Security and Privacy in Mobile Devices: Novel Attacks and Countermeasures

by

Yimin Chen

A Dissertation Presented in Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

Approved October 2018 by the
Graduate Supervisory Committee:

Yanchao Zhang, Chair
Martin Reisslein
Lei Ying
Junshan Zhang

ARIZONA STATE UNIVERSITY

December 2018

ABSTRACT

Mobile devices have penetrated into every aspect of modern world. For one thing, they are becoming ubiquitous in daily life. For the other thing, they are storing more and more data, including sensitive data. Therefore, security and privacy of mobile devices are indispensable. This dissertation consists of five parts: two authentication schemes, two attacks, and one countermeasure related to security and privacy of mobile devices.

Specifically, in Chapter 1, I give an overview the challenges and existing solutions in these areas. In Chapter 2, a novel authentication scheme is presented, which is based on a user's tapping or sliding on the touchscreen of a mobile device. In Chapter 3, I focus on mobile app fingerprinting and propose a method based on analyzing the power profiles of targeted mobile devices. In Chapter 4, I mainly explore a novel liveness detection method for face authentication on mobile devices. In Chapter 5, I investigate a novel keystroke inference attack on mobile devices based on user eye movements. In Chapter 6, a novel authentication scheme is proposed, based on detecting a user's finger gesture through acoustic sensing. In Chapter 7, I discuss the future work.

To my parents, my sister, and my brother.

ACKNOWLEDGMENTS

I owe my gratitude to several people who have advised, supported, or inspired me during the course of the work.

First, I want to truly thank my advisor Dr. Yanchao Zhang, who through his immense patience and forbearance has shown me an exciting world of research. You have been a constant source and guide of knowledge for the past five years. Without your encouragement and help, I would not have completed this work. Thank you.

I would also like to acknowledge Dr. Junshan Zhang, Dr. Lei Ying, and Dr. Martin Reisslein, who have supported me in many different ways over these years. I greatly appreciate Dr. Junshan Zhang, Dr. Lei Ying, and Dr. Martin Reisslein for serving on my dissertation committee and providing me with guidance from time to time about my dissertation.

I have received the help and support from a great number of people, including, but not limited to, my colleagues Dr. Rui Zhang, Dr. Jingchao Sun, Dr. Jinxue Zhang, Dr. Xiaocong Jin, Tao Li, Dianqi Han, Dr. Junwei Zhang, Dr. Xin Yao, Ang Li, Yan Zhang, and Lili Zhang, who I have closely worked with over the years.

My research work and the writing of this dissertation could not have been completed without the enormous support of my family and friends. I thank my family members for being a constant source of support and encouragement. My deepest gratitude also goes to my friends who encourage and believe in me over the time, including, but not limited to, Dr. Siyuan Wei, Dr. Yanmin Zhang, and Dr. Yinnan Chen.

I also gratefully acknowledge the financial support I received from the National Science Foundation through grant CNS-1117462, CNS-1320906, CNS-1421999, and CNS-1422301, CNS-1514381, CNS-1619251, CNS-1651954 (CAREER), CNS-1700032, and

CNS-1700039 and from the US Army Research Office through grant W911NF-15-1-0328.

TABLE OF CONTENTS

CHAPTER Page

CHAPTER Page

CHAPTER Page

CHAPTER Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

Figure Page

Chapter 1

INTRODUCTION

1.1 Mobile Authentication

Mobile devices such as smartphones, tablets, and eReaders have penetrated into everyday life. According to a recent Cisco report [4], the number of mobile-connected devices would exceed the world population in 2014 and hit 10 billion in 2018. More and more mobile devices have a multi-touch screen that can simultaneously detect more than one point of contact. People are using mobile devices in every aspect of life, including voice/video communications, Internet browsing, web transactions, online banking, business operations, route planning and navigation, personal health and wellbeing, *etc.*

There is urgent need for mobile authentication techniques to prevent illegitimate access to mobile devices. On the one hand, people are storing increasingly more private information on multi-touch mobile devices. On the other hand, many users do not or often forget to log out of personal accounts such as web accounts, email accounts, and various on-device application accounts. Therefore, illegitimate access to a mobile device may seriously jeopardize the legitimate user's information and communication security. Mobile authentication techniques allow the legitimate user to unlock a mobile device and also deny illegitimate access. This is commonly accomplished by letting a user input a password only the legitimate user knows.

Sound mobile authentication techniques for multi-touch mobile devices should be both secure and usable. The security requirement demands strong resilience to notably three attacks. The first is the random-guessing attack in which an attacker

tries to guess or emulate the password the legitimate user uses to unlock a mobile device; the second is the shoulder-surfing attack in which malicious bystanders try to observe the password of the legitimate user [5]; and the last is the smudge attack in which an attacker tries to infer the password based on the finger smudges the legitimate user left on the screen [6]. In contrast, the usability requirement has two implications. First, the authentication technique should be very easy to use by the legitimate user. Second, it should be highly accessible to visually impaired people with visual impairment. The second aspect is often neglected in the literature, despite that there are 285 million people worldwhile [7] and 21.5 million US adults aged 18 and older with visual impairment [8].

Existing authentication techniques for multi-touch mobile devices can be broadly classified into three categories.

Something-You-Know. This category of techniques require a user to input the correct password on the device screen to be admitted. The legitimate user presets the correct password, which can be an alphanumeric password or a gesture/picture password used in Android, iOS, and Windows 8. This category of techniques have some well-known drawbacks. Firstly, such techniques are quite vulnerable to shoulder-surfing attacks in public places. Secondly, these techniques require users to input at specific positions on a touch screen. This requirement may be a great frustration for people with fat fingers, and it may also open the door to smudge attacks. Finally, these techniques are not accessible to people with visual impairment.

Something-You-Have. This category of techniques require auxiliary hardware device only the legitimate user should possess. Examples include tMagkey/Mickey [9] and signet rings [10]. Although resilient to shoulder-surfing attacks, these techniques require additional hardware components to be specifically built. Also, such techniques authenticate a hardware component rather than a user to a mobile device.

Someone-You-Are. This category of techniques require physiological or behavioral biometrics of mobile users. Physiological biometrics relates to a person's physical features such as fingerprints, which are susceptible to well-known spoofing mechanisms. For example, the fingerprint-based Touch ID security system has been broken shortly after iPhone 5S was launched [11]. In contrast, behavioral biometrics relates to a user's behavioral patterns such as location traces [12, 13], gaits [14, 15], and touch dynamics [16, 17, 18]. These techniques are best suitable as secondary authentication mechanisms supplementing the primary password-based authentication mechanism, as they may be vulnerable to the adversary (*e.g.*, a close friend) familiar with the target's behavioral patterns.

1.2 Mobile App Fingerprinting

The popularity of mobile devices has driven the fast development of attractive mobile apps, which in turn further accelerates the ubiquity of mobile devices. For example, a recent Nielsen analysis [19] found that U.S. smartphone users accessed 26.7 apps on average and spent 37 hours and 28 minutes per month in Q4 2014. Mobile app fingerprinting, by which one can know *the apps a user has installed and how s/he uses these apps*, can be used for user profiling and inferring sensitive information about the user such as hobbies, health conditions, locations, habits, and life styles. The disclosure of such sensitive information endangers user privacy.

How could the app usage information be collected? Mobile app stores such as Google Play Store and Apple App Store are obviously in the best position to collect such sensitive information. Such app stores are fortunately operated by trustworthy business giants and not a major threat against user privacy. Mobile malware can play the main role in collecting sensitive app usage information. According to Alcatel-Lucent's Motive Security Labs [20], the malware infection rate on mobile devices

rose to 0.75% in Q2 2015 from 0.68% in December 2014, and there were as many Android devices infected with malware as Windows laptops in the second half of 2014 alone. Mobile malware can be embedded into apps purposefully by malicious app developers or through hacked app development tools. An instance for the later case is the XcodeGhost malware found in September 2015 and from a malicious version of Xcode, Apple's official tool for developing iOS and OS X apps. Another instance is the backdoor in Baidu Android SDK which was found in November 2015 and may have put 100 million Android devices at risk. Besides of malware-infected apps, an enterprise app may collect its employees' app usage information without prior consent.

Significant effort has been made to infer sensitive user information on mobile devices. For example, internal sensors on a mobile device have been exploited to infer user inputs on the touchscreen [21, 22, 23, 24, 25] and user locations [26, 27]. Android public resources that can be accessed without requiring user permission have also been used to infer sensitive user information in [28, 29, 30, 31]. None of these schemes aims at app usage information. Existing work on mobile app fingerprinting mostly relies on traffic analysis [32, 33, 34, 35, 36, 37], all of which require the attacker to obtain the entire web traffic from the victim's device. As a result, the attacker needs to either be in the vicinity of the victim or even compromise network service providers to obtain the traffic data, which limits their applicability. In addition, these traffic-based methods do not work well with apps which generate only a limited amount of traffic or stay offline for most of the time.

1.3 Liveness Detection for Mobile Face Authentication

Protecting mobile devices from unauthorized access is becoming more than indispensable in these days. In particular, mobile devices such as smartphones and tablets

are pervasive in personal life and business world. They are storing increasingly more highly sensitive information such as personal contacts and multimedia information, usernames and passwords, emails, browsing histories, business secrets, and health conditions. At the same time, mobile devices may be lost, stolen, or hacked. For example, 70 million smartphones are lost every year, with only 7% recovered, and 4.3% of company-issued smartphones are lost/stolen every year [38]. In addition, the malware infection rate on mobile devices rose to 0.75% in Q2 2015 from 0.68% in December 2014, and there were as many Android devices infected with malware as Windows laptops in the second half of 2014 alone [20].

Mobile authentication is widely adopted to protect mobile devices from unauthorized access and has two forms. First, a user is authenticated to unlock a device. Second, many mobile apps such as bank apps and password managers authenticate the user before s/he can use these apps. Mobile authentication traditionally follow a password approach based on PINs, alphanumeric passwords, or pattern locks. As functionalities of mobile devices keep improving, people have recently developed more secure and/or usable mobile authentication techniques based on behavioral biometrics such as inputting habits [18, 39, 40, 41] and physiological biometrics such as fingerprints and deauthentication techniques based on proximity [42].

Working on this line, we focus on improving the security of face authentication on mobile devices. As the name suggests, face authentication verifies or identifies a person by validating selected facial features from a digital image or a video frame. The facial features of a person are quite unique and difficult to forge. So face authentication has been very popular in various traditional application scenarios, e.g., gate and automated border control systems. It has also been introduced into mobile devices as a strong authentication method since Android 4.0, as well as many apps such as BioID and MobileID. Although we aim at face authentication on mobile devices,

our work can be generalized to other scenarios involving face authentication without much modification.

Face authentication is vulnerable to both photo-based forgery attacks (PFA) and video-based forgery attacks (VFA). In PFA (or VFA), the adversary uses a photo (or video) containing the user's frontal face to bypass the otherwise highly-secure face authentication system. Both PFA and VFA are fairly easy to conduct, as the victim's photo or video usually can be easily found online, e.g., on popular social network sites. The adversary may also capture the victim's photo or video without being noticed, e.g., in crowded public places or through a high-definition camcorder from a long distance.

The prior defenses against PFA and/or VFA aim at *liveness detection*, which seeks to find a live indicator that the submitted face photo or video of the legitimate user is indeed captured in real time. The user's eye blink, lip movement, or head rotation in a video have been proposed as live indicators [43, 44]. These schemes are effective against PFA but invalid for VFA. The countermeasures against both PFA and VFA either use an infrared camera to obtain the thermogram of the user's face [45], or utilize texture analysis to detect the existence of a printed photo [46], or explore motion analysis to detect the existence of 2D images [47]. Besides very high computation complexity, these methods [45, 46, 47] require additional sensors or advanced cameras unavailable in COTS mobile devices.

The accelerometer in almost all COTS devices has recently been explored for liveness detection against PFA and VFA. In [48], Chen *et al.* proposed to compare the small motions extracted from the recorded video of the user's frontal face and those from the accelerometer to see if the motions are consistent. Similarly, Li *et al.* compared two motion vectors independently extracted from the video and the accelerometer of the mobile device for liveness detection [49]. Although these schemes

[48, 49] are very effective against PFA and VFA, they require the legitimate user to move the mobile device in front of him/herself in some predefined manner, which can be inconvenient or even socially awkward. In addition, the randomness of the user-generated device movement may be too limited so that the adversary may have a good chance to successfully imitate the user after careful observations.

1.4 Keystroke Inference Attacks on Mobile Devices

Keystroke inference attacks pose an increasing threat to mobile devices which have penetrated into everyday life. In a typical attack scenario, a victim types on the soft keyboard of his [1] smartphone or tablet in an insecure public environment such as a public library, a coffee shop, or a train. The attacker tries to infer the victim's keystrokes in order to obtain sensitive information such as the victim's device passwords, web account passwords, or even emails. Based on the inferred keystrokes, the attacker can proceed to launch further attacks. One example is that the attacker can use the inferred password to pass the authentication system of the victim's device. The severe security and privacy implications make keystroke inference a very active research topic in mobile device security.

Many keystroke inference attacks rely on analyzing a video recording the victim's typing process. They require that either the recorded video capture the victim's typing process with little or no visual obstruction [50, 51, 52, 53, 54, 55, 56, 57, 58] or the device be placed on a static holder [58]. Given the video recording, the attacker infers keystrokes by analyzing touchscreen reflection [55], spatial hand dynamics [56], relative finger movements on the touchscreen [57], or the backside motion of the device [58]. While these attacks have been demonstrated quite effective, their strong assumptions may not always hold in practice.

[1] No gender implication.

1.5 Acoustic Sensing for Mobile Authentication

Wearable devices are increasingly pervasive in our day-to-day lives. According to the 2017–2021 Cisco Visual Networking Index, the number of wearable devices globally is expected to grow threefold from 325 million in 2016 to 929 million by 2021. There are many reasons driving the ever-increasing popularity of wearable devices. For example, most wearable devices are of small form factor and specifically designed for fitness/health-related applications. In addition, latest wearable devices start to have many basic yet commonly used functionalities of smartphones, such as phone calls, text messages, emails, music streaming, mobile payment, navigation, and even an personal AI assistant, but they are much more preferred in contexts like exercising or sporting events.

Secure and usable user authentication for wearable devices is necessitated by growing concerns about data privacy. In particular, many wearable devices are designed to capture and temporarily store various physiological data such as ECG over long periods of time for health monitoring and fitness tracking. Moreover, wearable devices may contain contacts, texts, emails, location history, and other personal information. So we must design secure authentication techniques to prevent illegitimate access to such sensitive data. The authentication techniques also need to be very convenient for legitimate users to use, corresponding to the usability requirement.

Current authentication techniques for wearable devices do not satisfy the security or usability requirements. There are mainly two authentication techniques for wearable devices. The first involves the user inputting an unlock PIN, pattern lock, or password in the same way as unlocking a smartphone or tablet. Normal wearable devices have a much smaller screen than those on smartphones or tablets. For example, the display areas of the latest Apple Watch 4 (44 mm version) and Samsung

Galaxy Watch (46 mm version) are 9.77 and 10.8 cm^2, respectively. Inputting on such small screens is often challenging and frustrating for mobile users, especially for those with fat fingers, visually impaired users, senior citizens, and children. In addition, many wearable devices may not even have a screen. The second requires the user to pair his wearable device with his smartphone or tablet beforehand. Each subsequent unlocking of his wearable device is achieved by him unlocking the paired smartphone or tablet. For this technique to work, the user has to always carry the paired smartphone or tablet along with his wearable device, which somehow diminished the convenience of wearable devices. Furthermore, many users such as children may not even have a smartphone or tablet for pairing with their wearable devices. The lack of secure and usable authentication techniques makes many users leave their wearables unprotected, which may pose greater security threats beyond data privacy and are expected to be the top source of security breaches among IoT devices. [2]

[2]https://www.spiceworks.com/marketing/reports/iot-trends/

Chapter 2

RHYTHM-BASED TWO-FACTOR AUTHENTICATION FOR MULTI-TOUCH MOBILE DEVICES

2.1 Overview

This chapter explores a new direction to authenticate a mobile user based on her rhythmic taps/slides on the device screen. The strong promise of this direction is firmly rooted in some observations in daily life. First, many people tend to tap/slide on something nearby with a rhythm while singing a melody loudly or silently. Second, a user can easily repeat her rhythmic taps/slides over time for a familiar melody. Finally, different people are very likely to have different personal interpretations about the same melody and thus tap/slide in different ways; a user can even compose her own melody in mind instead of picking up a known melody. Therefore, rhythmic taps/slides are very difficult to emulate by an attacker with or without knowledge of the legitimate user's melody.

Our contributions are threefold.

- We propose RhyAuth [41], a novel two-factor rhythm-based authentication scheme for multi-touch mobile devices. RhyAuth requires a user to perform a sequence of rhythmic taps/slides on the device screen. The user is authenticated and admitted only when the features extracted from her rhythmic taps/slides match those stored on the device. RhyAuth is a two-factor authentication scheme because it requires both the correct rhythm (something-you-know) and the right way of performing the rhythm (someone-you-are).

- We theoretically analyze the security of RhyAuth. We show that RhyAuth is much more secure than the commonly used 4-digit PIN method, complex alphanumeric passwords, and Android Pattern Lock.

- We report comprehensive experimental evaluations of RhyAuth on Google Nexus 7 tablets, involving 22 legitimate users and 10 attackers. Our results show that RhyAuth is highly secure with false-positive and false-negative rates up to 0.7% and 4.2%, respectively. RhyAuth is also very efficient and can authenticate a user in less than 500 ms.

RhyAuth has many desirable features over existing techniques. Firstly, RhyAuth is highly resilient to brute-forth guessing attacks due to its two-factor nature. Secondly, RhyAuth is robust to shoulder-surfing attacks because it is very difficult for the attacker to figure out the exact rhythm by pure observations. Thirdly, RhyAuth is immune to smudge attacks, as the user can tap/slide on anywhere on the touch screen such that finger smudges can be more randomly distributed. Lastly, RhyAuth does not require the user to look at the screen while performing rhythmic taps/slides. The last feature indicates the high usability of RhyAuth to visually impaired people. It also means that a discrete user can conduct authentication with her device put under some cover (*e.g.*, a jacket or table) to eliminate shoulder-surfing attacks.

2.2 Related Work

There are some rhythm-based authentication schemes. In [59], Wobbrock *et al.* used the tapping on a button as the input of a rhythm for user authentication. Their experimental results showed a relatively low successful acceptance rate of 83.2%. In addition, their scheme does not target multi-touch mobile devices. In [60], Marques *et al.* transformed the timing information of taps on a touch screen into a sequence

and proposed a Hamming-distance-based matching approach for user authentication. However, the acceptance and rejection rates of their scheme are not reported. In [61], Lin *et al.* presented *RhythmLink*, a protocol to securely pair I/O-constrained devices by tapping.

Our work differs from the above schemes in many aspects. First, RhyAuth allows a user to input a rhythm by either tapping or sliding, while the above schemes only allow tapping. Second, RhyAuth additionally incorporates the behavioral biometrics of a user inputting the rhythm and thus can achieve much higher true acceptance and rejection rates. Finally, we conduct theoretical analysis of rhythm-based authentication for the first time in literature.

Recent years have also seen many mobile authentication techniques based on behavioral biometrics [16, 62, 39, 18]. In [63], however, Abdul *et al.* suggested that a programmable Lego robot could emulate users' behavioral biometrics to some extent, which poses potential threats on such techniques. In contrast, RhyAuth combines an additional user-chosen secret rhythm with her behavioral biometrics, thus achieving stronger attack resilience.

Finally, TouchIn [40] is a two-factor mobile authentication scheme that works by letting a user draw secret geometric curves on the device screen with one or multiple fingers. RhyAuth and TouchIn have similar authentication performance. But RhyAuth is more usable for people who have better memory for rhythms than for geometric curves.

