and proximity—allowing clients of our framework, like authentication services, to better adapt to different contexts. As opposed to existing work, our context detection framework is based on an optimistic approach to context detection. Our assumption is that the absence of data, like the inability to detect nearby people or devices, can be taken as a sign that a context is safe. Such an optimistic approach may provide less security than a pessimistic approach, but provides a significantly improved user experience due to reducing the number of explicit authentications.

We provide an Android implementation of the framework, including an API that allows other developers to contribute modules to the system. We also conduct a statistical analysis of our framework based on a large real-world dataset. We find that PUPy compares favourably to existing works, permitting a 77.2% reduction in the number of explicit authentications.

I. Introduction

The usage of 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 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. Therefore, all modern smart devices employ some form of user authentication to ensure that access to this confidential data by the wrong person is avoided.

Modern authentication methods are generally knowledge-based or biometric. 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]–[4]. In response to this trend, the field of implicit authentication has arisen. Implicit authentication aims to limit the number of explicit authentications that a user is subjected to by 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 device sensor data.

In contrast, context detection frameworks aim to reduce explicit authentications by disabling explicit authentication entirely when appropriate in contexts deemed "safe" [5], [6]. Since explicit and implicit authentication are not mutually exclusive, there also exist context detection frameworks [7], [8] that use the context around them to decide which of the two authentication approaches to use when authentication is required. This combination of context detection with implicit authentication is the approach taken in this work.

We introduce a new context detection framework called PUPy. PUPy provides context information to apps and services running on a device through a simple interface. PUPy takes in sensor data sensed by the device and condenses it into three values—privacy, unfamiliarity, and proximity—each describing a different aspect of the device's context. (The first letters of these values form the basis for the name of our framework.) Briefly, privacy measures how private the device's context is, based on the number of nearby people detected. Unfamiliarity indicates how unfamiliar the device's context is, based on the number of unfamiliar nearby people detected. Proximity measures how close the device is to its owner.

We demonstrate the flexibility of PUPy by providing three sample modules that take advantage of the three values provided by PUPy to decide whether to enable the module:

1) an authentication module that uses context information to switch between explicit, implicit, and no authentication,
2) a device theft module that decides whether the device is in a context where it may get stolen and therefore raises an alert if the device ends up being away from the owner, and
3) a device loss module that raises an alert when the device is in danger of unintentionally being left behind in a public context.

PUPy provides *optimistic* context detection, which assumes that a context is safe unless PUPy's perception of the context indicates otherwise. In particular, our assumption is that the absence of data, like the inability to detect nearby people and devices, can be taken as a sign that the context is safe. This optimistic approach may provide less security than a pessimistic approach, but provides a significantly improved

user experience due to reducing the number of explicit authentications.

We make the following contributions:

- We introduce the novel theoretical underpinnings of our context detection framework. We improve on existing works through wider applicability and optimistic context detection.
- We present an Android implementation of PUPy and provide three sample modules that take advantage of PUPy.[1]
- We evaluate our framework on a real-world dataset [9] and compare PUPy to existing work. We find that PUPy allows for a 77.2% reduction in explicit user authentications.

II. Goals

The goals for PUPy are 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*. Existing works mention how the point of these systems is to provide a more user-friendly approach to authentication. Despite this, existing approaches all 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. 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.

The second aim of this work is to allow for 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 available 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.

The third aim of this work is to bring a more nuanced and accurate 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. 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, as well as allowing a better description of the sort of context the device is in.

III. Related Work

This section consists of an overview and discussion of a number of existing works and systems that have been devised to achieve similar goals as ours.

[1]https://github.com/mattrafuse/PUPy

A. 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.

We use a slightly broader definition of implicit authentication than existing works. 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.

Implicit authentication forms an important use case for our system, and many existing works hold important insights and concepts for understanding context detection. The term was first coined by Shi et al. [10], who investigate using user behaviour as a way to reduce the number of explicit authentications.

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 [11]. 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. The basic concept of combining many inputs to get a better sense of the context is central to PUPy, though we use it for context detection as opposed to authentication.

We will now outline a number of existing implicit authentication schemes that could eventually be combined with PUPy to further improve the user experience. SilentSense by Bo et al. [12] and Touchalytics by Frank et al. [13] both investigate using user interaction with the device to authenticate the user. Cola et al. [14] investigate using gait-based authentication via a wrist worn device (such as a smartwatch). MULE by Studer et al. [15] uses location as a method of authentication.

B. Context Detection

Before discussing context detection frameworks, first let us define what is meant by context. The device context is a shorthand for the situation the device is in. 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. In this work, we are concerned with using context detection to adapt the behaviour of various applications the user interacts with. We focus primarily on adapting the behaviour of implicit authentication frameworks.

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.

Ramakrishnan et al. [16] 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 the approach taken in our system, where rules are used to govern the actions taken by listening applications when changes in context are taken. However our approach uses developer defined rules to interpret the core values of the framework, providing more flexibility in how the values are used.

Gupta et al. [6] provide an important basis for part of the theoretical framework we build 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. We build upon future work mentioned in their paper—that of marking unclassified contexts as safe, instead of unsafe (i.e., an optimistic approach). Due to the close nature of the two frameworks, further comparisons between PUPy and the system proposed by Gupta et al. will be made later on.

Another framework to build off the concepts developed by Gupta et al. is ConXSense, developed by Miettinen et al. [17]. ConXSense controls the use of the 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. They categorize contexts (location and WiFi based) into private, work or public contexts, either safe or unsafe, as opposed to the continuous approach used in PUPy.

C. Implicit Authentication and Context Detection

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

Wójtowicz and Joachimiak [8] investigate combining context detection and biometric authentication in order to provide a more accurate authentication system. 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. 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.

The CORMORANT framework devised by Hintze et al. [7] is closely related structurally to our system. The modular approach devised 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. CORMORANT focuses on using cross-device authentication to achieve their goals. They aim to provide continuous authentication across all devices by providing different methods of authentication across many devices, and sharing authentication results between them. They determine what device to use for authentication based on the user's context. While we are also interested in different methods of authentication through our functionality modules, our focus is 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 more advanced approach to risk estimation, using realtime data of the context to determine risk.

IV. System Design

In this section, we will discuss the theoretical design of the framework underlying PUPy. We will first outline the threat model we operate under, followed by a discussion of the theoretical framework.

A. Adversary Model

Our adversary model does not include protection from 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 [18], [19]. 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. We assume that Bluetooth is always enabled.

For authentication, the adversary takes the form of a person unknown to the owner who can physically access the device. The adversary can be malicious, honest-but-curious or clueless, though we focus mainly on the malicious case. Their aim is to unlock the device. The goal of the system 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 Section VI.

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). 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).

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

B. Theoretical Framework

This section will outline what aspects of the device context we track in PUPy, and how we track these aspects based on device sensor 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 use C to represent a particular real-world context, and C_n to represent the nth occurrence of that real-world context. The context consists of a feature vector, where each value is a value between 0 and 1, representing some part of the context. This feature vector consists of three values:

1) **Privacy.** The measure of how private the context is, based on the number of nearby people detected. A value of 0 is very public, while 1 is very private. We denote the privacy of C_n by $\mathcal{P}(C_n)$.
2) **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 $\mathcal{U}(C_n)$.
3) **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 $\mathcal{D}(C_n)$.

Privacy allows modules to adapt their behaviour based on the privacy of a given context. 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. In addition to the standalone case, the combination of privacy and unfamiliarity can provide helpful insights into the current context. In general, using the values together vastly improves usefulness and the amount of information one can gather about the context.

Unfamiliarity allows modules to adapt their behaviour based on the number of unfamiliar people 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 people nearby is a more threatening context than one without any unfamiliar people. This translates to helping us adapt our behaviour based on the threat level of the context. In more threatening contexts, 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 people 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 people with a fairly public setting may prompt the use of stronger authentication methods, or activation of a device theft module.

Proximity allows access to the proximity of the device, so we can adapt our behaviour. 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.

There are 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. The second step is the aggregation step, which is completed by the context engine. The context engine takes the estimates provided by the range of input modules installed and aggregates them into the three values above. In the third step, functionality modules will take the values as calculated by the context engine and use them to adapt to the current context.