2.3 Basics of Multi-Touch Screens

We introduce some background on multi-touch screens to help illustrate the RhyAuth design. Since we implement RhyAuth as an application on Google Nexus 7 tablets powered by Android 4.2, our illustrations here focus on Android and are applicable

to iOS with small modifications. A multi-touch screen can recognize two or more simultaneous contacts with the screen. When the screen is touched, a *touch event* is generated. The individual fingers or other objects, *e.g.*, a pen, that generate such events are referred to as *pointers*. Hereafter we assume that touch events are generated by fingers for simplicity.

RhyAuth collects the information about a touch event as a vector $\mathsf{info} = [t, \mathsf{fID}, x, y, P, S]$, where t is the time of the event, fID is the ID of the finger, x and y are the x and y coordinates of the touch point on the screen, respectively, P is the pressure generated by the finger, and S is the size of the touch point on the screen. We refer to (x, y) as a *sampled point*, or more simply, a *point*. Both P and S are normalized values in the range of $[0, 1]$. One thing worth mentioning is that a finger may generate a series of touch events even though a user believes that she does not move her finger. The reason is that a touch event will be generated whenever there is a slight change in any of x, y, P, and S, which may be imperceptible.

2.4 System Overview of RhyAuth

In this section, we give an overview of the RhyAuth design. RhyAuth consists of two subsystems, TapAuth and SlideAuth, in which a rhythm is input via finger taps and slides on the screen, respectively. A user needs to choose one to proceed when RhyAuth is invoked. Whichever subsystem is chosen, the whole authentication process comprises two phases: an enrollment phase and a verification phase.

2.4.1 Enrollment Phase

During this phase, a user first needs to choose a melody, of which the rhythm becomes her password. A good melody should be sufficiently familiar to the user so that she has little difficulty in repeating her password. It should also be sufficiently

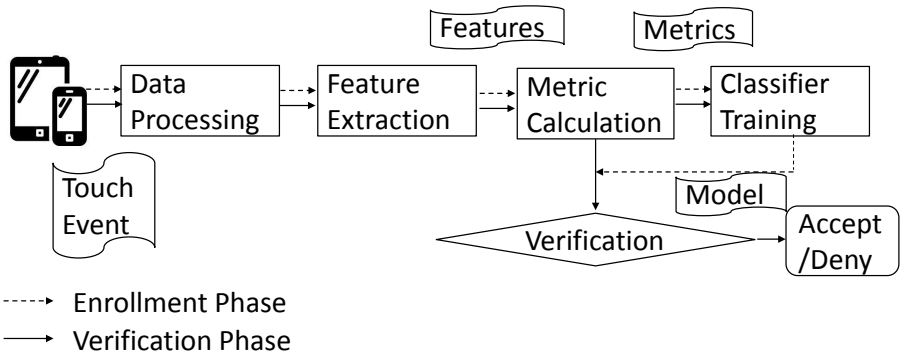


Figure 2.1: A system overview of RhyAuth, in which the dash and solid arrows represent the data flows in enrollment and verification phases, respectively.

random and thus cannot be easily figured out by an attacker. We will come back to this issue when analyzing the security of RhyAuth in Section 2.6.

Fig. 2.2 shows an excerpt of "Amazing Grace," which we use to introduce some relevant musical terms. A *note* is a sign used in musical notation to represent the relative duration and *pitch* of a sound. A pitch is an auditory sensation in which a listener assigns musical tones to relative positions on a musical scale. Usually, we denote a pitch by one of the seven letters of the Latin alphabet, *i.e.*, A, B, C, D, E, F, and G. So a note is a pitch with a defined duration. For example, in Fig. 2.2, the first note in the first *measure* is a *quarter note* with a duration of $\frac{1}{4}$ and a pitch of C4, while the first note in the second measure is a *half note* with a duration of $\frac{1}{2}$ and a pitch of F4. In our system, we are interested in the number of *"extended notes"* of a melody. An extended note refers to one note or multiple continuous notes of the same pitch connected by *ties*. A tie is a curved line connecting the heads of two notes of the same pitch and name, indicating that they are to be played as a single note with a duration equal to the sum of the individual durations. In practice, an extended note probably corresponds to one *tap* or *sub-slide* of RhyAuth, which will be explained shortly. For simplicity, we abbreviate "extened note" to " ex-note." Obviously, the

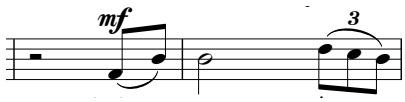


Figure 2.2: An excerpt of "Amazing Grace" [1].

number of ex-notes cannot be too small; otherwise, an attacker may figure out the rhythm easily. We assume that the melody has at least six ex-notes, and the user is asked about the number of ex-notes at the beginning of the enrollment phase.

Assume that the user chooses TapAuth. She continues to decide which finger(s) to tap on the screen. TapAuth gives the user full freedom to decide how she taps. A green user can choose and stick to one finger, while an advanced user may use multiple fingers and also switch fingers during her input. The user has to remember how many fingers she uses for each tap and inform RhyAuth about it. For example, if a melody has eight ex-notes, an advanced user may input the first four ex-notes with her middle finger, and the other ex-notes with her index and ring fingers together. Afterwards, the user needs to input the rhythm in the exact way. To avoid confusion, we refer to the tapping of one finger as a *touch*. A user may input an ex-note with multiple fingers, in which we refer to all the touches of an ex-note as a *tap*. Therefore, a tap can have one touch or multiple touches, and the number of taps is the same as the number of ex-notes.

Then a user inputs the rhythm by tapping on the screen according to her interpretation of her chosen melody. She needs to input the rhythm multiple times until a sufficiently good classifier can be obtained. As illustrated in Fig. 2.1, the touch-event data are first sent into a Data Processing module, which prepares the data for a Feature Extraction module. Then multiple distinguishable features are extracted and fed into a Metric Calculation module. Subsequently, a metric vector is generated

and sent to a Classifier Training module. Finally, a binary classifier is generated for the verification phase to determine whether a new input is legitimate or not.

SlideAuth follows the same system architecture and only differs in some implementation details. Firstly, the rhythm in SlideAuth is input via continuous finger sliding on the screen. Therefore, the user does not switch the finger(s) while she is inputting the rhythm. Secondly, we need to divide a continuous slide into multiple sub-slides, each corresponding to an ex-note. The end of each ex-note is marked by an abrupt change of the sliding direction. Finally, some features of SlideAuth are different from those of TapAuth.

2.4.2 Verification Phase

In this phase, the user first chooses between TapAuth and SlideAuth, and then the user inputs her rhythm by tapping or sliding. The input goes through the same Data Processing, Feature Extraction, and Metric Calculation modules in sequence. The resulting metric vector is finally fed into the Verification module, where the established classifier is applied to determine whether the user is legitimate or not.

2.5 Illustration of RhyAuth Modules

In this section, we detail each module of RhyAuth.

2.5.1 Data Processing

This module checks the consistency of the user input and prepares data for feature extraction. The steps below apply to each finger involved in either TapAuth or SlideAuth.

Data Processing for TapAuth

Firstly, as mentioned in Section 2.3, a touch of a finger on the screen generates multiple info vectors of format $[t, \mathsf{fID}, x, y, P, S]$. These info vectors have the same fID and are slightly different in other fields. Let $\bar{x}$, $\bar{y}$, $\bar{P}$, and $\bar{S}$ denote the average x, y, P, and S values, respectively. To reduce the data redundancy, we merge these info vectors into a single one with the same fID, $\bar{x}$, $\bar{y}$, $\bar{P}$, $\bar{S}$, and all the t values remain intact.

Secondly, the number of taps and the number of fingers in each tap are extracted. If these numbers are not consistent with the user's setting in the enrollment phase, the user input is immediately considered invalid and not further processed.

Data Processing for SlideAuth

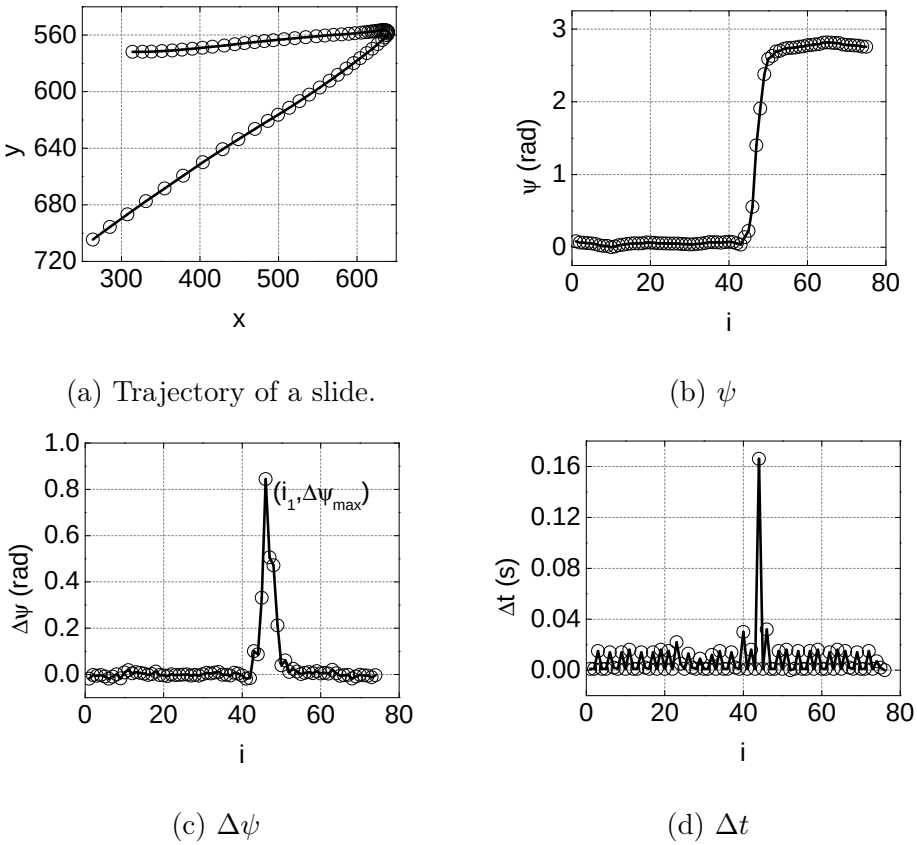
Firstly, we need to adjust the orientation of the slide to ensure that the device orientation and the starting direction of the slide have little effect on the authentication result. The orientation adjustment allows the user to input the rhythm more freely. We denote the coordinates of the slide as $\{(x_i, y_i)\}_{i=1}^{l}$, where l denotes the number of points of the slide. The slide orientation is adjusted such that the starting direction is aligned with the x axis, which is defined with the screen in the portrait mode. This is achieved in three steps. We first move the whole slide to make the coordinate of the first point $(0, 0)$ and change the coordinates to be $\{x_i', y_i'\}_{i=1}^{l}$, where $x_i' = x_i - x_1, y_i' = y_i - y_1$. Then we calculate the angles of $(\eta_1 - 1)$ vectors which start from $(0, 0)$ and end at $\{(x_i', y_i')\}_{i=2}^{\eta_1}$, respectively, denoted by $\{\theta_i\}_{i=2}^{\eta_1}$. The starting direction of the slide is denoted by θ_s and defined as the average of $\{\theta_i\}_{i=2}^{\eta_1}$. Finally, the coordinates of a slide are transformed to $\{x_i'', y_i''\}_{i=1}^{l}$, where $x_i'' = x_i' \cos\theta_s + y_i' \sin\theta_s, y_i'' = -x_i' \sin\theta_s + y_i' \cos\theta_s$. Here η_1 is an empirical parameter, and it should be chosen such that the resulting θ_s is a good estimation of the direction

of the first sub-slide. Note that η_1 cannot be too small, *e.g.*, two or three, to avoid instability. We use $\eta_1 = 5$ in our implementation.

Secondly, we smooth the trajectory of the slide because the collected data usually exhibit a jagged trajectory. We use a 10-point simple moving average (SMA) [64] filter for this purpose. After filtering, the coordinates become $\{\overline{x}_i, \overline{y}_i\}_{i=1}^{l}$.

Thirdly, we divide the whole smoothed slide into multiple sub-slides, each corresponding to an ex-note. This is equivalent to locating the last point of each sub-slide. Consider Fig. 2.3a as an example, where the slide consists of two sub-slides. A sharp change in the sliding direction indicates the end of the current ex-note or the beginning of the next ex-note. Given $\{\overline{x}_i, \overline{y}_i\}_{i=1}^{l}$, we first calculate the angles of the vectors connecting two consecutive points, *i.e.*, $\psi_i = \arccos(\frac{\overline{x}_{i+1}-\overline{x}_i}{\sqrt{(\overline{x}_{i+1}-\overline{x}_i)^2+(\overline{y}_{i+1}-\overline{y}_i)^2}}), i = 1, \ldots, l-1$. If a sequence of vectors are associated with the same sub-slide, the corresponding ψs should be similar. Fig. 2.3b is the corresponding plot of ψ. We can see that ψ switches from one stable value to another through a sharp transition phase, indicating a noticeable change of the sliding direction. To locate the last point of the first sub-slide, we further calculate $\Delta\psi_i = \psi_{i+1} - \psi_i, i = 1, \ldots, l-2$. Fig. 2.3c is the plot of $\Delta\psi$ in our example. Denote the index of the point with the largest $\Delta\psi$ by i_1. Then $\{i_1 - \eta_2, \ldots, i_1, \ldots, i_1 + \eta_2\}$ are indices of the points and include the last point of the first sub-slide. Here η_2 is an empirical parameter, and $\eta_2 = 10$ is adopted in our implementation. We proceed to calculate the time difference Δt of two consecutive points as $\Delta t_i = t_{i+1} - t_i$ as shown in Fig. 2.3d. The largest Δt corresponds to the last point. The above process can be easily extended to a slide with two or more sub-slides, in which case we just need to look for the last points of multiple sub-slides.

Finally, the number of fingers and the number of sub-slides are compared with the user's setting in the enrollment phase. If these numbers are not consistent, the sliding input is deemed invalid and not further processed.



(a) Trajectory of a slide.

(b) ψ

(c) $\Delta\psi$

(d) Δt

Figure 2.3: An example on dividing a slide into two sub-slides.

Some parameters need to be adjusted to properly divide a slide in practice. First, we need to decide when a change of ψ (or sliding direction) occurs. A straightforward solution is that when $\Delta\psi$ is larger than some threshold φ, a change of ψ occurs. In general, the suitable φ for different users is different, and we can train it during the enrollment phase. More specifically, we let $\varphi = \frac{\pi}{2}$ at the beginning of the enrollment phase, and the number of sub-slides set by the user is n. After slide division as described above, the number of sub-slides, denoted by n', is the number of changes of ψ plus one. If $n = n'$, the training of φ ends, and the current φ is used during the verification phase. If $n > n'$, φ is decreased by $\frac{\pi}{32}$ and increased by $\frac{\pi}{32}$ otherwise until $n = n'$.

2.5.2 Feature Extraction

This module takes the processed data from the Data Processing module as input and extracts the features used in RhyAuth. Depending on which of TapAuth and SlideAuth is chosen, the corresponding features are different. Most features are extracted from the data of each individual fID, while others are extracted by combining the data of multiple fIDs. The following descriptions apply to each fID involved in either TapAuth or SlideAuth.

Features of TapAuth

Below, n denotes the number of ex-notes (or equivalently taps), each corresponding to an integrated info vector formed in Data Processing.

- Intra-tap and inter-tap intervals: We denote the intra-tap interval by $\{\Delta t_{1,i}\}_{i=1}^{n}$ and the inter-tap interval by $\{\Delta t_{2,i}\}_{i=1}^{n-1}$. Let $t_{f,i}$ and $t_{l,i}$ denote the time of the first and last touch events associated with the ith tap, respectively. Then we have $\Delta t_{1,i} = t_{l,i} - t_{f,i}$ and $\Delta t_{2,i} = t_{f,i+1} - t_{l,i}$.

- Maximum and minimum pressure: These two features are denoted by $P_{\max}$ and $P_{\min}$, respectively. Then $P_{\max} = \max\{\overline{P}_i\}_{i=1}^{n}$ and $P_{\min} = \min\{\overline{P}_i\}_{i=1}^{n}$.

- Maximum and minimum size: We denote maximum and minimum size by $S_{\max}$ and $S_{\min}$, respectively. We have $S_{\max} = \max\{\overline{S}_i\}_{i=1}^{n}$ and $S_{\min} = \min\{\overline{S}_i\}_{i=1}^{n}$.

- Maximum and minimum distance: We denote them by $D_{\max}$ and $D_{\min}$, respectively. Suppose that the ith tap is associated with m fIDs and thus m coordinates. The distance of the ith tap is defined as the average Euclidean distance of each pair of coordinates, and it equals zero if $m = 1$. Then $D_{\max}$ and $D_{\min}$ are the maximum and minimum of the n tap-distance values.

- Maximum and minimum areas: We denote them by $A_{\max}$ and $A_{\min}$, respectively. Suppose that the ith tap is associated with m fIDs and thus m coordinates. If $m \geq 3$, the area of the ith tap is defined as the area of the convex hull determined by the m coordinates; otherwise, it is defined as zero. $A_{\max}$ and $A_{\min}$ are the maximum and minimum of the n tap-area values.

Features of SlideAuth

Below, n denotes the number of ex-notes (or equivalently the number of sub-slides) and N denotes the number of touch events.

- Intra-slide and inter-slide intervals: We denote them by $\{\Delta t_{3,i}\}_{i=1}^{n}$ and $\{\Delta t_{4,i}\}_{i=1}^{n-1}$, which are defined similarly to those of TapAuth.

- Maximum and minimum pressure: They are defined as $P_{\max} = \max\{P_i\}_{i=1}^{N}$ and $P_{\min} = \min\{P_i\}_{i=1}^{N}$.

- Maximum and minimum sizes: They are define as $S_{\max} = \max\{S_i\}_{i=1}^{N}$ and $S_{\min} = \min\{S_i\}_{i=1}^{N}$.

- Maximum and minimum slide length: We denote them by $L_{\max}$ and $L_{\min}$, respectively. For each sub-slide, we define its slide length as the distance between the coordinates of the first and last associated touch events. Then $L_{\max}$ and $L_{\min}$ are the maximum and minimum of the n slide lengths, respectively.

- Slide direction: For the ith ($i \in [1, N-1]$) point of the slide, we denote its slide direction by the angle, θ_i, of the vector from the ith point to the $(i+1)$th point. Therefore, $\{\theta_i\}_{i=1}^{N-1}$ can be calculated as

$$\theta_i = \arccos\left(\frac{\overline{x}_{i+1} - \overline{x}_i}{\sqrt{(\overline{x}_{i+1} - \overline{x}_i)^2 + (\overline{y}_{i+1} - \overline{y}_i)^2}}\right).$$

- Curvature: It is denoted by $\{\kappa_i\}_{i=2}^{N-1}$ and computed as

$$\kappa_i = \frac{4\Psi_i^y \Delta_i^x - 4\Psi_i^x \Delta_i^y}{((\Delta_i^x)^2 + (\Delta_i^y)^2)^{3/2}},$$

where $\Delta_i^x = (\overline{x}_{i-1} + \overline{x}_{i+1})/2, \Delta_i^y = (\overline{y}_{i-1} + \overline{y}_{i+1})/2, \Psi_i^x = (\overline{x}_{i+1} - 2\overline{x}_i + \overline{x}_{i-1})$, and $\Psi_i^y = (\overline{y}_{i+1} - 2\overline{y}_i + \overline{y}_{i-1})$.