1) Privacy: We will now examine the procedure for calculating privacy based on input module data. The set P denotes 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:

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

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

$$\mathcal{P}(C_n) = 1 - \frac{\alpha_{\mathcal{C}}^{1 - \frac{1}{L(C_n)}}}{\alpha_{\mathcal{C}}} = 1 - \alpha_{\mathcal{C}}^{-\frac{1}{L(C_n)}} \tag{2}$$

$\alpha_{\mathcal{C}}$ controls how quickly the value decays. $\alpha_{\mathcal{C}}$ 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 IV-B4. $W_{\mathcal{P}}$ 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.

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 [6].

We first define device familiarity for a device d as follows:

$$F_d(d, C_n) = \alpha_F * occ(d, C_n) + (1 - \alpha_F) * F_d(d, C_{n-1}) \quad (3)$$

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. 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:

$$\mathcal{F}(C_n) = \frac{1}{|D_{C_n}|} \sum_{d \in D_{C_n}} F_d(d, C_n) \quad (4)$$

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:

$$U(C_n) = L(C_n) * (1 - \mathcal{F}(C_n)) \quad (5)$$

This equation takes the number of people detected ($L(C_n)$, from Equation 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. Our assumption is that each nearby person carries a mobile device, like a smartphone, with them.

We then use the similar conversion as in Equation 2 to convert this unbounded number to a value between 0 and 1:

$$\mathcal{U}(C_n) = \frac{\alpha_{\mathcal{C}}^{1 - \frac{1}{U(C_n)}}}{\alpha_{\mathcal{C}}} = 1 - \alpha_{\mathcal{C}}^{-\frac{1}{U(C_n)}} \quad (6)$$

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. 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 to *low* privacy. In contrast, as unfamiliarity changes, we move from *low* unfamiliarity to *high* unfamiliarity.

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

$$d(C_n) = \sum_{i \in 1 \ldots |R|} \frac{r_i(C_n) w_i}{\sum W_{\mathcal{R}}} \quad (7)$$

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

$$\mathcal{D}(C_n) = \begin{cases} 1 & d(C_n) \leq \alpha_d \\ 1 - \alpha_{\mathcal{D}}^{-\frac{1}{d(C_n) - \alpha_d}} & d(C_n) > \alpha_d \end{cases} \quad (8)$$

This allows the relationship to be strong while the device is in close proximity to the owner via α_d, and we can use $\alpha_{\mathcal{D}}$ to decide how quickly the relationship decays as distance increases. 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.

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 $\alpha_{\mathcal{C}}$ 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 types through functionality modules with a small amount of work.

There is the potential to use context familiarity 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

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. 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. The value of context familiarity for the context C at the nth visit could be calculated as:

$$\alpha_{\mathcal{C}} = 2 + n \quad (9)$$

The use of $\alpha_{\mathcal{C}}$ in Equations 2 and 6 causes pathological behaviour when it is set to 1, and below 3 it decays too rapidly to be useful. 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, which is undesirable. In order to maintain functionality in frequently visited contexts, $\alpha_{\mathcal{C}}$ will be capped at 200.

5) Functionality Modules: In order for these values to actually interface with functionality modules, we will use a set of rules provided by the developer to decide when the functionality modules should be notified. For the purposes of evaluation and experimentation, three functionality modules have been implemented as outlined in Section V. An example set of rules governing them is outlined in Table II.

V. Implementation

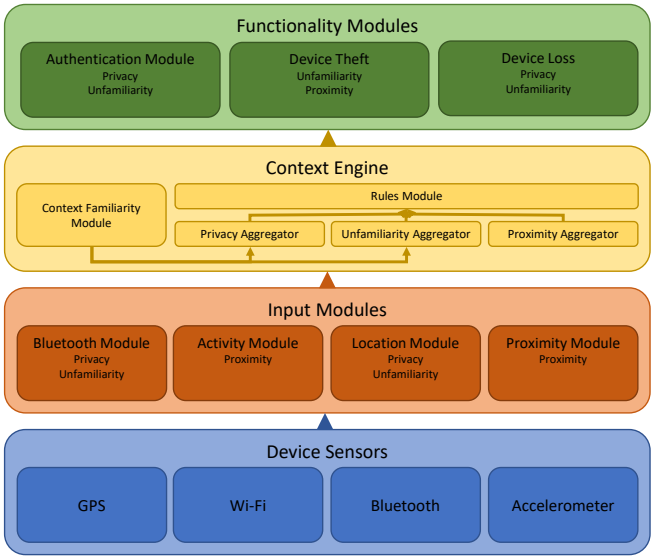

Fig. 1. System diagram of PUPy's Android implementation.

In this section, we will describe the Android implementation of PUPy, including the implemented structure and the specific purpose of each module. Figure 1 shows a general outline 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 that aggregates the estimates and tracks the familiarity of a given context, and the functionality modules that act upon the resulting values.

The structure of the system leans heavily on CORMORANT [7], [20], 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 rewritten, or just removed wholesale. At a high level, three large changes were made. First, we removed the cross-device aspect of CORMORANT given this work focuses on authentication for a single device. Second, we changed the calculations done in the core of the CORMORANT system, replacing CORMORANT's confidence/risk values with the values outlined in Section IV-B. Finally, we built into the system a location-based context identification system, to track the current location-based context the owner is in.

In Section V-A, we will outline the currently implemented input modules. In Section V-B, 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 V-C, we will discuss the current implementation of the functionality modules.

A. 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 emphasize that all our implemented input modules are proofs of concept whose main purpose is to allow us to test whether our framework works correctly on Android; they are not meant for testing the efficacy of the framework.

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 1 and 5 from Section IV-B. This means there are two main purposes of the application, estimating the total number of people and the total number of unfamiliar 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). In order to track the number of unfamiliar people $U(C_n)$, we must calculate $\mathcal{F}(C_n)$ from Equation 4. We do this using the same basic approach as Gupta et al. [6], by storing previous Bluetooth 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.

Due to growing interest in privacy among device users, manufacturers are more frequently implementing measures that make tracking devices across time and space more difficult. This makes identifying (un)familiar devices more challenging. It also poses an 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.

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

TABLE I
THE MAPPING FOR ACTIVITY TO DISTANCE ESTIMATE USED BY THE ACTIVITY APPLICATION.

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

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 7.

In Table I, 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 proof of concept application is only to give us an estimate to test with.

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. First, the user sets locations they deem sufficiently safe. Second, the application calculates the distance between all defined safe locations and the user's current location, 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 to the context engine.

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 $r_i(C_n)$ to the context engine.

B. Context Engine

The context engine combines the estimates obtained from the input modules, using the processes outlined in Section IV-B. It then provides the aggregated values to the functionality modules. The context engine forms the core of PUPy. Alongside the main modules shown in Figure 1, there is a fair amount of supporting code. Overall, the structure of the code can be broken down into five sections, which we will discuss next.

1) User Interface: The user interface is mainly used to list active input modules, and give the user a way of seeing the estimate provided by each module. The user interface allows the user to access the configuration activities (if they exist) for these input modules. It is also the method through which the context engine asks for its required permissions.

2) 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. When a new module is registered, it provides to the plugin manager information that allows the context engine to interface with the new module. There may be multiple entries in the plugin manager for each input module installed, if the module reports estimates for multiple values.

3) Aggregator Modules: The aggregator modules aggregate the estimates from the input modules.

The privacy aggregator takes all estimates $p_i(C_n)$ from the privacy input modules and combines them as per Equation 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 2.

The unfamiliarity aggregator is implemented in a slightly different way than the theoretical basis outlined in Section IV-B2. Instead of reusing the value $L(C_n)$ and multiplying it by $(1-\mathcal{F}(C_n))$ as shown in Equation 5, we instead aggregate individual estimates $u_i(C_n) \in V$ and use them to calculate $U(C_n)$:

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

$W_{\mathcal{U}}$ 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 V-A1. This method allows for alternative means of estimating the number of unfamiliar people, making the context engine more adaptable.

The proximity aggregator takes all estimates $r_i(C_n)$ for the distance between the device and user and aggregates them into the value $d(C_n)$ as in Equation 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 are used as in Equation 8 to obtain the calculated value $\mathcal{D}(C_n)$.

4) 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 privacy and unfamiliarity aggregators.

TABLE II
EXAMPLE RULES TO GOVERN THE THREE IMPLEMENTED MODULES.