- Velocities along the x axis and y axis: We denote them by $\{\upsilon_x\}_{i=1}^{N-1}$ and $\{\upsilon_y\}_{i=1}^{N-1}$, respectively and compute them as

$$\upsilon_{x,i} = \frac{\overline{x}_{i+1} - \overline{x}_i}{t_{i+1} - t_i} \text{ and } \upsilon_{y,i} = \frac{\overline{y}_{i+1} - \overline{y}_i}{t_{i+1} - t_i}.$$

- Accelerations along the x axis and y axis: We denote them by $\{a_{x,i}\}_{i=1}^{N-2}$ and $\{a_{y,i}\}_{i=1}^{N-2}$, respectively, which are computed as

$$a_{x,i} = \frac{\upsilon_{x,i+1} - \upsilon_{x,i}}{t_{i+1} - t_i} \text{ and } a_{y,i} = \frac{\upsilon_{y,i+1} - \upsilon_{y,i}}{t_{i+1} - t_i}.$$

- Distance: We denote it by D. Suppose that the slide is associated with m fIDs. If $m \geq 2$, D is defined as the average pairwise Euclidean distance among the first points of all the trajectories. If $m = 1$, we let $D = 0$.

- Area: We denote it by A. Suppose that the slide is associated with m fIDs. If $m \geq 3$, A is defined as the area of the convex hull determined by the first points of their trajectories; otherwise, we let $A = 0$.

2.5.3 Metric Calculation

This module is to consolidate the output from the Feature Extraction module into a metric vector. For TapAuth, the extracted features include $\{\Delta t_{1,i}\}_{i=1}^{n}, \{\Delta t_{2,i}\}_{i=1}^{n-1}$, $P_{\max}, P_{\min}, S_{\max}, S_{\min}, D_{\max}, D_{\min}, A_{\max}$, and $A_{\min}$; for SlideAuth, the extracted features include $\{\Delta t_{3,i}\}_{i=1}^{n}, \{\Delta t_{4,i}\}_{i=1}^{n-1}, P_{\max}, P_{\min}, S_{\max}, S_{\min}, L_{\max}, L_{\min}, \{\theta_i\}_{i=1}^{N-1}, \{\kappa_i\}_{i=2}^{N-1}$,

$\{v_{x,i}\}_{i=1}^{N-1}, \{v_{y,i}\}_{i=1}^{N-1}, \{a_{x,i}\}_{i=1}^{N-2}, \{a_{y,i}\}_{i=1}^{N-2}, D$, and A. Features like $\{\Delta t_{1,i}\}_{i=1}^{n}$ are in the vector form, and we would like to use a real number to denote each such feature for integration with non-vector features. This can be done by computing the distance between any vector feature and a reference vector. The comparison result (or the vector distance) is the real number we seek.

The vector features can also be divided into two categories, which require different comparison methods. Specifically, each feature vector of TapAuth is of length n, while that of SlideAuth is of length n or N. Both TapAuth and SlideAuth require two matching inputs to have the same number of ex-notes. This consistency check is done in the Data Processing module. Therefore, we can use statistical models to compare such feature vectors of the same length. In contrast, different inputs in SlideAuth most likely generate different numbers of touch events, leading to feature vectors of different lengths. We adopt Dynamic Time Warping (DTW) [65] to compare such feature vectors of variable lengths.

Comparison Based on Statistical Model

We take $\{\Delta t_{1,i}\}_{i=1}^{n}$ as an example to explain how to calculate the distance based on the statistical model. Suppose that there are Q samples from one user with the same n. We treat each element of $\{\Delta t_{1,i}\}_{i=1}^{n}$ as a Gaussian random variable. Given Q samples of $\{\Delta t_{1,i}\}_{i=1}^{n}$, we can calculate the mean and variance of each element. That is, we will have $\{(\mu_1, \sigma_1^2), \ldots, (\mu_n, \sigma_n^2)\}$ as the statistical model for $\{\Delta t_{1,i}\}_{i=1}^{n}$. Given a new sample of $\{\Delta t_{1,i}\}_{i=1}^{n}$, it is converted into $d_{\Delta t_1}$ as

$$d_{\Delta t_1} = \left(\sum_{i=1}^{n} \frac{(\Delta t_{1,i} - \mu_i)^2}{\sigma_i^2} \right)^{\frac{1}{2}} . \tag{2.1}$$

The intuition here is that a new sample of the same user most probably well follows the statistical model built from her historical data well, leading to a small $d_{\Delta t_1}$.

However, a sample from a different user is very likely to deviate much from this statistical model, resulting in a large $d_{\Delta t_1}$. In the same way, $\{\Delta t_{2,i}\}_{i=1}^{n-1}, \{\Delta t_{3,i}\}_{i=1}^{n}$, and $\{\Delta t_{4,i}\}_{i=1}^{n-1}$ are converted as $d_{\Delta t_2}, d_{\Delta t_3}$, and $d_{\Delta t_4}$, respectively.

Comparison Based on DTW

We use $\{\theta_i\}_{i=1}^{N-1}$ to explain how to calculate the distance based on DTW. Suppose that there are Q samples from one user. Assuming that the Q samples are quite similar, we randomly choose one as a reference and denote it by $\Theta^* = \{\theta_i^*\}_{i=1}^{N^*}$. DTW constructs a $(N-1) \times N^*$ matrix M with its (i,j) element $M(i,j) = |\theta_i - \theta_j^*|, i = 1, \ldots, N-1, j = 1, \ldots, N^*$. Then DTW looks for a non-decreasing path starting from $M(1,1)$ to $M(N-1, N^*)$ along which the sum of all elements would be the minimum of all possible paths. This minimum sum is used as the transformed value of $\{\theta_i\}_{i=1}^{N-1}$ and denoted by d_θ. Similarly, $\{\kappa_i\}_{i=2}^{N-1}, \{\upsilon_{x,i}\}_{i=1}^{N-1}, \{\upsilon_{y,i}\}_{i=1}^{N-1}, \{a_{x,i}\}_{i=1}^{N-2}$, and $\{a_{y,i}\}_{i=1}^{N-2}$ are transformed into $d_\kappa, d_{\upsilon_x}, d_{\upsilon_y}, d_{a_x}$, and d_{a_y}, respectively.

2.5.4 Classifier Training

This module is to train a binary classifier from the metric vectors of the legitimate user and other users.

We use SVM as the classification algorithm and LibSVM [66] in our implementation, which has been widely used and proved to achieve satisfactory performance under various circumstances. The classifier we need is a binary classifier, which classifies a sample (or metric vector) into the positive class or negative class. We use f_i to denote the class label of the ith sample. If $f_i = 1$, the sample is classified into the positive class, meaning that the sample is legitimate. If $f_i = -1$, the sample is classified into the negative class, indicating that the sample is illegitimate. In order to train the classifier, we need a training dataset consisting of metric vectors of both

the legitimate user and other users. For this purpose, a library of metric vectors of other users can be preloaded with each RhyAuth; it can also be downloaded in real time from a trusted server. Now suppose that we have n_s metric vectors or samples in total. Each of them is expanded into a sample-label pair (u_i, f_i), where u_i denotes the ith sample. $f_i = 1$ if u_i is a metric vector of the legitimate user and $f_i = -1$ otherwise. Given $\{(u_i, f_i)\}_{i=1}^{n_s}$, SVM solves the following optimization problem:

$$\begin{aligned} \min_{w,b,\xi} \quad & \frac{1}{2} w^T w + C \sum_{i=1}^{n_s} \xi_i \\ \text{subject to} \quad & f_i \cdot \left(w^T \phi(u_i) + b \right) \geq 1 - \xi_i, \\ & \xi_i \geq 0. \end{aligned} \tag{2.2}$$

Here u_is are mapped into a higher dimensional space by the function ϕ , and $K(u_i, u_j) \equiv \phi(u_i)^T \phi(u_j)$ is called the kernel function. SVM finds a linear separating hyperplane $w^T v + b = 0$ with the maximal margin in this higher dimensional space. Here v is a vector in the higher dimensional space, and $C > 0$ is the penalty parameter of the error term ξ_i. In our implementation, we choose the radial basis function (RBF) as the kernel function which has been proved to be a reasonable first choice. More specifically, the kernel function we choose is $K(u_i, u_j) = \exp(-\gamma \|u_i - u_j\|^2)$, where $\gamma > 0$ is the kernel parameter. The result of classifier training is a SVM model for the legitimate user, which predicts the class label of a new metric vector or sample.

2.5.5 Verification

A candidate user input goes through the same Data Processing, Feature Extraction, and Metric Calculation modules until a metric vector u is generated in either TapAuth or SlideAuth. The Verification module first verifies whether the user input has the same numbers of ex-notes and fingers as those of the legitimate user. If not, the user fails the authentication, and the verification stops. Otherwise, the Verifica-

tion module tests the candidate metric vector using the SVM model of the legitimate user. The SVM model consists of the optimal w and b, which are obtained by solving the optimization problem in Eq. 2.2. Given a candidate vector u, the decision function is $\mathsf{sgn}(w\phi(u) + b)$. If the result is 1, the user is considered legitimate and illegitimate otherwise.

2.6 Security Analysis

In this section, we analyze the security of RhyAuth. Unlike conventional authentication schemes involving alphanumerical or patter passwords, we cannot answer the question: "What is the size of the password space?" The reason is that RhyAuth combines a user-chosen rhythm and the user's behavioral biometrics together. In [67], Sherman *et al.* studied the security and memorability of user-generated free-form gestures for authentication. The metric they used is to quantify the "*surprisingness*" of a given gesture, rather than the security of their authentication scheme. Similarly, we focus on the security of a rhythm, which can be regarded as a lower bound of the overall security assessment of RhyAuth.

First, we want to answer the question: "Given a melody of n ex-notes, how many rhythms can there be?" Here we assume that a melody chosen by a user follows the music convention. Specifically, each ex-note consists of multiple notes; the duration of a note is one of the 12 note values, from 8 (*i.e.*, 2^3) corresponding to a maxima to $\frac{1}{256}$ (*i.e.*, 2^{-8}) corresponding to a two-hundred-fifty-sixth note; and the duration of a note with zero dot can be further augmented by adding one dot, two dots, and three dots. Therefore, a note may have $12 \times 4 = 48$ possible duration values. Although there is no limit on how many notes an ex-note can consist of, we assume that an ex-note lasts no more than two measures, each comprising no more than 24 notes for usability concerns. Therefore, an ex-note can consist of up to 48 notes. It is worth

noting that the number of possible duration values of an ex-note is not $48 \times 48 = 2304$, as some of the 2304 values are the same. For example, the duration of two notes with a note value of $\frac{1}{2}$ is the same as that of a single note with a note value of 1. So we further refine the 2304 values to eliminate redundant ones. Finally, we obtain 1002 unique possible duration values of an ex-note. Then for a melody of n ex-notes, the number of possible rhythms is simply $1002^n \approx 2^{10n}$. We should point out that this is an underestimation due to the assumption on how many notes an ex-note consists of. Here we give some numerical examples to briefly compare RhyAuth with the following schemes: (1) 4-digit PIN simple password of iOS, $10^4 \approx 2^{13}$; (2) n-character complex password of iOS, $77^n \approx 2^{6.27n}$; (3) Android Pattern Lock, around 2^{19} [68]. RhyAuth is obviously much more secure than all of them.

Secondly, we would like to answer the question: "Given an input rhythm, can the system suggest whether it is a good choice or not?" A firm answer to this question can help a user to choose a rhythm of high security strength. Inspired by [67], we use the surprisingness of a rhythm as a measure of its security strength. Specifically, we denote a rhythm of n ex-notes by $\{T_i\}_{i=1}^n$, where T_i is the duration of the ith ex-note and calculated as

$$T_i = \begin{cases} \Delta t_{1,i} + \Delta t_{2,i} & \text{if i} < \text{n}, \quad (2.3a) \\ \Delta t_{1,i} & \text{if i} = \text{n}. \quad (2.3b) \end{cases}$$

We assume $\{T_i\}_{i=1}^n$ follows a second-order autoregressive model as $T_i = \beta_0 + \beta_1 T_{i-1} + \beta_2 T_{i-2} + \epsilon_i$, where β_0, β_1, and β_2 are parameters to optimize by least squares fitting, and ϵ_i is the error term of T_i. Suppose that the least squares estimates are $\hat{\beta}_0, \hat{\beta}_1$, and $\hat{\beta}_2$ after parameter fitting. We then use $h(T)$ to denote the surprisingness of $\{T_i\}_{i=1}^n$, calculated as $h(T) = \left(\sum_{i=1}^n (T_i - \hat{T}_i)^2\right)^{\frac{1}{2}}$, where $\hat{T}_i = \hat{\beta}_0 + \hat{\beta}_1 \hat{T}_{i-1} + \hat{\beta}_2 \hat{T}_{i-2}$.

2.7 Performance Evaluation

In this section, we report the performance evaluation of RhyAuth, which we implemented as an application on Google Nexus 7 tablets running Android 4.2. In the rest of this section, we describe the attacker models, experimental setup, performance metrics, and experimental results in sequence.

2.7.1 Attacker Models

We consider the following models with increasingly capable attackers.

Type-I. The attacker knows neither the rhythm a user chooses nor how a user inputs the rhythm on the screen. Then the attacker's best effort is a brute-force attack.

Type-II. The attacker can observe how a user taps or slides on the screen multiple times, but he cannot figure out the exact rhythm the user chooses. The attacker can at best obtain a general idea of the rhythm through observations. For example, he may notice that the user taps eight times on the screen.

Type-III. The attacker knows the exact rhythm a user chooses and can also observe how the user taps or slides on the screen. Under this model, the attacker can input on the screen according to his own interpretation of the rhythm and his observations of the user.

Type-IV. An attacker knows exactly how a user taps or slides on the screen and the rhythm the user chooses. He, however, still needs to input the rhythm according to his own perception.

2.7.2 Experimental Setup

We recruited 32 volunteers for the experiments aged 18 to 35. Most of them are/were BS/MS students in Computer Science, Electrical Engineering, and Com-

puter Engineering. These volunteers were divided into two groups. The first group consisted of 22 volunteers to emulate legitimate users, and the second comprised 10 volunteers to emulate various attackers.

Every user was asked to come up with one melody with at least six ex-notes s/he can easily memorize and repeat. Also, every user was asked to input her/his rhythm with one-finger tapping, multi-finger tapping, one-finger sliding, and multi-finger sliding. Then every user practiced less than five minutes until s/he was confident to input them. Finally, every user input her/his rhythm using each method for 25 to 45 times, and we obtained 888 one-finger tapping samples, 870 multi-finger tapping samples, 894 single-finger sliding samples, and 873 multi-finger sliding samples. In reality, a RhyAuth user only needs to tap or slide a few times during the enrollment phase, as shown later. In addition, all the users were asked not to look at the screen to emulate people with visual impairment.

We also conducted experiments to evaluate the resilience of RhyAuth to various attacks. First, we video-recorded the input process of 14 users, and we also made the sound tracks of their chosen rhythms. We divided the 10 attackers into two groups of equal size. Each attacker in the first group randomly chose three users, watched their one-finger-tapping videos, and mimicked them. Then s/he randomly chose another three users, watched their multi-finger-tapping videos, and mimicked them. In total, there were six videos for each attacker in the first group. Each attacker in the second group did the similar experiments after watching one-finger and multi-finger sliding videos only. In accordance with our attacker models, we simulated the following five attack scenarios and collected $5 \times 6 \times 5 \times 2 = 300$ attacker samples for each attack.

1. One-time observation. The attacker was shown the video once. Then the attacker decided how to mimic the user, practiced, and input the rhythm five times.

2. Four-time observations. For each video he watched in Scenario 1, the attacker watched it for three more times and made five more attempts.

3. Four-time observations and one-time listening. After watching the video for four times, the attacker was allowed to listen to the sound track of the corresponding rhythm. Then the attacker combined his observations of the video and his perception of the rhythm together to make five more attempts.

4. Arbitrary observations and listenings. The attacker was allowed to watch each video and listen to the corresponding sound track for arbitrary times. Again, he could control how to watch the video and listen to the sound track. Finally, he made five more attempts when he was ready.

5. Arbitrary observations and listenings as well as how the user inputs her/his rhythm. The attacker could watch each video and listen to the corresponding sound track for arbitrary time in his own way. Furthermore, we asked each user to write down how s/he input her/his rhythm, including how many taps in her/his rhythm and whether the time between two consecutive taps was short or long for one-finger tapping, which fingers s/he used for each tap for multi-finger tapping, how many sub-slides s/he drew and whether the time s/he used to draw each sub-slide was short or long for one-finger sliding, and which fingers s/he used to slide for multi-finger sliding. Finally, the attacker combined all these information and mimicked the user for five more times.

Attacks 1 and 2 correspond to Type-II attackers, Attacks 3 and 4 correspond to Type-III attackers, and Attack 5 corresponds to Type-IV attackers.

2.7.3 Performance Metrics

We use receiver operating characteristic (ROC) and `Precision-Recall` curves to evaluate RhyAuth.

ROC Curve. A ROC curve is used to illustrate the performance of a binary classifier as its discrimination threshold changes. We can plot a ROC curve by plotting true positive rate (TPR) with respect to false positive rate (FPR) at various threshold settings. Denote the number of true positives, false positives, true negatives, and false negatives by #TP, #FP, #TN, and #FN. Then TPR and FPR can be calculated as

$$\texttt{TPR} = \frac{\#\texttt{TP}}{\#\texttt{TP} + \#\texttt{FN}} \text{ and } \texttt{FPR} = \frac{\#\texttt{FP}}{\#\texttt{FP} + \#\texttt{TN}}. \tag{2.4}$$

Precision-Recall Curve. Precision represents the percentage of legitimate users out of all admitted users and can be calculated as

$$\texttt{Precision} = \frac{\#\texttt{TP}}{\#\texttt{TP} + \#\texttt{FP}}. \tag{2.5}$$

Recall in authentication systems is the same as TPR, which measures the proportion of legitimate users who are correctly identified as such.

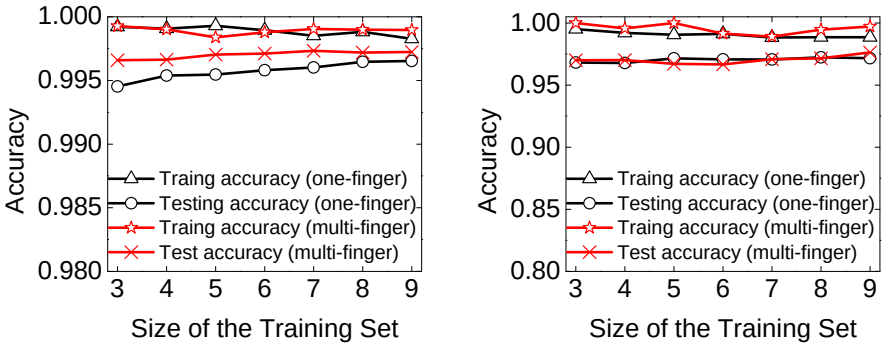
Authentication Time. We also measure the time RhyAuth takes to determine whether a user is legitimate or not, which should be as short as possible.

2.7.4 Experimental Results

Performance Without Attackers

This section demonstrates the performance of RhyAuth without attackers. Recall that each user was required to input her/his rhythm by four methods. The evaluation for each method was done as follows. For each user, we randomly chose ω samples from all the legitimate users to form a training set of 22ω samples for classifier training. The remaining samples were treated as the testing set. We did this evaluation 30 times for each user, and the results are the average results over the 30 times.

We first report the impact of the size of the training set on classification accuracy which can further be divided into training accuracy and testing accuracy. Here we



(a) Finger tapping.

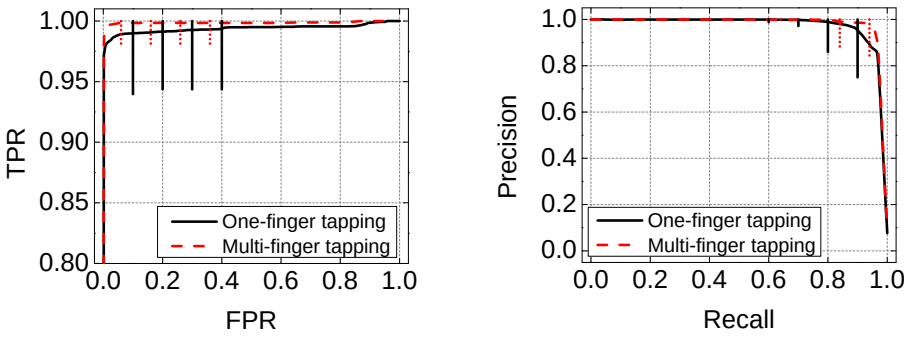
(b) Finger sliding.