Rule	Action
Authentication Module	
$\mathcal{P}(C_n) - \mathcal{U}(C_n) < .1$	Enable Authentication Module
$\mathcal{P}(C_n) - \mathcal{U}(C_n) \geq .1$	Disable Authentication Module
$\mathcal{P}(C_n) - \mathcal{U}(C_n) < -.4$	Start High Alert
$\mathcal{P}(C_n) - \mathcal{U}(C_n) \geq -.4$	End High Alert
Device Theft Module	
$\mathcal{U}(C_n) > .5 \wedge \mathcal{D}(C_n) < .9$	Enable Device Theft Module
$\mathcal{U}(C_n) \leq .1 \vee \mathcal{D}(C_n) \geq 1$	Disable Device Theft Module
$\mathcal{U}(C_n) > .5 \wedge \mathcal{D}(C_n) \leq .2$	Start Audible Alarm
Device Loss Module	
$\mathcal{D}(C_n) \leq .75$	Enable Device Loss Module
$\mathcal{D}(C_n) > .85$	Disable Device Loss Module
$\mathcal{D}(C_n) < .3 \wedge \mathcal{P}(C_n) < .5$	Send Lost Device Notification

5) Rule Module: The rule 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 with the new values of $\mathcal{P}(C_n)$, $\mathcal{U}(C_n)$ and $\mathcal{D}(C_n)$ to all registered functionality modules, so they may check these values against the rules they set, and adapt accordingly.

C. 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. We will examine each of these functionality modules in this section. All three functionality modules are basic implementations, as they are mainly a proof of concept for feasibility of the context engine.

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 actual authentication, relying only on the context data calculated by PUPy. Instead, the authentication module performs two actions, following the example rules outlined in Table II. It 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 $\mathcal{U}(C_n)$, and sends a notification declaring the start and end of high alert mode. If the module is disengaged, this corresponds to no authentication whatsoever. If the module is engaged, we would use implicit authentication. Finally, if the module is in high alert, the authentication module would require explicit authentication.

The presented sample rules for the authentication module are based on the logic that when enabling and disabling the module, the presence of unfamiliar people should be the primary factor in the decision. However, in more private locations, the number of unfamiliar people can be higher. When moving the authentication module into high alert (i.e. requiring explicit authentication), we care if a large fraction of people in the current context are unfamiliar.

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 $\mathcal{D}(C_n)$. 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 II.

The presented sample rules for the device theft module are based on the logic that if there are unfamiliar people around, and the device is not on person, we should enable the device theft module. When the context changes to a safer one, it can be safely disabled. When the distance between the owner and device is too large, it is likely the device is being stolen, and an audible alarm should sound.

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. 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. The device loss 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 II.

The presented sample rules for the device loss module are based on the logic that the device loss module should be enabled when the user is fairly far away, and if it was left behind in a public area, it should attempt to notify the owner.

VI. EVALUATION

In order to evaluate the efficacy of PUPy, we take an approach shared by previous works [6], [7], using an existing dataset to evaluate our system.

A. Dataset

For running evaluations of the system, we rely on the MDC Dataset. The MDC Dataset [9] is a large dataset based on the Lausanne Data Collection Campaign [21]. 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 useful when calculating values for context familiarity, privacy and unfamiliarity. Inferences on user behaviour based on acceleration data are also available, which we use to estimate proximity.

The vast quantities of data collected allows 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 [6], [7]. It is also used by Gupta et al. to test their familiarity system, a work whose contributions formed an important part of the

TABLE III
PER-PARTICIPANT STATISTICS ON THE MDC DATASET.

	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	11,882	8,523	10,025

theoretical framework of PUPy. It also used a viral marketing approach to recruit participants. This approach ensured that many of the participants regularly interacted with each other, meaning that we would have a solid basis for learning what devices are familiar.

In order to gain a better understanding of the sort of data the dataset contains and how much there is, we can look to Table III. This table shows some statistics on the average timespan of each user in the dataset. 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. A subset of the users in the dataset labelled their current context periodically throughout the experiment. Similar to Gupta et al, we use these labels to extrapolate the ground truth data about the safety of a context for our evaluation later 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. 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. [3], which shows that on average, users use their device for 70 seconds, with a standard deviation of 240 seconds. We assume the user must reauthenticate if the device has not been used for more than one standard deviation from the average. 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.

B. 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.

PUPy can benefit in two ways from having a large amount of time and data to learn from when determining the context. First, there is unfamiliarity, which learns what devices are familiar and which are not over time. Second, there is 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, starting with the success ratio, which is defined as follows:

$$\frac{A_{DISABLED}}{A_{TOTAL}} \times 100$$

where $A_{DISABLED}$ is the number of authentications when the authentication module was disabled so far, and A_{TOTAL} is the total number of authentications so far. Figure 2 plots the cumulative success ratio over time. 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%.

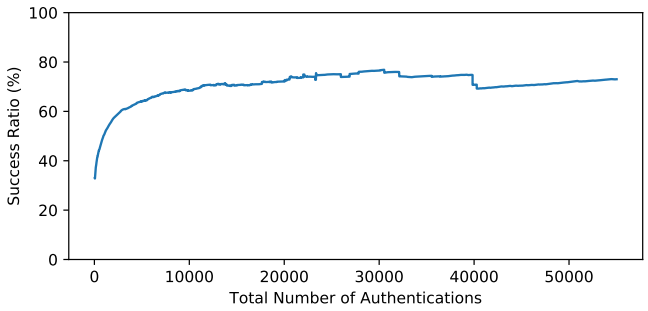

Fig. 2. The cumulative success rate of the system in disabling authentications compared to the total number of authentications.

TABLE IV
STATISTICS OUTLINING THE PERFORMANCE OF PUPY.

	Mean	Standard Deviation	Median
Success Ratio	0.772	0.09	0.80
Total Highlights	24.70	43.59	8.00
Auth Time	52 days	44 days	46 days
Auth Ratio	0.15	0.11	0.13
Theft Time	29 days	25 days	23 days
Theft Ratio	0.17	0.20	0.14
Loss Time	69 days	51 days	56 days
Loss Ratio	0.38	0.26	0.35
High $\alpha_{\mathcal{C}}$	27.56	15.91	25.00
Medium $\alpha_{\mathcal{C}}$	43.50	29.30	36.00

Next, we will break down some overall performance statistics of how PUPy performs. First, we present the success ratio, as defined above. As shown in Table IV, this ratio was 0.772 on average, corresponding to a 77.2% reduction in explicit authentication requests. Despite PUPy being mainly a context detection framework and not directly authenticating the user, PUPy compares nicely to a number of existing authentication-focused works. Progressive authentication [11] saw a reduction of 42%. CORMORANT [7] 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 [17] achieves similar performance to us, relaxing security in roughly 70% of contexts.

The next statistic of interest is the total number of highlighted contexts that the user encounters on average. Highlighted contexts are contexts in which there is a large gap between $\mathcal{P}(C_n)$ and $1 - \mathcal{U}(C_n)$. If a context has entirely unfamiliar devices, $\mathcal{P}(C_n)$ must be equal to $1 - \mathcal{U}(C_n)$. 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, $\mathcal{F}(C_n)$ 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. Therefore, we keep an eye out in our calculations for such situations. 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 in these contexts.

Looking at the amount of time each of the three modules were enabled and their corresponding ratios, we see that the device loss module was by far the most active module, activated roughly 38% 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 15% and 17% of the time, respectively.

The final 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 IV-B. 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. On average, 27 locations are marked as high familiarity, and 43 contexts are marked as medium familiarity.

VII. Conclusion and Future Work

We proposed, implemented and evaluated a novel context detection framework called PUPy that breaks with existing works in several ways: wider applicability, better accuracy through aggregating multiple data sources, and taking a fundamentally optimistic approach to context detection to improve the user experience. We outlined the theoretical framework underlying PUPy and created an implementation of the framework on Android. Finally, we evaluated the framework based on an existing dataset. The framework showed significant promise in improving the user experience by significantly reducing the number of explicit authentications.

In terms of future work, the functionality modules are currently rudimentary and could be expanded. Due to the COVID-19 pandemic, performing a user study and soliciting feedback was not possible.

Acknowledgments

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.

References

[1] Y. Albayram, M. M. H. Khan, T. Jensen, and N. 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 *13th Symposium on Usable Privacy and Security (SOUPS 2017)*, 2017, pp. 49–63.

[2] S. Egelman, S. Jain, R. S. Portnoff, K. Liao, S. Consolvo, and D. Wagner, "Are you ready to lock? Understanding user motivations for smartphone locking behaviors," in *21st ACM Conference on Computer and Communications Security (CCS 2014)*, 2014, pp. 750–761.

[3] M. Harbach, A. De Luca, N. Malkin, and S. Egelman, "Keep on lockin' in the free world: A multi-national comparison of smartphone locking," in *ACM Conference on Human Factors in Computing Systems (CHI 2016)*, 2016, pp. 4823–4827.