Figure 2.4: Impact of the size of the training set.

define accuracy as the ratio of correctly classified users among all users in a dataset. We changed the size of the training set by varying ω and showed the results in Fig. 2.4. We can see that when ω varies from 3 to 9, the training and testing accuracy of four input methods only vary slightly. A smaller ω means a legitimate user can input her/his rhythm fewer times in the enrollment phase. When ω is larger than 5, the training and testing accuracy of four input methods stay stable. Therefore, we chose $\omega = 5$ for the later evaluations.

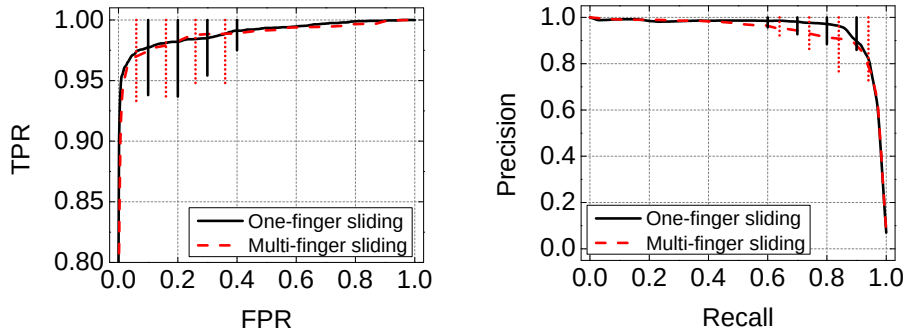
Fig. 2.5 shows the authentication performance of one-finger and multi-finger tapping, including the average results and the upper and lower bounds in the `Precision-Recall` and ROC curves. The ROC curves of the two methods are close to the top-left corner, which indicates that RhyAuth can achieve high `TPR` with low `FPR`. The `Precision-Recall` curves are close to the top-right corner, meaning that our system can achieve high `Precision` and high `Recall` at the same time.

Similarly, Fig. 2.6 illustrates the authentication performance of one-finger and multi-finger sliding. We can see that the ROC curves are close to the top-left corner and the `Precision-Recall` curves are close to the top-right corner, indicating that RhyAuth performs well in distinguishing a legitimate user from others. In contrast



(a) ROC curves. (b) `Precision-Recall` curves.

Figure 2.5: Authentication results with finger tapping.



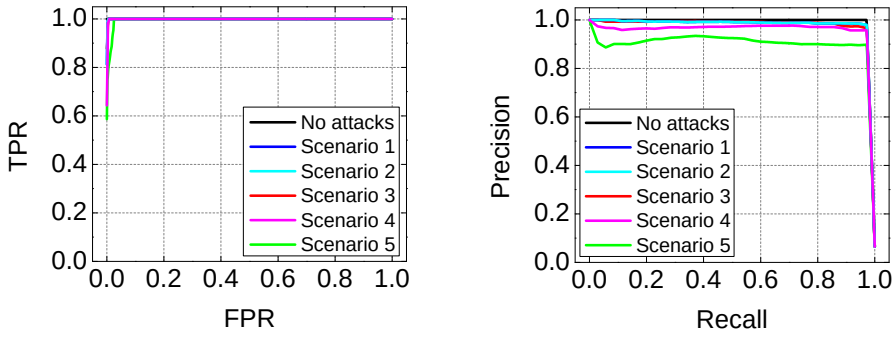
(a) ROC curves. (b) `Precision-Recall` curves.

Figure 2.6: Authentication results with finger sliding.

to finger tapping, finger sliding has slightly worse performance. The reason is that tapping with a rhythm is more natural than sliding with a rhythm for most people. As a result, tapping inputs are more consistent than sliding inputs, leading to fewer classification errors and thus fewer false negatives.

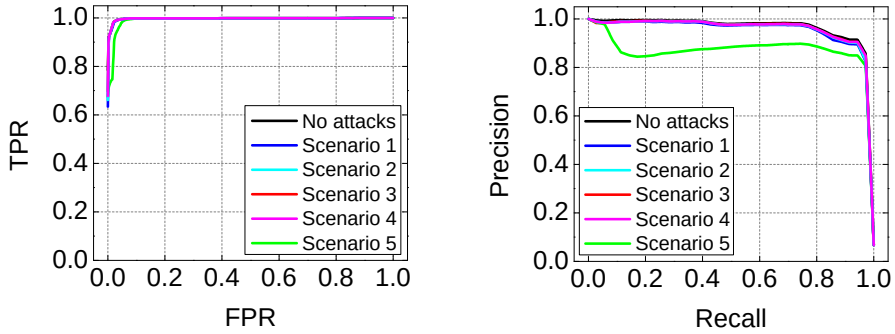
Performance With Attackers

This section reports the resilience of RhyAuth to attacks. For each victim under attacks, we first trained the classifier and obtained the SVM model of the victim.



(a) ROC curves. (b) `Precision-Recall` curves.

Figure 2.7: Attack resilience with one-finger tapping.

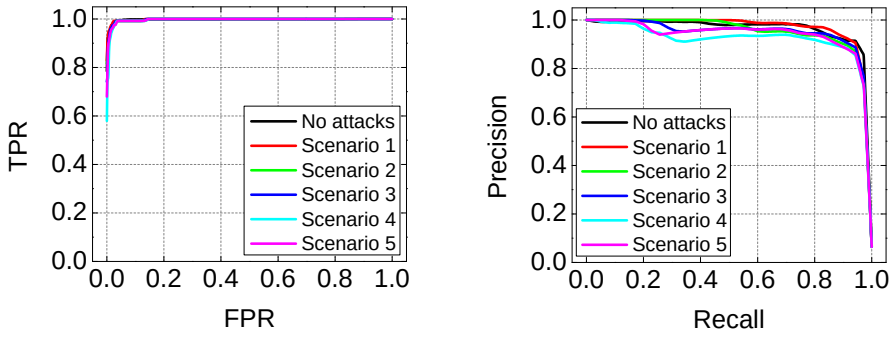


(a) ROC curves. (b) `Precision-Recall` curves.

Figure 2.8: Attack resilience with multi-finger tapping.

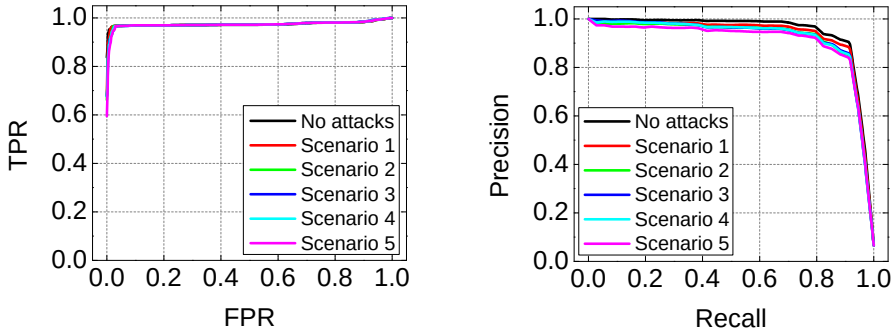
Then we added the samples of the attackers into the testing set and evaluated the performance of RhyAuth.

Fig. 2.7 and Fig. 2.8 show the results for one-finger and multi-finger tapping, respectively. We can see that both are highly resilient to the attacks. In addition, their attack resilience both slightly decreases as the capability of the attackers increases. Compared with one-finger tapping, multi-finger tapping is more resilient to the attacks. The reason is that more features are extracted from multi-finger tapping inputs.



(a) ROC curves. (b) `Precision-Recall` curves.

Figure 2.9: Attack resilience with one-finger sliding.



(a) ROC curves. (b) `Precision-Recall` curves.

Figure 2.10: Attack resilience with multi-finger sliding.

Fig. 2.9 and Fig. 2.10 show the results for one-finger sliding and multi-finger sliding, respectively. As we can see, both methods are also highly secure against the attacks, and their attacker resilience slightly decreases as well as the attacker becomes more capable. In addition, multi-finger sliding is more secure than one-finger sliding because more features are available in the former. Finally, finger sliding is more resilient to finger tapping, which is simply due to more features available in finger-sliding inputs.

Computation Time

We have also measured the computation time of RhyAuth. In particular, our measurements show that the one-time enrollment time of one-finger tapping and sliding are 14s and 200s, respectively, measured on a Dell desktop with 2.67 GHz CPU, 9 GB RAM, and Windows 7 64-bit Professional. Since the enrollment phase of RhyAuth takes relatively long time, we suggest that the classifier can be trained on a powerful desktop computer and then downloaded to the mobile device. This strategy has been advocated by many classifier-based mobile authentication schemes such as [18]. In addition, the verification time of one-finger tapping and sliding are 3.8ms and 500ms, respectively, measured on Google Nexus 7 tablet, which make them very practical. Finally, the computation time of multi-finger taping/sliding is simply that of one-finger taping/sliding multiplied by the number of fingers involved.

2.8 Conclusion

In this chapter, we presented the design and evaluation of RhyAuth, a novel rhythm-based two-factor authentication scheme for multi-touch mobile devices. Detailed security analysis and user experiments confirmed that RhyAuth is highly secure and usable for sighted and visually impaired people, thus has the great potential for wide adoption.

Chapter 3

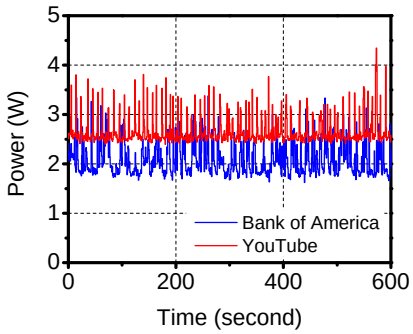
MOBILE APP FINGERPRINTING VIA POWER ANALYSIS

3.1 Overview

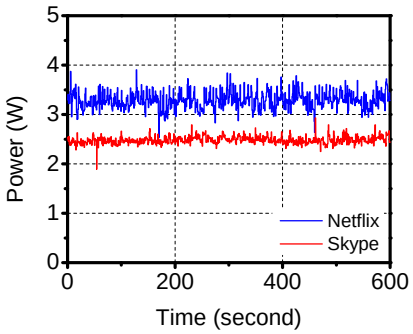
In this chapter, we present the design and evaluation of POWERFUL [69], a novel and practical attack framework for mobile app fingerprinting on Android devices through power profile analysis. POWERFUL is built upon the observation that different apps use different components of a device (e.g., touchscreen, CPU, Wi-Fi, and Bluetooth) and have different usage patterns, which result in distinguishable power consumption profiles. POWERFUL exploits the inherent heterogeneity of app power profiles for app characterization and usage inference. Since the power profiles on Android devices can be directly accessed without requiring user permission, POWERFUL is very difficult to detect and thus poses a serious and realistic threat against user privacy. Compared with existing traffic-based app fingerprinting techniques, POWERFUL does not require the adversary to be in the vicinity of the victim or compromise network service providers. Instead, it exploits the zero-permission Android public resources to obtain the power profiles of a device for app fingerprinting. Meanwhile, POWERFUL works well with apps generating little traffic.

Our main contributions are as follows.

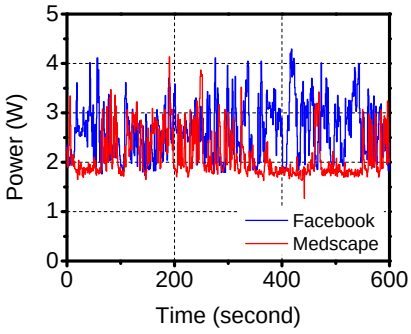
- We propose POWERFUL, the first mobile app fingerprinting framework for Android devices based on power analysis. Combining signal processing and machine learning techniques, POWERFUL is able to identify the app being used from a set of candidate apps with a high accuracy based on the corresponding power profile. Since Android requires no user permission to access and collect



(a) Bank of America and YouTube.



(b) Netflix and Skype.



(c) Facebook and Medscape.

Figure 3.1: Power profiles of several exemplary apps.

power profiles, POWERFUL poses a serious and realistic threat against user privacy.

- We evaluate the efficacy of POWERFUL via extensive experiments on a set of 22 most popular and privacy-related apps in Google Play Store. Experiment results show that POWERFUL can identify the app being used at any particular time from a set of candidate apps with accuracy up to 92.9% and is resilient to the change in various factors, such as locations (office, apartments, etc.), user activities (static or walking), and user variation.

3.2 Related Work

This section briefs some work closely related to POWERFUL.

3.2.1 Sensitive Information Inference in Android

Inferring sensitive information on Android mobile devices has received much attention in recent years. Previous work [21, 22, 23, 24, 25, 70] has shown that user input on the touchscreen, such as PIN, pattern password, user name, or even sentences, can be inferred from various onboard sensors such as accelerometer, gyroscope, microphone, or camera. In [26, 27], researchers show that accelerometer, microphone, camera, and light sensor can be used to infer target user's driving routes or locations. Some of these attacks [27] require user permissions such as `android.permission.CAMERA`, while our attack does not. Although access to sensors such as accelerometer does not require user permissions, accessing such information can be easily detected by analyzing API calls (e.g., `SensorManager.getDefaultSensor(int)`) using existing app analysis tools like [71].

Inferring sensitive information from Android's public resources has also been studied. In [28], Jana *et al.* show that the websites the user has visited and other finer-grained browsing behavior can be inferred from the memory footprint of the web browser. Zhang *et al.* show that keystroke events can be identified from the ESP data in a multi-core system [29].

In [30], Zhou *et al.* demonstrate that user's location, real identity, health conditions, and driving route can be inferred from the network usage statistics of an app and the status of public Android APIs. In addition, Chen *et al.* find that the UI state of Android device can be inferred from the memory usage of an app [31].

Compared with the above work, we work on a new attack on user privacy and make use of the power profile of Android devices, which is considered to be harmless.

3.2.2 App Fingerprinting

Our work is also related to the line of research in app fingerprinting, which aims at identifying apps through traffic analysis. In [32], Stöber *et al.* show that a group of apps can be identified as a whole by analyzing 3G/UMTS data traffic. In [33], Xu *et al.* design a learning system to automatically fingerprint an app using the key-value pairs in HTTP headers, while in [34], Miskovic *et al.* tackle a similar problem by exploring the scarcity of key app-identification sources. In [35], Dai *et al.* show that an app can be identified by analyzing different HTTP requests. In [36], Verde *et al.* propose a user-fingerprinting framework using NetFlow records only, rather than the entire traffic. Recently, Wang *et al.* show that the app being used can be inferred by analyzing the overheard encrypted data using machine learning techniques [37]. Compared with this line of work, our attacker does not need to in the vicinity of the adversary or compromise large network service providers. Also, our attack is valid for apps generated very limited traffic.

3.2.3 Power Analysis

There has also been some effort [72, 73, 74, 75, 76] in power analysis on mobile devices, which mainly focuses on understanding how power is consumed. In [72], Zhang *et al.* propose to first generate power models for device components such as CPU, LCD, and Wi-Fi and then use a function of these models to determine system-level power consumption. Similar approach is also adopted in [73], either to estimate the power consumption of an individual app or to fully understand the impact of different operating systems and hardware models. In [74, 75], Pathak *et al.* propose

to use system call tracing rather than the power states of hardware components to model power usage, which improves both accuracy and granularity. In [76], Brouwers *et al.* present NEAT, a novel energy analysis toolkit for smartphones, which combines both the accuracy of a customized power measurement board and detailed system traces of hardware and software together.

In [77], Michalevsky *et al.* introduce a novel attack that reveals user locations via power analysis of the user's smartphone. Assuming that the distance between the smartphone and the base station greatly impacts the total power consumption, they are able to infer the user's driving routes by applying machine learning techniques.

In contrast to the above work, we study a new attack using power analysis, which poses a serious and realistic threat on user privacy.

3.3 Preliminaries

3.3.1 Background

In this section, we briefly introduce the background of Andriod's public resources.

Android makes a subset of resources publicly accessible to all apps without requiring them to explicitly obtain permissions, as sharing these resources is generally considered harmless and makes them convenient to access by all apps whenever needed. The public directories in the Linux layer are an important category of the publicly accessible resources, most of which reside in two virtual filesystems: the proc filesystem (`/proc`) and the sys filesystem (`/sys`). In `/proc`, an app can access the resource usage of a process such as its usage of memory, CPU, and network, while in `/sys`, an app can find information about various kernel subsystems, hardware devices, and associated device drivers, etc. We obtain the device's voltage and current measure-

ments from the `voltage_now` and `current_now` files, respectively, both of which are public resources residing under the `/sys/class/power_supply/battery` folder.

3.3.2 Feasibility Study

In this section, we show the feasibility of inferring app usage by analyzing their power profiles. As mentioned in Section 3.1, POWERFUL explores the distinct characteristics in the power profiles of different apps caused by the heterogeneity of their resource usage and usage patterns.

Fig. 3.1a shows the power profiles of Bank of America (BoA) and YouTube apps. We can see from Fig. 3.1a that the two apps have similar power fluctuations, but YouTube has larger minimum and maximum powers than BoA. Therefore, minimum and maximum powers can be used to distinguish the two apps. Similarly, Fig. 3.1b shows that Netflix and Skype apps exhibit distinct characteristics of power profiles in terms of minimum power, maximum power, and power fluctuations, making them distinguishable by examining these features. Moreover, Facebook and Medscape apps have the similar minimum power as shown in Fig. 3.1c, but Facebook tends to have larger power fluctuations.

3.3.3 Adversary Model

We assume that the attacker runs a malicious app on the victim's device. As a standard assumption in Android security literature, it is backed up by the recent report that one out of ten Android apps are affected with malware and viruses [78]. The attacker also needs to know the device model and OS of the victim's device because these two factors directly affect the power consumption of the device. Such

information can be obtained from the `System` and `Build` class [1] without requesting any user permission.

The malicious app tries to be as stealthy as possible by running in the background to escape visual detection. According to our experiments, our "malicious" app for this research has a relatively stable power consumption of less than 20 mW, which has negligible influence on the collected power profiles. The app collects the power profile of the mobile device either periodically or following a predefined schedule. The app also needs to send the collected data to the attacker in a stealthy manner, which can be easily accomplished based on existing methods. For example, the malicious app may be inserted into or collude with another app which is legitimately given the `INTERNET` permission, and this approach is adopted by most existing work such as [23, 22, 21]. Alternatively, the malicious app can smuggle out the data across the Internet without requiring the user permission by using intent `URI ACTION_VIEW` to open a browser and sneaking the data to the parameters of an HTTP `GET` from the receiver side [79].

3.3.4 Targeted Sensitive Apps

Similar to [32, 36, 37, 31], we assume that the attacker is interested in fingerprinting a small set of selected apps that are popular, highly sensitive, or contain significant private user information. This is a common assumption as it is impractical for the attacker to build a classifier for all the existing mobile apps due to the large quantity. In addition, mobile apps are constantly evolving from time to time, and the cost to build and maintain a comprehensive database would be prohibitive. Moreover, many apps contain little private information about the user, which the attacker may lack incentives to fingerprint.

[1] For example, `System.getProperty("os.version")` returns the OS version of the device.

Categories	Apps
Communication	Gmail (GM), Messenger (MSG), Skype (SKY)
Education	TED (TED)
Entertainment	Netflix (NF), YouTube (YT)
Finance	Bank of America (BoA), Chase (CHA)
Games	Candy Crush (CCS), Pokémon Go (PM)
Health & Fitness	iTriage (iT), MedScape (MED), mySugr Diabetes Logbook (SDL)
Music & Audio	Spotify (SP)
News & Magazines	CNN (CNN)
Shopping	Amazon (AM), eBay (eBay), Groupon (GR)
Social	Facebook (FB), Twitter (TW), Tinder (TD)
Travel & Local	Priceline (PL)

Table 3.1: Apps and their abbreviations.