[4] M. Harbach, A. D. Luca, and M. 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)*, 2014, pp. 213–230.

[5] M. Conti and C. Lal, "Context-based co-presence detection techniques: A survey," *Computers & Security*, vol. 88, pp. 0167–4048, 2020.

[6] A. Gupta, M. Miettinen, N. Asokan, and M. Nagy, "Intuitive security policy configuration in mobile devices using context profiling," in *International Conference on Privacy, Security, Risk and Trust*, 2012, pp. 471–480.

[7] D. Hintze, M. Füller, S. Scholz, R. D. Findling, M. Muaaz, P. Kapfer, E. Koch, and R. Mayrhofer, "CORMORANT: Ubiquitous risk-aware multi-modal biometric authentication across mobile devices," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies (IMWUT 2017)*, vol. 3, no. 3, pp. 85:1–85:23, 2019.

[8] A. Wójtowicz and K. Joachimiak, "Model for adaptable context-based biometric authentication for mobile devices," *Personal and Ubiquitous Computing*, vol. 20, no. 2, pp. 195–207, 2016.

[9] J. K. Laurila, D. Gatica-Perez, I. Aad, J. Blom, O. Bornet, T. M. T. Do, O. Dousse, J. Eberle, and M. Miettinen, "From big smartphone data to worldwide research: The mobile data challenge," *Pervasive and Mobile Computing*, vol. 9, no. 6, pp. 752–771, 2012.

[10] E. Shi, Y. Niu, M. Jakobsson, and R. Chow, "Implicit authentication through learning user behavior," in *International Conference on Information Security (ISC 2010)*, 2010, pp. 99–113.

[11] O. Riva, C. Qin, K. Strauss, and D. Lymberopoulos, "Progressive authentication: deciding when to authenticate on mobile phones," in *21st USENIX Security Symposium*, 2012.

[12] C. Bo, L. Zhang, X.-Y. Li, Q. Huang, and Y. Wang, "Silentsense: Silent user identification via touch and movement behavioral biometrics," in *19th Annual International Conference on Mobile Computing & Networking (MobiCom 2013)*, 2013, pp. 187–190.

[13] M. Frank, R. Biedert, E. Ma, I. Martinovic, and D. Song, "Touchalytics: On the applicability of touchscreen input as a behavioral biometric for continuous authentication," *IEEE Transactions on Information Forensics and Security*, vol. 8, no. 1, pp. 136–148, 2013.

[14] G. Cola, M. Avvenuti, F. Musso, and A. Vecchio, "Gait-based authentication using a wrist-worn device," in *13th International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services (MOBIQUITOUS 2016)*, 2016, pp. 208–217.

[15] A. Studer and A. Perrig, "Mobile user location-specific encryption (MULE): Using your office as your password," in *3rd ACM Conference on Wireless Network Security (WiSec 2010)*, 2010, pp. 151–162.

[16] A. Ramakrishnan, J. Tombal, D. Preuveneers, and Y. 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)*, 2015, pp. 365–374.

[17] M. Miettinen, S. Heuser, W. Kronz, A.-R. Sadeghi, and N. Asokan, "ConXsense - Automated context classification for context-aware access control," in *9th ACM Symposium on Information, Computer and Communications Security (ASIACCS 2014)*, 2014, pp. 293–304.

[18] 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*, vol. 18, no. 2, pp. 444–457, 2019.

[19] H. Khan, U. Hengartner, and D. Vogel, "Targeted mimicry attacks on touch input based implicit authentication schemes," in *14th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys 2016)*, 2016, pp. 387–398.

[20] D. Hintze, M. Füller, S. Scholz, R. D. Findling, M. Muaaz, P. Kapfer, W. Nüßer, and R. Mayrhofer, "CORMORANT: On implementing risk-aware multi-modal biometric cross-device authentication for Android," in *17th International Conference on Advances in Mobile Computing & Multimedia (MoMM 2019)*, 2019, pp. 117–126.

[21] N. Kiukkonen, J. Blom, O. Dousse, D. Gatica-Perez, and J. Laurila, "Towards rich mobile phone datasets: Lausanne data collection campaign," *Proc. ICPS, Berlin*, vol. 68, p. 7, 2010.