Table 3.1 lists the 22 apps which we study and are selected based on the following criteria.

1. A selected app is popular and has been downloaded for more than 500,000 times in Google Play Store.

2. A selected app is usually closely related to user privacy in some way, or its usage can be exploited by the attacker for more advanced attacks [31]. For example, communication, finance, health/fitness, shopping, and social apps can directly reveal important private information about the user, such as her/his accounts, health conditions, and online history, while location-based apps like Pokémon Go and Priceline can disclose user location traces.

3. The selected apps cover as diverse categories as possible in Google Play Store.

3.4 Design of POWERFUL

In this section, we detail the design of POWERFUL.

3.4.1 Overview

As shown in Fig. 3.2, POWERFUL consists of the following five steps.

1. *Power profile collection.* In this step, we implement an Android app to collect the instantaneous current and voltage measurements of a Google Nexus 7 tablet when the user is using a target app. According to our experiments, our data collection app has a relatively stable power consumption of less than 20 mW and therefore has little influence on the collected power profiles.

2. *Data processing.* In this step, we process the collected power profiles by compensating the difference of power consumption due to different brightness levels and then extracting the minimums and maximums of the power profile to facilitate subsequent feature extraction.

3. *Feature extraction.* In this step, we extract a feature vector comprising features in both time and frequency domains.

4. *Classifier training.* In this step, we first obtain the power profiles of the targeted sensitive apps in Table 3.1 and use lightweight machine learning algorithms to train classifiers for subsequent testing.

5. *App inference.* In this step, given an instance of the power profile of the user's device, we use the trained classifiers to determine the app(s) being used.

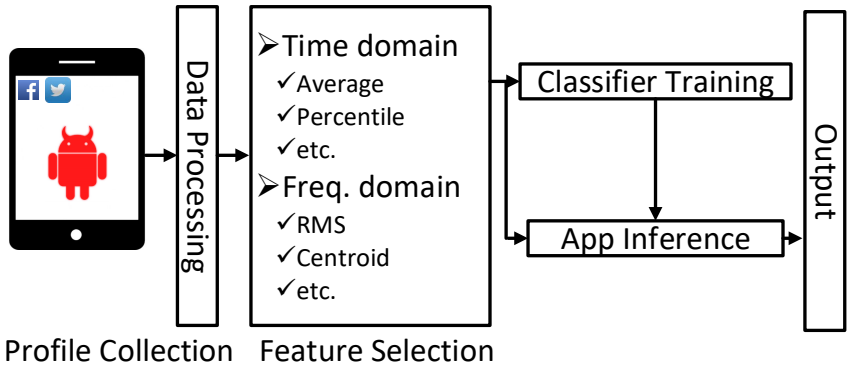


Figure 3.2: Flow chart of POWERFUL.

3.4.2 Power Profile Collection

On the victim side, the malicious app collects instantaneous current and voltage measurements of the device by reading `/sys/class/power_supply/battery/current_now` and `/sys/class/power_supply/battery/voltage_now`, respectively, either periodically or following a predefined schedule. In our experiments, we set the sampling frequency to 2 Hz to strike a good balance between profile accuracy and the amount of data that need be stealthily transmitted to the attacker through the Internet. After collecting voltage and current measurements for a sufficiently long period, the app constructs the power profile that comprises a sequence of instantaneous power measurements computed as the products of the corresponding current and voltage measurements. The app also obtains the current brightness level of the device in the public system setting `android.provider.Settings.System.SCREEN_BRIGHTNESS`, which requires no user permission to access. The app finally sends the power profile and the brightness level of the victim's device to the attacker.

The attacker also builds a power profile for each target app. In particular, the attacker employs multiple users to use every target app on a device with the same

model and OS and builds a power profile for each target app for subsequent classifier training and app inference.

3.4.3 Data Processing

In this step, we process the raw power profiles to facilitate subsequent feature extraction. Without loss of generality, we consider a power profile $P = (p_1, \ldots, p_n)$, where p_i is the ith power measurement for all $i \in [1, n]$ and n is the total number of power measurements.

We first apply a sliding window of length W and offset factor r on P to generate a sequence of power profile samples $S_1, \ldots, S_k$ of equal length, where

$$S_i = (p_{(i-1)rW+1}, \ldots, p_{(i-1)rW+W}),$$

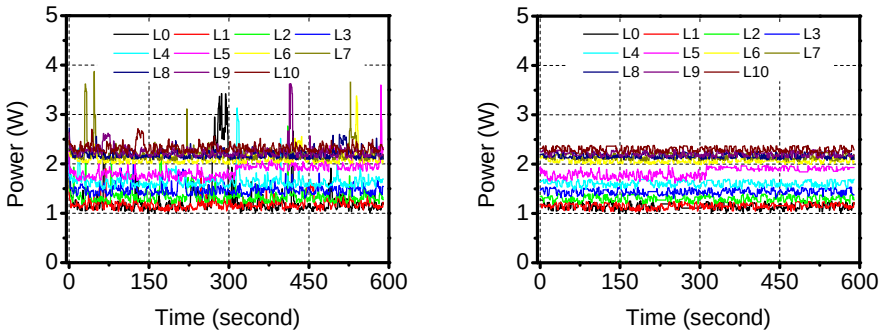
for all $i = 1, \ldots, k$, and $k = \lfloor \frac{n-W}{rW} \rfloor$. In our system, we empirically set r to 0.1 and choose W such that $rW \in \mathbb{Z}$.

For each sample S_i, we proceed with the following two steps: *power adjustment* and *min-max search*.

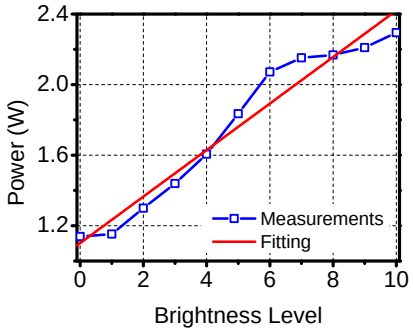
Power adjustment

In this step, we compensate the difference in power consumption caused by different brightness levels. Such adjustment is necessary because the touchscreen is a major energy-consuming component in modern mobile devices, and different brightness levels result in different power consumption rates of the touchscreen and therefore different power profiles for the same device.

Power adjustment requires a power model to characterize the relationship between touchscreen power consumption and brightness level. While several models have been proposed in the literature [72, 74], they are either device-specific due to the technology



(a) Original power profiles. (b) Our linear power model.



(c) Our linear power model

Figure 3.3: Power adjustment for different touchscreen brightness levels.

and hardware difference or require user permission to acquire the status of different components. In our system, we adopt a simple linear relationship between the power consumption and the brightness level built from fitting empirical data.

Specifically, we collect the power measurements at different brightness levels using a modified version of our app. No third-party app is installed on the device to minimize potential impact on accuracy. The app automatically sets the brightness coefficient from 0 to 1 with an interval of 0.1 (corresponding to level 0 to level 10) and records the voltage and current measurements at a frequency of 2 Hz for 10 minutes before changing to the next brightness level. Fig. 3.3a shows the power measurements at different brightness levels. We can see that the curves exhibit multiple peaks while

they are expected to be generally stable. We conjecture that the peaks are most likely due to pre-installed system apps (e.g., Google Play services) running in the background and thus should be excluded during parameter fitting. Our measurement results are similar to those collected using the tools in [72].

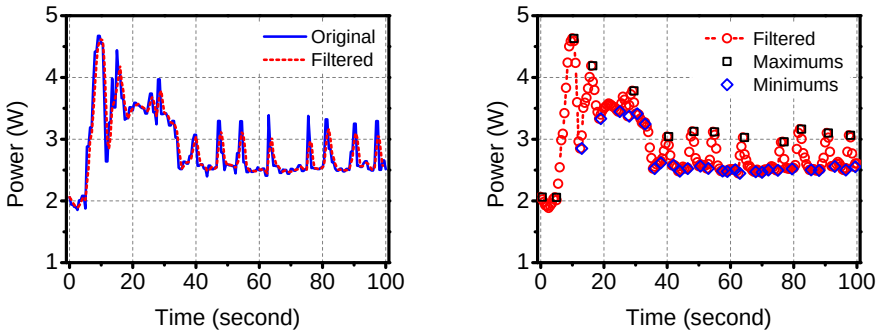
We remove these peaks in two steps. First, we calculate the cumulative distribution function (CDF) of the power measurements. Second, we remove the measurements above a certain percentile of the CDF, where we empirically choose 80% as the threshold. We plot the power measurements after removing the peaks in Fig. 3.3b, where we can see that the resulting power profiles are generally stable.

We then calculate the average of the (approximate) 10-minute measurements as the device's power consumption rate at the corresponding brightness level. Given a set of brightness levels and corresponding power consumption rates, we further calculate the slope s and intercept b of the linear model through least-squares fitting. We plot the measurements and the fitted model in Fig. 3.3c and show the fitted parameters in Table 3.2.

We finally adjust the power measurements using the power model obtained above. Specifically, given the device's current brightness level L, we calculate the power consumption rate difference between level L and level 0 as sL, where s is the slope of the linear power model. Then for every power profile sample $S_i = (p_{(i-1)rW+1}, \ldots, p_{((i-1)r+1)W})$, we compute a new sample $S'_i = (p'_{(i-1)rW+1}, \ldots, p'_{((i-1)r+1)W})$, where $p'_j = p_j - sL$ for all $j = (i-1)rW + 1, \ldots, (i-1)rW + W$.

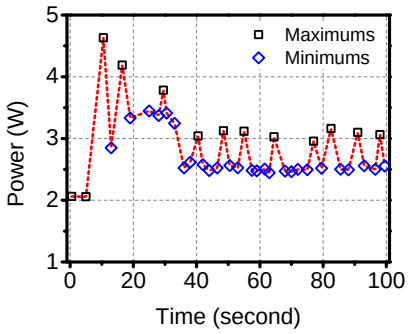
Parameter	Value	Standard Error
Intercept, b	1.10	0.057
Slope, s	0.132	9.65×10^{-3}

Table 3.2: Fitted parameters of our linear power model.



(a) Effect of SMA filtering.

(b) Local maximums and minimums found by our approach.



(c) "Skeleton" composed of local maximums and minimums.

Figure 3.4: Illustration of min-max search.

Power adjustment for different touchscreen brightness levels.

Min-max search

Next, we extract the "skeleton" of each power profile sample by finding the local minimums and maximums of its power measurements.

Without loss of generality, we consider a power profile sample $S' = (p'_1, \ldots, p'_W)$ and use the example in Fig. 3.4 to illustrate how local minimums and maximums are determined. First, we apply a five-point simple moving average (SMA) filter to

smooth the power profile sample to reduce the impact of small fluctuations. Fig. 3.4a shows the original and filtered power traces in our example. Denote the power trace after filtering by $\tilde{p}_1, \ldots, \tilde{p}_W$. Then for each $\tilde{p}_j, j = 1, \ldots, W$, we select $\tilde{p}_j$ as a *local minimum (maximum)* if the following two conditions are satisfied.

- **Condition 1**: Its value is no larger (smaller) its two neighboring values, i.e., $\tilde{p}_j \leq \tilde{p}_{j-1}$ and $\tilde{p}_j \leq \tilde{p}_{j+1}$ ($\tilde{p}_j \geq \tilde{p}_{j-1}$ and $\tilde{p}_j \geq \tilde{p}_{j+1}$).
- **Condition 2**: Its value is at least δ_t smaller (larger) than the previous closest local maximum (minimum).

Here δ_t is an important parameter that need be chosen carefully. On the one hand, if δ_t is too small, too many minimums or maximums will be selected due to small fluctuations in the power profile caused by spontaneous noise, which do not contribute to the characterization of the app. On the other hand, an overly large δ_t makes the second condition difficult to satisfy, resulting in some meaningful minimums or maximums being omitted and thus poor characterization of the app. Moreover, since different apps exhibit distinct characteristics, the choice of δ_t should not be universal but app-specific. We observe that a proper δ_t should be positively correlated with the standard deviation σ of the power measurements for a given app. Therefore, we choose $\delta_t = c\sigma$ and empirically set $c = 1$ in this work. Fig. 3.4b and 3.4c show the labeled local minimums and maximums of the given power trace in Fig. 3.4a. The figures shows that they capture the overall shape of the power trace, indicating the capability of extracting the "skeleton".

After finding all the local minimums and maximums from the power profile sample, we generate a vector of pair $V = \big((m_1, t_1), \ldots, (m_e, t_e)\big)$, where m_j and t_j are the jth local maximum or minimum and the corresponding time stamp, respectively, and e is the total number of local maximums and minimums. Moreover, we generate a label

vector $L = (l_1, \ldots, l_e)$, where $l_j = -1$ if m_j is a local minimum and 1 otherwise. It follows that $l_j l_{j+1} = -1$ if m_j and m_{j+1} are a pair of adjacent minimum and maximum.

We further compute a power difference vector ΔV, a time difference vector ΔT, and a slope vector R from V and L using Algorithm 1, which capture the power and time differences of pairs of adjacent minimum and maximum and the sharpness of the corresponding rising or falling slopes, respectively.

Algorithm 1 Computing $\Delta V, \Delta T$, and R

Input: V, L, e

Output: $\Delta V, \Delta T, R$

Initialization: $\Delta V \leftarrow \emptyset, \Delta T \leftarrow \emptyset, R \leftarrow \emptyset, j \leftarrow 1$
for $i = 1, \ldots, e-1$ **do**
  if $l_i l_{i+1} = -1$ **then**
    $\Delta V[j] \leftarrow |m_{i+1} - m_i|$;
    $\Delta T[j] \leftarrow t_{i+1} - t_i$;
    $R[j] \leftarrow \frac{\Delta V(i)}{\Delta T(i)}$;
    $j \leftarrow j + 1$;
  end if
end for
return $\Delta V, \Delta T, R$

3.4.4 Feature Extraction

In this step, we extract features from both time and frequency domains to represent a given power profile sample. For each power profile sample, the extracted features form a vector, which is referred to as an instance hereafter.

Features in time domain

For a given power profile sample $S' = (p'_1, \ldots, p'_W)$, we extract the following statistic measures as the features in time domain.

- The average, the 20th, 50th, and 80th percentile, the standard deviation (SD), the maximum, and the minimum of $(p'_1, \ldots, p'_W)$, denoted by $p_{avg}, p_{20pctl}, p_{50pctl}$, $p_{80pctl}, p_{SD}, p_{max}$, and p_{min}, respectively.

- The average, the 20th, 50th, and 80th percentile, SD of $\Delta V, \Delta T$, and R, respectively. We denote them as $\Delta V_{avg}, \Delta V_{20pctl}, \Delta V_{50pctl}, \Delta V_{80pctl}, \Delta V_{SD}, \Delta T_{avg}, \Delta T_{20pctl}$, $\Delta T_{50pctl}, \Delta T_{80pctl}, \Delta T_{SD}, R_{avg}, R_{20pctl}, R_{50pctl}, R_{80pctl}$, and R_{SD}, respectively.

Features in frequency domain

Given a power profile sample $S' = (p'_1, \ldots, p'_W)$, we first calculate its Fourier Transform as $Q = (q_1, \ldots, q_W)$ using Fast Fourier Transform (FFT). We then extract the following features from Q.

- *Root-mean-square (RMS) energy.* The RMS energy is an approximation of the average signal strength and calculated as the square root of the arithmetic mean of the squares of Q.

$$\mathsf{RMS} = \sqrt{\frac{1}{W}\sum_{k=1}^{W} q_k^2}.$$

- *Spectral centroid.* The spectral centroid represents the "center" of Q and is the weighted mean of the frequencies of Q with q_k as the weights.

$$\mu = \frac{\sum_{k=1}^{W} q_k f_k}{\sum_{k=1}^{W} q_k},$$

where $f_k = \frac{k f_s}{2}$, and f_s is the sampling frequency.

- *Spectral entropy.* The spectral entropy captures the locations of the peaks of Q and is computed as

$$H = \sum_{k=1}^{W} \omega_k \log_2 \omega_k,$$

where $\omega_k = \frac{q_k}{\sum_{k=1}^{W} q_k}$ is the normalized frequency.

- *Spectral irregularity.* The spectral irregularity captures the jitter or noise in Q and is given by

$$\mathsf{irregularity} = \frac{\sum_{l=1}^{n_p} (\phi(x) - \phi(x+1))^2}{\sum_{x=1}^{n_p} \phi(x)^2},$$

where $\phi(x), x = 1, 2, \ldots, n_p$ are the peaks in Q and $\phi(n_p + 1) = 0$.

- *Spectral spread.* The spectral spread is to capture the dispersion of Q with respect to its centroid and calculated as the standard deviation of the spectral distribution.

$$\rho = \sqrt{\sum_{k=1}^{W} [\omega_k (f_k - \mu)^2]}.$$

- *Spectral skewness.* The spectral skewness captures the symmetry of Q and is computed as

$$\mathsf{skewness} = \frac{\sum_{k=1}^{W} \omega_k (f_k - \mu)^3}{\rho^3}.$$

- *Spectral kurtosis.* The spectral kurtosis captures the flatness of Q compared with Gaussian distribution and is given by

$$\mathsf{kurtosis} = \frac{\sum_{k=1}^{W} \omega_k (f_k - \mu)^4}{\rho^4}.$$

- *Spectral flatness.* The spectral flatness captures how energy is spread across Q and is given by

$$\mathsf{flatness} = \frac{(\prod_{k=1}^{W} q_k)^{\frac{1}{W}}}{\frac{1}{W} \sum_{k=1}^{W} q_k}.$$

3.4.5 *Classifier Training*

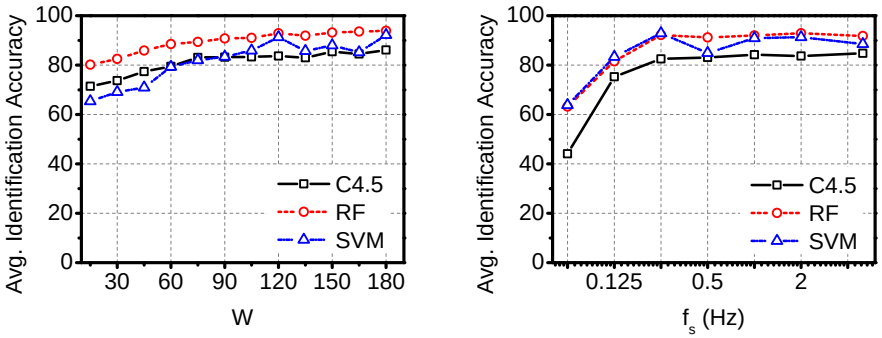
In this step, we train a classifier from a training set with known labels. POWERFUL can work with many existing machine learning techniques. In our system, we consider three lightweight supervised machine learning techniques, i.e., C4.5, random forest (RF), and support vector machine (SVM) for classifier training and testing. In particular, C4.5 generates a decision tree according to C4.5 algorithm and uses the decision tree to map an instance to a finite set of values, which are the class labels [80]. RF builds a forest of uncorrelated decision tress for classification [81]. The unique feature of RF is that it keeps selecting a random subset of features to control the variance of classification result during the process of generating the forest. SVM maps the instances of different classes in space such that they are divided by a clear gap which is as wide as possible [82, 66]. In Weka [83], the corresponding implementations of the three techniques are `J48` , `RandomForest`, and `LibSVM` class, respectively.

3.4.6 *App Inference*

In this step, we infer the app being used from the given power profile. Specifically, the power profile first goes through Data Processing and Feature Extraction steps and becomes a series of instances. Then given a specific instance, we use the classifier trained in Section 3.4.5 to calculate the class label. The app corresponding to the output class label is considered as the app being used at the particular time.

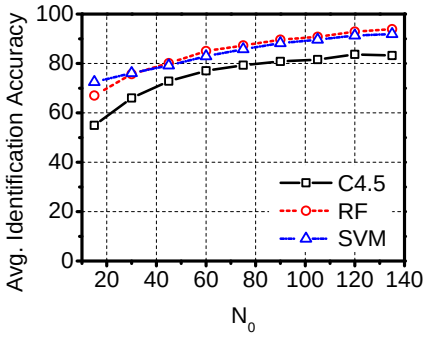
3.5 Performance Evaluation

In this section, we first describe the experimental setup, then introduce the performance metric adopted, and finally report the evaluation results in details.



(a) Impact of W.

(b) Impact of f_s.



(c) Impact of N_0.

Figure 3.5: Impact of different factors on POWERFUL.

3.5.1 Experiment Setup

Data collection

We used a Google Nexus 7 tablet with Android 4.4.4 and a Google Nexus 6 smartphone with Android 5.1.1 with our app installed to collect power profiles. We recruited 24 users to participant in the experiments, including two females and twenty-two males. Each participant was assigned ten apps and then delivered the following instructions. First, each participant adjusted the screen brightness level according to her/his need, connected the device to a reliable Wi-Fi AP, and turned off the Bluetooth connection. Then, s/he activated the data collection app and started to use one

of the assigned apps. Each participant was required to stay where s/he was, such as in the office or at her/his apartment, when s/he was using an app. Each participant was also asked to use only one assigned app at any given time. For each assigned app, the participant was asked to use it for at least half an hour. To complete the data collection of one assigned app, the participant was allowed to use it either continuously or from time to time, as long as the total usage time of each assigned app exceeded half an hour. We also required the participant not to change the brightness setting during the experiments.

Evaluation protocol

Our main dataset, denoted by $\mathcal{S}$, consists of the instances of all 22 sensitive mobile apps. In total, 24 participants are involved in the experiments, who are graduate students in Arizona State University and age between 20 and 35. Each participant was assigned ten apps to use. Considering the total number of participants in the experiments, only two apps (BoA and CHA) are used by ten participants while all the other apps are each used by eleven participants.

For each run of evaluation, we randomly divided $\mathcal{S}$ into one training set $\mathcal{S}_{\text{train}}$ and one testing set $\mathcal{S}_{\text{test}}$. Based on our adversary model, $\mathcal{S}_{\text{train}}$ is built by the attacker while $\mathcal{S}_{\text{test}}$ is obtained from the malicious app on the victim's device. For each app we studied, we randomly selected one participant as the victim and allocated her/his instances to $\mathcal{S}_{\text{test}}$ while allocating the remaining instances of the same app to $\mathcal{S}_{\text{train}}$. By doing so, we ensured that the instances of $\mathcal{S}_{\text{train}}$ were from the attacker and those of $\mathcal{S}_{\text{test}}$ were from the victim. Finally, we ran the evaluation for 40 times and reported the average results.

3.5.2 Performance Metric

We use *identification rate* as the performance metric to evaluate the attack capability of POWERFUL. In specific, for each app we study, we define the identification rate as the ratio between the number of correctly-classified instances and that of all instances of the app in a testing set. A higher identification rate means that given a power profile, the attacker (POWERFUL) is able to identify the corresponding app used by the victim more accurately, thus posing a more serious threat on user privacy.

3.5.3 Experimental Results

Impact of window length

Fig. 3.5a shows the average identification rate of all the apps when window length W increases from 15 to 180. As we can see, the average identification rate increases as W increases for all three machine learning techniques. This is expected because a larger W means that the power profile with more measurements is used for identification, and it is thus more likely to extract app-specific features to increase the identification rate. On the other hand, a smaller W means that the attacker only needs to collect a power profile for a shorter period, making the attack more practical. We set W to 120 for the rest of our experiments, corresponding to a duration of 60 seconds.

Impact of sampling frequency

Fig. 3.5b shows the average identification rate across all the 22 apps with sampling frequency f_s varying from 0.00625 Hz to 5 Hz. We can see that the average identification rate increases as f_s increases for all three machine learning techniques. The reason is that the higher f_s, the finer-grained characteristics of the collected power profiles. We can also observe that the identification rate tends to be stable when f_s is

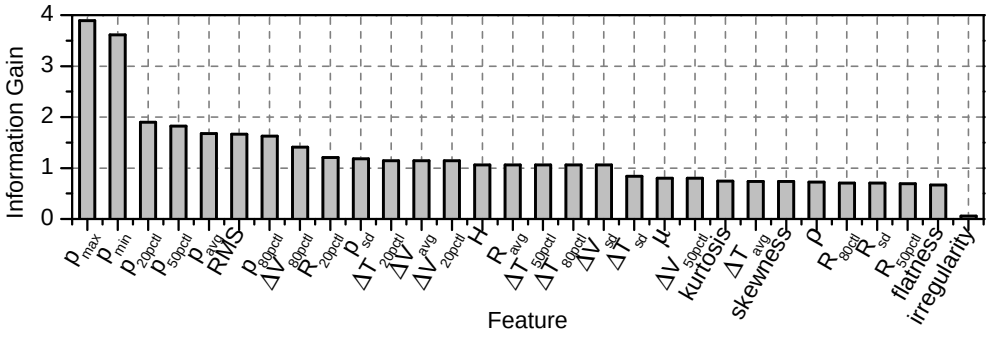


Figure 3.6: Importance of features.

higher than 2 Hz. This is mainly because that the extracted features do not change much when further increasing f_s. Since a higher f_s leads to more power profiles that need be stealthily transmitted over the Internet, making our attack easier to detect, f_s is set as 2 Hz in our experiments to strike a balance.

Impact of number of training instances

Fig. 3.5c shows the average identification rate across all the 22 apps with the number of training instances N_0 varying from 15 to 135. We can see that the average identification rate increases as N_0 increases for all three machine learning techniques. This is expected because the classifiers get better trained with more training instances and consequently achieve higher identification rate. As a result, the attacker always uses all the available training instances in practice.

Feature importance

We have also studied the importance of different features used in POWERFUL, characterized by *information gain* [84, 85]. Specifically, information gain measures the amount of information about class prediction, given that the only information available is the presence of a feature and the corresponding class distribution. The higher

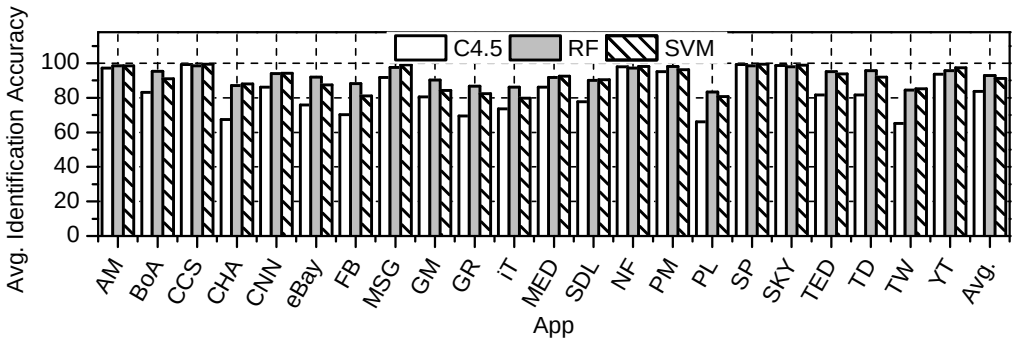


Figure 3.7: Identification accuracy of POWERFUL on Nexus 7.

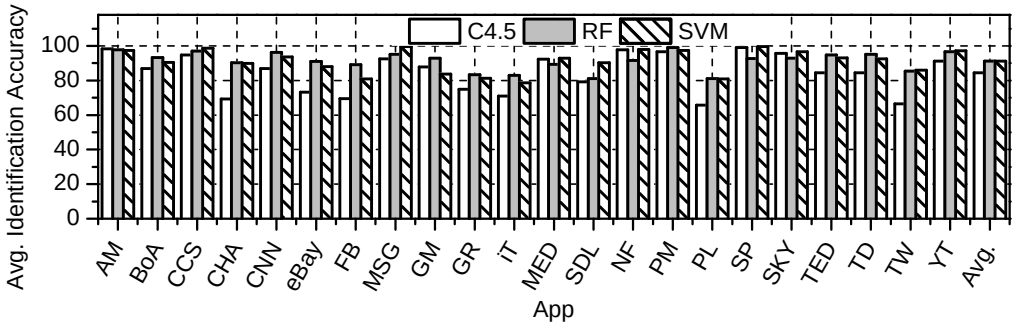


Figure 3.8: Identification accuracy of POWERFUL on Nexus 6.

information gain, the more important a feature is, and vice versa. In Weka, we obtain the information gain of different features using its `InfoGainAttributeEval` class. Fig. 3.6 shows the information gain of all the features of POWERFUL in descending order. We can see that two most important features are p_{max} and p_{min} while the least important is irregularity.

The less important features (i.e., with small information gain) still affect the overall classification results. For example, if we remove the features with information gain less than one, the average identification rate decreases from 92.9% to 86.1%. We therefore use all the extracted features for classifier training and testing to achieve higher identification rate.

Attack on Nexus 7

Fig. 3.7 shows the identification rate of POWERFUL on a Google Nexus 7 with Android 4.4.4. As we can see, POWERFUL can correctly identify the 22 apps with high probabilities, and the identification rates using RF and SVM are similar and higher than that of using C4.5. In addition, the identification rates of most apps are higher than 80% for RF and SVM. The average identification rates of all the apps using C4.5, RF, and SVM are 83.7%, 92.9%, and 91.3%, respectively. In [37], the authors reported an overall inference accuracy of 93.96% on a smaller set of 13 apps. We believe that the performance of POWERFUL is similar to that of state-of-the-art solution.

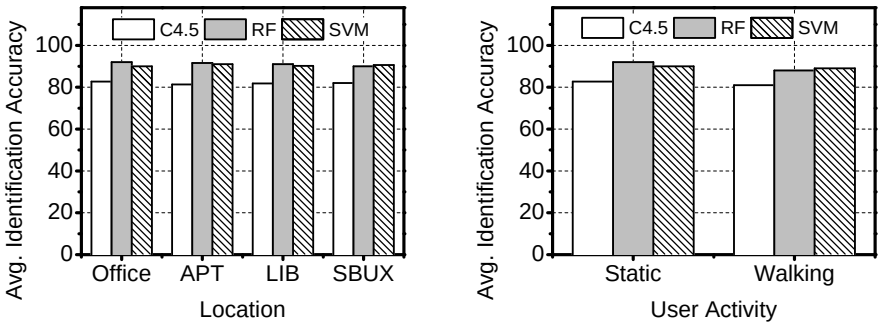
Attack on Nexus 6

Fig. 3.8 shows the identification rate of POWERFUL on a Google Nexus 6 with Android 5.1.1. We can see that the results are similar to those on a Google Nexus 7 with Android 4.4.4. The average identification rates of all the apps using C4.5, RF, and SVM are 84.45%, 91.3%, and 91.23%, respectively. These results confirm that POWERFUL can work with devices of different models.

Robustness

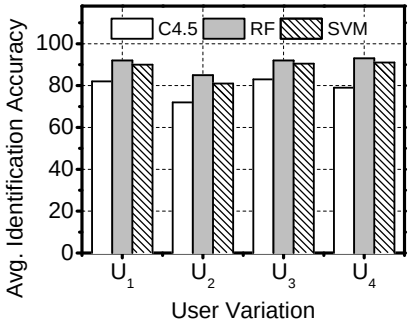
We also conducted a separate set of experiments to evaluate the robustness of POWERFUL to locations, user activities, and user variation.

Location. To evaluate the impact of locations, we let two participants to use the apps in Table 3.1 in four different locations, including our office, their apartments (APTs), the university library (LIB), and a Starbucks (SBUX) store, where Wi-Fi access is available. Each participant used each targeted app for five minutes in the same location with a Google Nexus 7. We then applied the trained classifiers to the



(a) Impact of location.

(b) Impact of user activity.



(c) Impact of user variation.

Figure 3.9: Performance of POWERFUL under different scenarios.

collected power profiles and obtained the average identification rate. Fig. 4.13 shows the average identification rate of POWERFUL with different locations. As we can see, the average identification rate is relatively stable across different locations. These results indicate that POWERFUL is robust to the change in location and thus can effectively fingerprint sensitive mobile apps even if the victim is at different locations. The main reason is that location has very limited impact on the power profiles and thus little impact on the classification results.

User Activity. We tested the performance of POWERFUL when users are conducting different activities. Specifically , we let two participants to use the apps in Table 3.1 while they were sitting statically in our office and walking slowly along the

corridor in our office building. Under each scenario, a participant used each targeted app for five minutes with a Google Nexus 7. We then applied the trained classifiers to the collected power profiles and obtained the average identification rate. Fig. 3.9b shows the average identification rate of POWERFUL under the two user activities. As expected, we can see that the average identification rates in the two scenarios are similar to each other. The reason is that slow user movement does not cause much change in the power profiles and thus has little impact on the classification results. Therefore, POWERFUL is robust to slow user movement.

User Variation. We let four participants each to use each app in Table 3.1 for five minutes in our office with a Google Nexus 7 and applied the trained classifiers to the collected dataset to obtain the average identification rate. Fig. 3.9c shows the average identification rate of POWERFUL on four users. As shown in the figure, the average identification rate of U_2 is lower than those of the other three users regardless of which machine learning algorithm is used. We conjecture that the variation is caused by different users having different usage patterns for the same app. For example, some users use Skype mostly for voice call and instant messaging while other users use it mostly for video call. In practice, the attacker can alleviate the impact of user variation by collecting more diverse dataset for classifier training. Nevertheless, POWERFUL can still achieve an average identification rate of 85% on U_2 using RF.

3.6 Conclusion

In this chapter, we presented the design and evaluation of POWERFUL, a novel attack framework on Android mobile device which combines power analysis and machine learning for mobile app usage inference. POWERFUL exploits the app-specific characteristics of the power profiles without requiring user permission. Our extensive experiments demonstrated that POWERFUL is able to infer the app being used at

a specific time with high accuracy, thus posing a realistic and serious threat to user privacy.

Chapter 4

SECURE MOBILE FACE AUTHENTICATION WITH RHYAUTH IS HIGHLY SECURE AGAINST VARIOUS ATTACKS

4.1 Overview

In this chapter, we propose FaceHeart [86], a novel and practical liveness detection scheme for securing face authentication on mobile devices. FaceHeart targets mobile devices with both front and rear cameras that are available on most recently shipped mobile devices. The key idea of FaceHeart is to check the consistency of two concurrent and independently extracted photoplethysmograms of the user as the live indicator. For this purpose, FaceHeart records a video of the user's face by the front camera and a video of the user's fingertip by the rear camera at the same time. Then FaceHeart applies photoplethysmography (PPG) to extract two underlying photoplethysmograms from the face and fingertip videos. If the two photoplethysmograms are from the same live person and measured at the same time, they must be highly consistent and vice versa. As photoplethysmograms are closely tied to human cardiac activity and almost impossible for the adversary to forge or control, the consistency level of two extracted photoplethysmograms can well indicate the confidence level in the liveness of a face authentication request.

We design a complete set of tools to check the consistency of two photoplethysmograms for liveness detection. Specifically, given the face or fingertip video, the corresponding photoplethysmogram is extracted as a time series according to the principle of PPG. As a result, two time series can be obtained by using similar computer vision tools. After that, a set of features such as estimated heart rates and cross

correlation of the two photoplethysmograms can be calculated by combining the two time series. Finally, lightweight machine learning algorithms are used for classifier training and subsequent testing. In our system, we adopt and compare three machine learning algorithms, i.e., Bayesian network (BN), logistic regression (LR), and multilayer perceptron (MLP), to demonstrate the feasibility of FaceHeart.

We also conduct extensive experiments to evaluate FaceHeart. 18 users from diverse background are involved in our experiments. In typical settings, FaceHeart achieves a true positive rate (TPR) as high as 97.5%, a false negative rate (FNR) as low as 5.2%, and an equal error rate (EER) as low as 5.98%. Furthermore, we study the impact of various factors on FaceHeart, such as the head pose, background illumination, and location. Overall, the experimental results confirm that FaceHeart can effectively and reliably defend against PFA and VFA and thus secure face authentication on mobile devices.

4.2 Background of Camera-Based PPG

In PPG, a photoplethysmogram is an optically obtained plethysmogram, which is a volumetric measurement of cardiovascular shock and sedation [87]. With each cardiac cycle, the heart pumps blood to the periphery, which generates pressure pulse that distends arteries and arterioles in the subcutaneous tissue. The corresponding volume change generated by the pressure pulse can be detected by measuring the amount of light either transmitted through or reflected from the skin. The evolvement of such volume changes across time carries exactly the user's heart beat signal.

We adopt the model in [88] for camera-based PPG-based heart rate measurements. When the incident light arrives at the user's skin, a major part gets reflected back by the skin surface and does not interact with the tissue underneath the skin. The remaining (minor) part of the incident light first penetrates underneath the skin

surface, then is absorbed by the tissue and the chromophores in blood inside arteries and capillaries, and finally gets reflected back to the camera. These two parts are usually referred to as surface reflectance and subsurface reflectance, respectively. The former dominates the overall light received by the camera but does not carry any information of human cardiac activity, while the latter is much smaller but bears the heart beat signal.

Given a skin region-of-interest (ROI) R in the video, the average pixel value at time t can be modeled as

$$y(t) = I(\alpha p(t) + b) + n(t), \tag{4.1}$$

in which $y(t)$ is the average pixel value, I is the incident light intensity in R, α is the strength of blood perfusion, $p(t)$ is the blood volume change pulse, b is surface reflectance from the skin in R, and $n(t)$ is the quantization noise of the camera. $\alpha p(t)$ denotes subsurface reflectance and is much smaller compared to b (i.e., $\alpha p(t) \ll b$). Normally, I can vary across R and may change significantly across time if the illumination source or the environment change across time. In our system, we assume I to be constant as the duration of the entire authentication process is usually less than five seconds and can be considered very short. Meanwhile, the user is asked to keep as still as possible, and we try to keep the environment, such as the illumination, as stable as possible. α and b are also assumed to be constants for the same ROI and the same user. On the contrary, $n(t)$ is a random variable, and a large variance of $n(t)$ may mask the small heart beat signal exhibited in $p(t)$. Equivalently, if noise is not considered, $y(t)$ can be viewed as the combination of a large DC part and a small AC part. The latter carries the information of human cardiac activity and can be extracted through a set of signal processing tools.

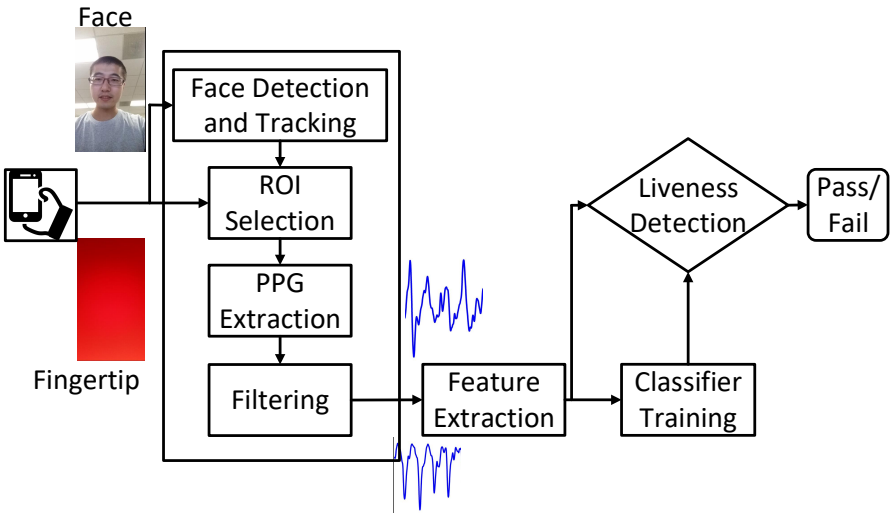


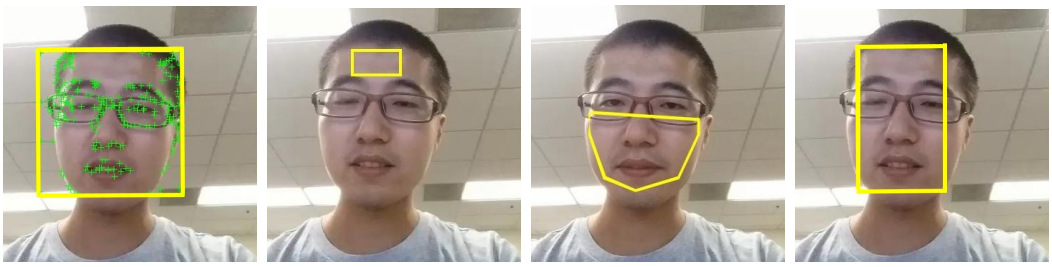
Figure 4.1: A system overview of FaceHeart.

4.3 FaceHeart

FaceHeart can be used as a standalone mobile authentication module in the mobile OS or integrated in any app desiring face authentication. In this section, we give an overview of FaceHeart and then detail its design.

4.3.1 Overview

FaceHeart works as follows. First, the user uses his/her fingertip to cover the rear camera and also flashlight without applying any pressure. Then FaceHeart uses the front and rear cameras simultaneously to record the face and fingertip videos, respectively. The user needs to stay as still as possible while the recording is ongoing. Next, FaceHeart extracts two photoplethysmograms from the two videos and compares them for liveness detection. In the meantime, one frame of the face video (for instance, any frame after the first second of recording) is sent to the conventional face authentication module to decide whether the person in the frame is the legitimate



(a) Detected face and "good features" (b) R_1, forehead [89] (c) R_2, lower face [90] (d) R_3, central face [91]

Figure 4.2: Camera-based PPG.

user. Only when liveness detection and conventional face authentication both succeed is the user considered authentic.

Fig. 4.1 depicts the flow chart of FaceHeart. Given a pair of face and fingertip videos, FaceHeart uses the following modules to accomplish liveness detection. The Signal Processing module is first invoked to obtain two photoplethysmograms independently from the two videos. Then the output is fed into the Feature Extraction module to generate a feature vector which characterizes the consistency level of the two photoplethysmograms. In the next Classifier Training module, machine learning algorithms are used to train a classifier based on a library of feature vectors. Finally, the classifier is used in the Liveness Detection module to determine whether a new pair of face and fingertip videos can pass liveness detection.

4.3.2 Signal Processing

As shown in Fig. 4.1, the Signal Processing module comprises four submodules: face detection and tracking, ROI (region-of-interest) selection, photoplethysmogram extraction, and filtering. The face video requires all four submodules, while the fingertip video just needs the last three.

Face detection and tracking

In this step, we first detect the user's face in the first frame of the face video using the classical Viola-Jones detection algorithm [92]. This algorithm can work in real time and is highly accurate.

Next, instead of applying relatively costly face detection to every frame, we use the Kanade-Lucas-Tomasi (KLT) feature tracker to track the identified features from frame to frame [93, 94]. More specifically, the KLT feature tracker identifies multiple local feature points, commonly known as "good features to track" [95]. Then it tries to search as many as possible of the identified feature points in the previous frame. Given two sets of features points in the current and previous frame, the KLT feature tracker can estimate the translation, rotation, and scale between the two consecutive frames and then compute an affine function for face tracking. Since the duration of the face video is short, the established feature tracker is still valid for the last frame.

Finally, we can obtain the coordinates of the user's face in each frame. As depicted in Fig. 4.2a, we obtain four coordinates forming a rectangular box in each frame, which approximates the whole face region. The green cross markers depict the "good features to track" of the shown frame.

ROI selection

Different types of ROIs have been used in the literature. Fig. 4.2b, Fig. 4.2c, and Fig. 4.2d illustrate three most frequently used ROIs, denoted by R_1 [89], R_2 [90], and R_3 [91], respectively. Some schemes use random selection while some others assign weights to every segmented unit of the face. Intuitively, the amount of photoplethysmogram information extracted from a specific ROI is closely related to where the ROI is. The reason is that the extracted photoplethysmogram is proportional to $p(t)$ in

Eq. (4.1), i.e., the amount of blood volume change underneath the ROI. Meanwhile, the distribution of blood carrying capillaries differs from region to region, further resulting in different amount of extractable photoplethysmogram information. The size of the selected ROI may also have influence on the extracted photoplethysmogram. On the one hand, a smaller size requires a highly accurate face tracker to avoid too much noise in the extracted photoplethysmogram. On the other hand, a larger size averages the contribution across the entire region and therefore may shrink the strength of the photoplethysmogram.

In our system, we choose R_3 as the ROI for extracting photoplethysmogram, which is the central part of the whole face and encompasses 60% of the width and the full height of the detected face region. In contrast to R_1 and R_2 that require a resource-demanding feature detector [96], R_3 only requires the basic computationally efficient Viola-Jones detector. In addition, our experimental evaluations in Section 4.4.4 show that R_1 and R_2 do not show much performance improvement over R_3 mainly because the required face tracker has limited accuracy in constrained mobile environments. It is possible to have a weighted combination of multiple ROIs as in [88], which nevertheless requires multiple iterations and thus incurs larger computation overhead. How to use multiple ROIs more efficiently in FaceHeart is part of our future work.

Photoplethysmogram extraction

We extract the photoplethysmogram from an ROI by averaging all pixel values therein. A recorded video has three channels: red, green, and blue. In the literature [97, 91, 90, 98, 88], it is widely accepted that the three channels carry different amount of photoplethysmogram information. The green channel carries the strongest photoplethysmogram, as the green light is easier to absorb by hemoglobin in the blood and thus penetrates deeper into the skin [88]. It is tempting to use all three channels

to enhance the SNR of the extracted photoplethysmogram, but the recent studies [97, 90, 88] show that this approach is not necessarily beneficial because the three channels do not yield statistically mutually independent information. So we follow the suggestion in [97, 90, 88] to obtain the photoplethysmogram only from the green channel.

Filtering

This step applies two filters to the extracted photoplethysmogram. First, we use a Normalized Least Mean Square (NLMS) adaptive filter to alleviate the illumination interference [99]. The motivation is that small environment changes—such as a person passing by or small camera movements—may induce overall illumination shifting in the video. This undesirable effect can be mitigated by estimating the amount of interference and then subtracting it from the overall measurement. In Section 4.2, we use $y(t)$ to denote the photoplethysmogram of a selective ROI R. Given the illumination interference, $y(t)$ can be divided into two parts:

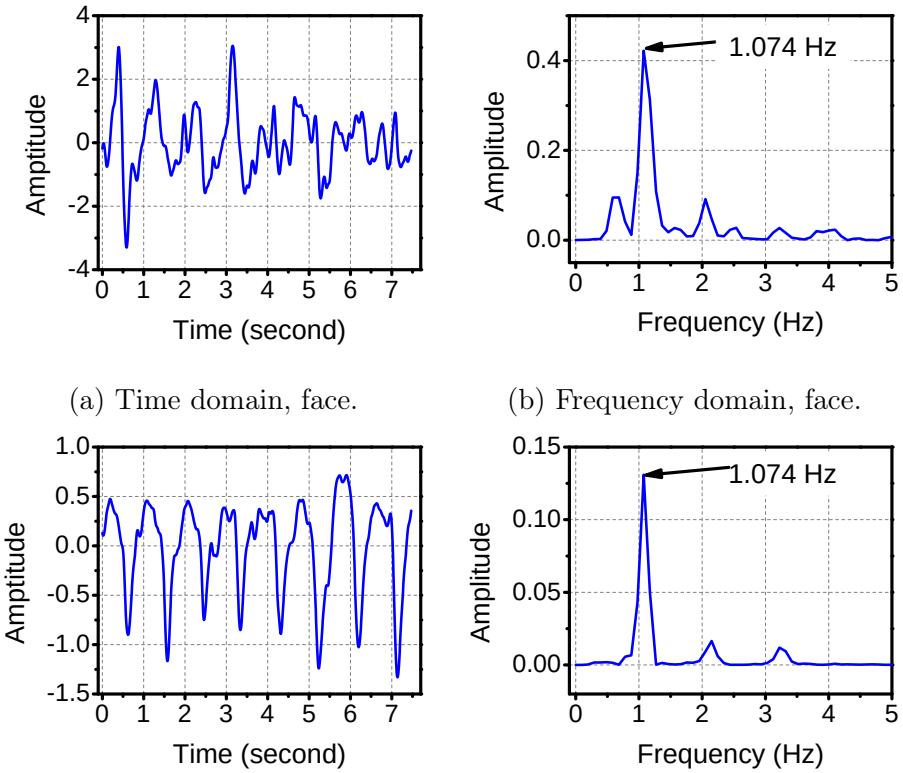
$$y(t) = y_c(t) + n_i(t), \tag{4.2}$$

where $y_c(t)$ is due to human cardiac activity, and $n_i(t)$ is due to illumination interference. $n_i(t)$ can be assumed to be proportional to the average pixel value of the background regions other than the face region. We thus have

$$n_i(t) = h y_{bg}(t), \tag{4.3}$$

where $y_{bg}(t)$ is the average pixel value of a selective background region, and h is a linear coefficient. In our implementation, we simply select a pixel block of 20×20 in the top-right corner in each frame as the background region. h can be estimated by the NLMS adaptive filter as

$$h(j+1) = h(j) + \mu \frac{y_c(j)}{y_{bg}(j)}, j = 0, 1, 2, \ldots, N-1. \tag{4.4}$$



(a) Time domain, face.

(b) Frequency domain, face.

(c) Time domain, fingertip.

(d) Frequency domain, fingertip.

Figure 4.3: Illustration of extracted photoplethysmograms.

Here μ is the step size equal to 1, and N is the length of $y(t)$ (or $y_c(t)$, equivalently). We also set $h(0) = 0$ in the implementation. After the final $h = h(N)$ is obtained, $n_i(t)$ can be subtracted from $y(t)$ according to Eq. (4.2) to finally reveal $y_c(t)$.

Next, we use a bandpass FIR filter (second-order Butterworth filter) with a pass-band of $[0.7, 4]$ Hz to reduce the interference of out-of-band noise. The signal after filtering is the final photoplethysmogram for liveness detection.

Processing fingertip video

Extracting the photoplethysmogram from a fingertip video is much easier. Specifically, no face detection or tracking is needed, and the entire frame is used as the ROI.

Meanwhile, since the rear camera is fully covered by the user's fingertip, there is no illumination interference so that the NLMS adaptive filter is not needed.

4.3.3 Feature Extraction

In this module, we use the two extracted photoplethysmograms to calculate a feature vector for classifier training and liveness detection. Denote the photoplethysmograms from the face and fingertip videos by P_{face} and P_{ftip}, respectively. P_{face} and P_{ftip} are two time series of the same length N, from which the following features are calculated.

- **Heart rate difference**. The heart rate difference is the absolute difference between the heart rates from the face and the fingertip. We denote them by h_{face} and h_{ftip}, respectively. To obtain h_{face}, we first multiply P_{face} with an N-point Hanning window such that the two endpoints of P_{face} can meet rather than having a sharp transition between them. Then we apply fast fourier transform (FFT) on windowed P_{face}, select the highest peak within $[0.7, 4]$ Hz, multiply it by 60, and obtain h_{face}. We can also obtain h_{ftip} in the same way. Then heart rate difference is calculated as

$$\Delta h = |h_{\text{face}} - h_{\text{ftip}}| \tag{4.5}$$

- **Maximum cross correlation**. We obtain the maximum cross correlation between P_{face} and P_{ftip} by searching the optimal alignment between them. Specifically, we first obtain the optimal alignment $\hat{k}$ by the following equation.

$$\begin{aligned} &\hat{k} = \arg\min \sum_{i=1}^{N-k+1} \frac{P_{\text{face}}(i) P_{\text{ftip}}}{N-k}, \\ &\text{subject to } 0 \leq k < N_{\text{ftip}}. \end{aligned} \tag{4.6}$$

Here N_{ftip} is the approximate length of a period of P_{ftip} and equals $\lceil \frac{60F_s}{h_{\text{ftip}}} \rceil$, where F_s is the frame rate of the fingertip video (and equivalently that of the face video). After $\hat{k}$ is found, we truncate P_{face} and P_{ftip} into two shorter vectors of the same length as

$$\tilde{P}_{\text{face}} = P_{\text{face}}(1 : N - \hat{k}), \tilde{P}_{\text{ftip}} = P_{\text{ftip}}(\hat{k} + 1 : N). \tag{4.7}$$

Then the maximum ratio is calculated as

$$\rho_{\max} = \sum_{i=1}^{\tilde{N}} \frac{\tilde{P}_{\text{face}}(i)\tilde{P}_{\text{ftip}}(i)}{\tilde{N}}, \tag{4.8}$$

where $\tilde{N} = N - \hat{k}$.

- **Mean, min, max, and standard deviation of amplitude ratio**. Given the aligned $\tilde{P}_{\text{face}}$ and $\tilde{P}_{\text{ftip}}$, we first calculate amplitude ratio as $R(i) = \frac{\tilde{P}_{\text{face}}(i)}{\tilde{P}_{\text{ftip}}(i)}, i = 1, 2, \ldots, \tilde{N}$. Then we further calculate the mean, min, max, and standard deviation of R as features, denoted by $R_{\text{mean}}, R_{\min}, R_{\max}$, and R_{SD}, respectively.

4.3.4 Classifier Training

Our training set contains two classes of instances. Each instance consists of a feature vector in the form of $v = [\Delta h, \rho_{\max}, R_{\text{mean}}, R_{\min}, R_{\max}, R_{\text{SD}}]$. The feature vectors of the instances in Class I (labelled as $l = 1$) are computed from a pair of simultaneously recorded face and fingertip videos. On the contrary, those of the instances in Class II (labelled as $l = 0$) are computed from a pair of face and fingertip videos recorded separately. Ideally, the classifier should be able to label the instances in both classes as accurately as possible. As in [49], we use and compare three supervised machine learning techniques in the Weka toolkit [83] for classifier training and testing: Bayesian network (BN), logistic regression (LR), and multilayer perceptron

(MLP). In particular, BN is based on constructing a probabilistic graphic model representing a set of random variables and their conditional dependencies via a directly acyclic graph [100]. The constructed probabilistic model is used to infer the label of unlabeled instances. LR uses the sigmoid function as the hypothesis to estimate the relationship between the features and corresponding labels [101]. MLP is a feedforward artifical neural network model that maps the sets of input data onto a set of appropriate output [102]. One important advantage of MLP is that it can be used to distinguish data that are not linearly separable.

The classifier training is neither user-specific nor device-specific. It is exclusively done by the FaceHeart developer who can easily maintain and update a large number of instances for Classes I and II. The trained classifier is preloaded into the mobile device when FaceHeart is installed.

4.3.5 Liveness Detection

Given a new pair of face and fingertip videos for authentication, FaceHeart computes the corresponding feature vector and then inputs into the classifier. If the output label is 1, the new pair passes liveness detection and fails otherwise. In the former case, if the face image additionally passes conventional face authentication, the user is deemed legitimate.

4.4 Performance Evaluation

This section evaluates the performance of FaceHeart.

4.4.1 Adversary Model

We consider a typical adversary model. The adversary possesses the victim's mobile device and seeks to pass the face authentication employed by the device it-

self or some sensitive apps. Since VFA can be considered an advanced version of PFA, we focus on evaluating the resilience of FaceHeart to VFA. The adversary can surreptitiously obtain the videos containing the legitimate user's frontal face, e.g., by online searches or realtime capturing through a high-definition camcorder from a long distance. In contrast, fingertip videos are very rare online or almost impossible to capture in real time, so the adversary can only use the fingertip video of himself or a random user. In addition, the adversary is fully aware of FaceHeart. We consider two types of VFA as follows.

Type-I VFA. This attack does not involve any realtime video recording and serves as a "stress test" for FaceHeart. In particular the adversary directly feeds his fingertip video and the victim's face video into FaceHeart. Each participant in our experiments is assumed as the adversary once, in which case the other participants are used as the victims.

Type-II VFA. This attack resembles the practical attack scenario. The adversary first replays the victim's face video on the screen of his/her own device such as an iPad. The distance between the victim device and the adversary's device screen is properly adjusted such that the victim device's front camera can well capture the victim's face in the replayed video. While the face video is replayed and recorded, the adversary let the victim device's rear camera take his/her fingertip video simultaneously. Two random participants are chosen as the adversary for the Type-II VFA. When either is chosen, each other participant serves as a victim.

4.4.2 Experiment Setup

We used a Samsung Galaxy S5 in the experiments. In particular, we utilized the dual-camera mode of the Camera app on Galaxy S5, which can record a video with both the front and rear cameras simultaneously. The frame size of the recorded

video is 720×1280, which can be equally divided into the upper and lower parts, corresponding to the face and fingertip videos, respectively. After the useless black region on left and right sides is removed, the frame size of both face and fingertip videos becomes 480×640. Since almost all recently shipped mobile devices have both front and rear cameras, it is rather straightforward to obtain the simultaneously-recorded face and fingertip videos on other device models.

We recruited 18 participants in the experiments, including two females and 16 males. The participants are graduate students in Arizona State University, whose ages range between 20 and 35. All the participants were given the following instructions. First, each participant tries to sit as still as possible. The distance between the user and the front camera varies between 30 to 45 cm, which has been proved to be a convenient distance for the users and that the captured user face is reliably detected. Then s/he activates the dual-camera mode of the Camera app on Galaxy S5 and ensures that the front camera properly captures her/his frontal face. Subsequently, s/he rests any of her/his fingertip on the rear camera without applying any pressure. Finally, s/he proceeds to record a video of approximately ten seconds by tapping the video recorder icon.

As cardiac activity highly depends on current user conditions, the videos were recorded when the participant was under different conditions to fully evaluate the performance of FaceHeart. In particular, we investigated three user conditions. Under the rest condition, each participant was asked to sit quietly without her/his legs crossed for five minutes. After that, s/he recorded videos for 15 times. Under the reading condition, each participant was asked to read recent news on a smartphone for five minutes. After that, s/he recorded videos for 15 times. Under the gaming condition, each participant was asked to play the video game "No Limits" or "Strikers 1945-3" on a smartphone for five minutes. After that, s/he recorded videos for 15

times. For the same participant, cardiac activities are expected to be different under these three conditions [103]. Particularly, the heart rate of the same user in the gaming condition is usually higher than those in the rest and reading conditions, which was also confirmed in the experiments.

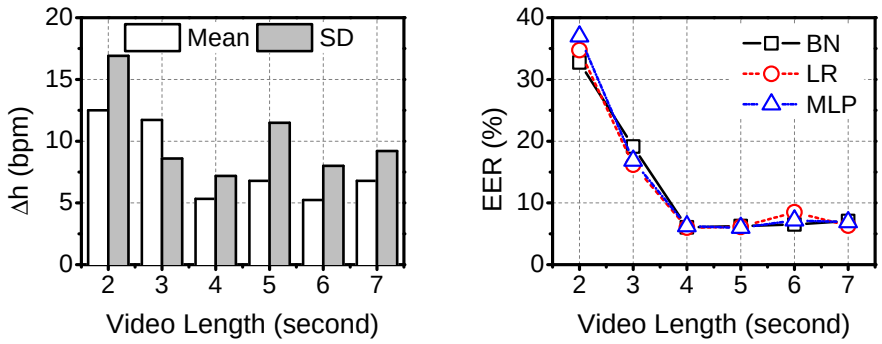
The following default settings were used unless stated otherwise. Participants were asked to maintain the front head pose during video recording. Videos were recorded under normal illumination in a typical research lab (e.g., 500 lux). During the recording process, other persons may leave/enter the lab.

Our main dataset, denoted by $\mathcal{S}$, consists of $\mathcal{S}_p$ for positive (Class I) instances and $\mathcal{S}_n$ for negative (Class II) instances. The instances in $\mathcal{S}_p$ come from legitimate users, while those in $\mathcal{S}_n$ are from Type-I adversary. Given 18 participants with each recording 15 videos under each of the three user conditions, there are $18 \times 3 \times 15 = 810$ instances in $\mathcal{S}_p$. To generate $\mathcal{S}_n$, we first randomly selected two pairs of face and fingertip videos for each participant. Each participant acted as the adversary once, in which case each other participant acted as the victim. So $\mathcal{S}_n$ contains $2 \times 2 \times 17 = 68$ instances per participant and $68 \times 18 = 1224$ instances in total. For the following evaluations, we repeated the generation process of $\mathcal{S}_n$ for 40 times and obtained the average results.

4.4.3 Performance Metrics

We use the following performance metrics.

Receiver operating characteristic (ROC) curve. An ROC curve can be used to illustrate the performance of a binary classifier as its discrimination threshold changes. According to the definition in [104], we can obtain an ROC curve by plotting TPR (true-positive rate) with respect to FPR (false-positive rate) in various threshold settings.



(a) On difference between h_{face} and h_{ftip}

(b) On EER

Figure 4.4: Impact of video length on Δh and EER.

Acceptance rate. We define the acceptance rate as the ratio between the number of correctly-classified positive (legitimate) instances and that of all positive instances in a testing dataset. A higher acceptance rate means that the system is more likely to admit legitimate users.

Detection rate. We define the detection rate as the ratio between the number of correctly-classified negative (adversarial) instances and that of all negative instances in a testing dataset. A higher detection rate means that the system can more effectively detect VFA.

Computation time. We define the computation time as the time FaceHeart takes to determine whether a given pair of face and fingertip videos can pass liveness detection. Intuitively, the computation time should be as short as possible.

4.4.4 Experimental results

Video length

Here we show the impact of video length on FaceHeart.

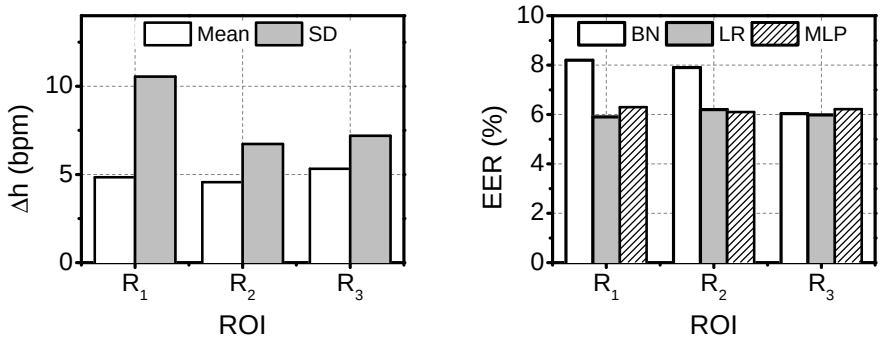
Fig. 4.4a shows the mean and standard deviation (SD) of Δh in $\mathcal{S}_p$, which is the absolute difference between h_{face} and h_{ftip} in the same authentication session. Since the SNR of the photoplethysmogram from the fingertip video is usually high, h_{ftip} can be treated as the reference heart rate. As we can see, the mean and SD of Δh decrease from around 12 and 17 bpm to around 5 and 7 bpm when the video length increases from two to four seconds. This means that the accuracy of h_{face} increases along with the video length. When the video length is larger than four seconds, the mean and SD of Δh do not change much.

Fig. 4.4b shows the EER (equal error rate) of FaceHeart under the Type-I attack using $\mathcal{S}$. We can see that FaceHeart exhibits similar EER performance with BN, LR, and MLP. Therefore, we believe that FaceHeart works well along with mainstream machine learning algorithms. Meanwhile, the EER decreases quickly when the video length increases from two to four seconds and then stays relatively the same as the video length further increases. Such results are consistent with those in Fig. 4.4a because a smaller Δh indicates that the two corresponding photoplethysmograms in the same authentication session are more consistent. Consequently, this makes it easier for the classifier to distinguish between positive and negative instances, leading to a lower EER.

As a shorter video length means that the legitimate user can record a shorter video for authentication, the required minimum video length of FaceHeart is preferably as short as possible. Based on the above results, the default video length is set to four seconds hereafter unless specified otherwise.

ROI

Now we demonstrate the impact of ROI on FaceHeart using $\mathcal{S}$.



(a) On difference between h_{face} and h_{ftip}

(b) On EER

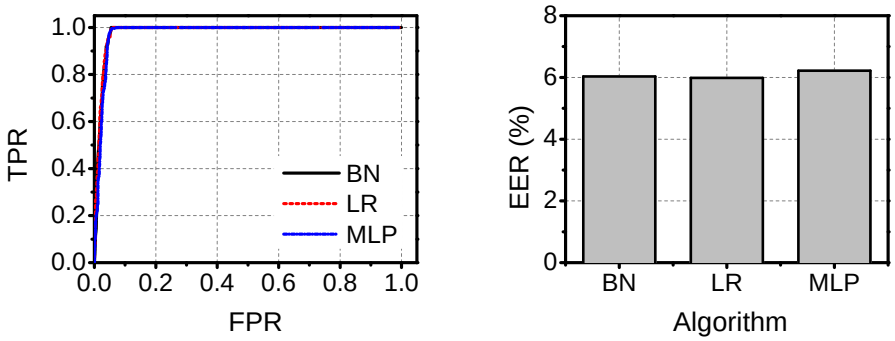
Figure 4.5: Impact of ROI on Δh and EER.

Fig. 4.2b, Fig. 4.2c, and Fig. 4.2d illustrate the three ROIs to study. Fig. 4.5a shows the mean and SD of Δh in $\mathcal{S}_p$. As we can see, the means of Δh using R_1, R_2, and R_3 are 4.84, 4.56, and 5.32 bpm, respectively, and the SDs are 10.55, 6.73, and 7.19 bpm, respectively. Fig. 4.5b shows the corresponding EERs when R_1, R_2, and R_3 are used as the selected ROI, respectively. The EERs with R_1 using BN, LR, and MLP are 8.2%, 5.9%, and 6.3%, respectively, those with R_2 are 7.9%, 6.2%, and 6.1%, respectively, and those with R_3 are 6.0%, 6.0%, and 6.2%, respectively.

The results above show that the three ROIs lead to similar EER performance while the EERs with R_3 are slightly better than those with R_1 or R_2. More importantly, the computation time of FaceHeart using R_3 as the selected ROI is much shorter than that using R_1 or R_2, as shown soon in Section 4.4.4. Therefore, we select R_3 as the ROI for photoplethysmogram extraction by default.

Type-I attack

Here we show the resilience of FaceHeart to the Type-I attack.



(a) ROC curve (b) EER

Figure 4.6: ROC and EER performance of FaceHeart under Type-I attacks.

Fig. 4.6a and Fig. 4.6b show the ROC curve and EER of FaceHeart, respectively. The TPRs using BN, LR, and MLP are 90.2%, 97.5%, and 94.6%, respectively, the FPRs are 3.8%, 5.2%, and 4.6%, respectively, and the EERs are 6.03%, 5.98%, and 6.21%, respectively. The results show that the performance of FaceHeart is similar to those of the state-of-the-art systems, such as FaceLive in [49]. To sum up, FaceHeart can achieve very high TPR and very low FPR at the same time, meaning that it can correctly distinguish between legitimate requests and VPAs with high probability.

Fig. 4.6b shows the EERs of FaceHeart in different user conditions. The EERs using BN, LR, and MLP under the rest condition are 7.70%, 5.57%, and 5.40%, respectively, those under the reading condition are 8.77%, 5.53%, 5.73%, respectively, and those under the gaming condition are 8.27%, 8.54%, and 5.65%, respectively. Overall, the EERs in the three user conditions are low, so FaceHeart can be used even when the user's cardiac activity changes. In addition, the EERs in the gaming condition are slightly higher than those under the other conditions. This is anticipated because the heart rate in the gaming condition is usually higher than others so that the SNR of the extracted photoplethysmogram usually decreases due to the increased noise level in the higher frequency range. Therefore, the consistency be-

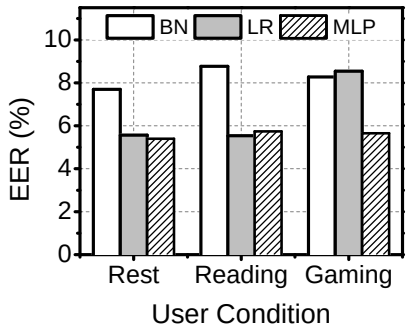


Figure 4.7: EER performance of FaceHeart under Type-I attacks in different user conditions.

tween the two photoplethysmograms from a pair of face and fingertip videos in the same authentication session drops, leading to a higher EER. Based on $\mathcal{S}$, we obtain the corresponding classifiers with BN, LR, and MLP, respectively, by using 10-fold cross validation for training. Then we use the trained classifier models for testing in the following.

Type-II attack

Now we show the detection rate of FaceHeart under the Type-II attack. We first obtained the negative (adversarial) instances for the Type-II attack as follows. Two of the 18 participants acted as the adversaries. For each adversary, the other 17 participants were regarded as her/his victims. For each victim, we randomly selected 10 face videos from her/his recordings. Then the two adversaries launched the Type-II attack, resulting in $2 \times 10 \times 17 = 340$ negative instances. After that, we applied the trained classifiers in Section 4.4.4 to the collected negative instances and obtained the detection rate. As shown in Fig. 4.8, the detection rates using BN, LR, and MLP are 94.71%, 97.94%, and 98.24%, respectively, indicating that FaceHeart can detect VFA with overwhelming probability.

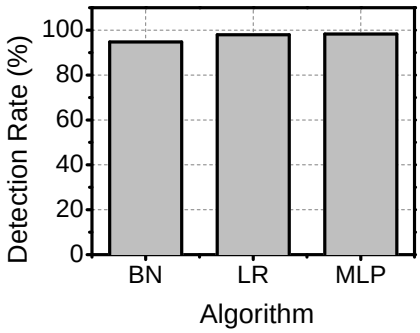


Figure 4.8: EER performance of FaceHeart under Type-II attacks.

Robustness of FaceHeart

In the following, we study the robustness of FaceHeart against different factors including head pose, illumination, and location.

Head pose. We first study the impact of head pose on the acceptance rate of FaceHeart. As illustrated in Fig. 4.9 [105], the relative rotation of a user's head to the front head pose can be described by rotation angles in three independent axes, which are yaw, pitch, and roll, respectively. Hereafter we also refer to the rotation angles in yaw, pitch, and roll axes as yaw, pitch, and roll, respectively. For the front head pose, yaw, pitch, and roll are equal to zero. Roll is easier to adjust by the user, and a zero roll also benefits face detection. So participants were asked to adjust their head poses such that the rolls are as near to zero as possible. As a result, we only focus on the other two types of head rotation angles, i.e., yaw and pitch.

Data collection worked as follows. First, we asked two participants to record videos for authentication with different yaws or pitches. Specifically, they recorded videos when the yaws changed and the pitches remained near to zero and continued when the pitches changed and the yaws remained near to zero. After that, we applied the trained classifiers in Section 4.4.4 to the collected dataset and obtained the acceptance

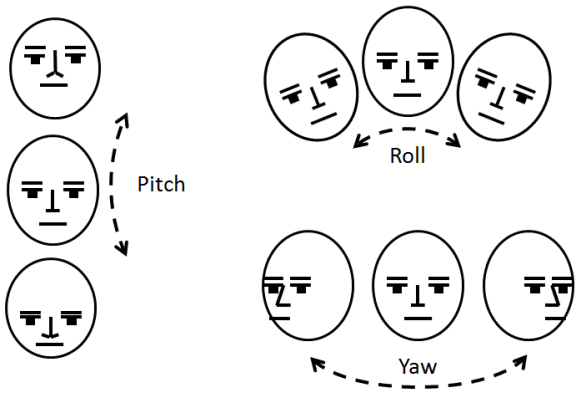
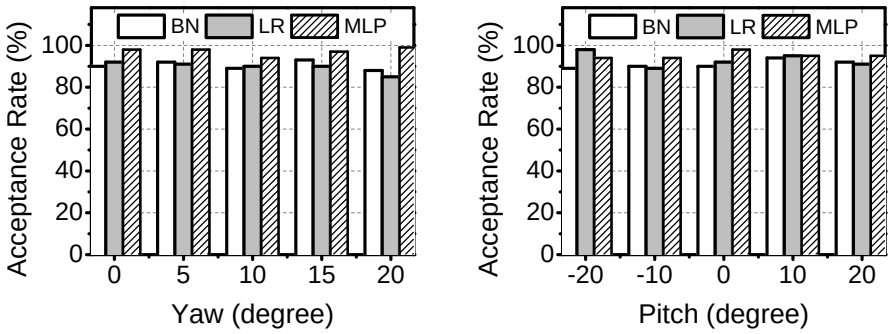


Figure 4.9: Illustration of head pose in yaw, pitch, and roll axes.

rate. Each participant recorded 50 videos for the same yaw or pitch, resulting in 1,000 videos in total.

Fig. 4.10a and Fig. 4.10b show the acceptance rates of FaceHeart with different yaws and pitches, respectively. The acceptance rate is almost always higher than 90% and changes only slightly when the yaw of user head pose changes from zero to 20 degrees, or the pitch changes from -20 to 20 degrees. The results are as expected because FaceHeart is based on comparing two photoplethysmograms extracted from a pair of face and fingertip videos, and a small yaw or pitch (less than $\pm$20 degrees) does not affect photoplethysmogram extraction much. Assuming that users tend to record videos with small yaws or pitches (less than $\pm$10 degrees) in practice, we believe that FaceHeart is robust to head pose changes.

Illumination. Here we study the impact of illumination on the acceptance rate of FaceHeart. For this experiment, we asked two participants to record videos for authentication under two different illuminations, i.e., normal (in the range of hundreds lux) and low illuminations (less than 20 lux). Fig. 4.11 illustrates the clear influence of normal and low illuminations on video recording. The illumination was adjusted by turning off part of the lights in our office. After that, we applied the trained



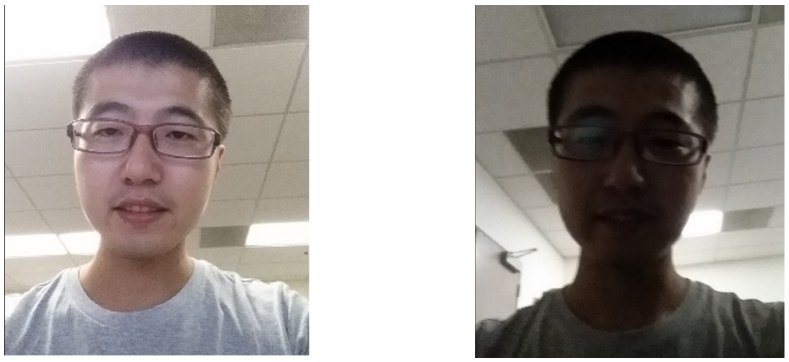
(a) Rotate in yaw axis (b) Rotate in pitch axis

Figure 4.10: Impact of head pose on acceptance rate.

classifiers in Section 4.4.4 to the collected dataset and obtained the acceptance rate. Each participant recorded 50 videos for the same illumination, resulting in 200 videos in total for this experiment.

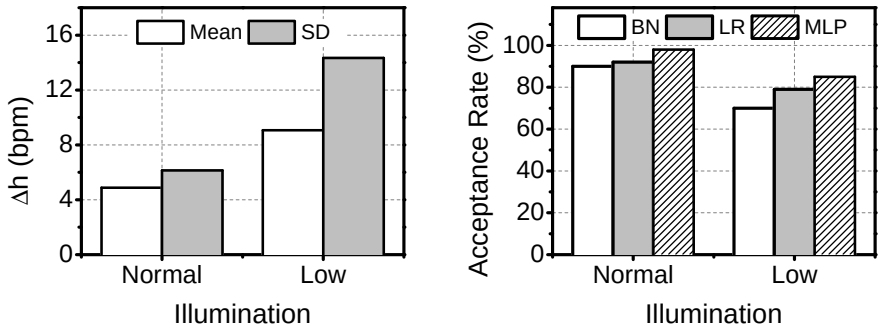
Fig. 4.12a and Fig. 4.12b show the mean and SD of Δh and acceptance rate of FaceHeart, respectively. The mean and SD of Δh increase from 4.88 and 6.14 bpm to 9.07 and 14.34 bpm, respectively, when the illumination switches from normal to low. Correspondingly, the acceptance rates using BN, LR, and MLP drop from 90%, 92%, and 98% to 70%, 79%, and 85%, respectively. The results indicate that FaceHeart is greatly affected by illumination in the environment, which can be explained as follows. FaceHeart relies on comparing the photoplethysmograms extracted from a pair of face and fingertip videos, and low illumination leads to a low SNR of the extracted photoplethysmogram. Hence, the consistency between the face and fingertip photoplethysmograms reduces (partially illustrated by the increased Δh), leading to the decreased acceptance rate.

Location. We also study the impact of locations on the acceptance rate of FaceHeart. First, we asked two participants to record videos for authentication in four different locations, i.e., our office, the apartments (APTs) of the participants, the



(a) Normal illumination (b) Low illumination

Figure 4.11: Captured images under different illuminations.



(a) On difference between h_{face} and h_{ftip} (b) On acceptance rate

Figure 4.12: Impact of illumination on Δh and acceptance rate.

university library (LIB), and an outdoor bench on our campus. After that, we applied the trained classifiers in Section 4.4.4 to the collected dataset and obtained the acceptance rate. Each participant recorded 50 videos for the same location, resulting in a dataset of 400 videos in total.

Fig. 4.13 shows the acceptance rate of FaceHeart with different locations. The acceptance rates are always higher than 90% and do not change much when the location changes. The results indicate that FaceHeart is robust to location changes and thus

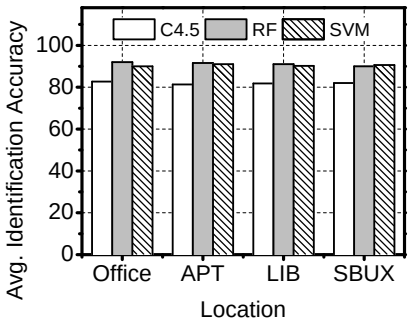


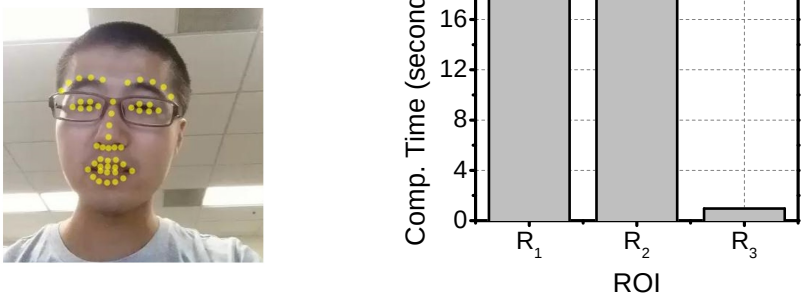
Figure 4.13: Impact of location on acceptance rate.

can be used in different locations. The reason is that locations have little impact on photoplethysmogram extraction and consequently little impact on the classification results.

Computation time

Here we study the computation time of FaceHeart for different ROIs. For this experiment, we randomly select 100 pairs of face and fingertip videos from our collected data. Each pair of videos were both chopped to a length of four seconds. Then we run FaceHeart with the given video pairs and obtained the average computation time. To use R_1 or R_2, we first used the face tracker in [96] to track the facial landmarks in each frame and then calculated the coordinates of R_1 or R_2. Fig. 4.14a depicts the tracked 49 landmarks on the user face which are used for the calculation of R_1 and R_2.

Fig. 4.14b shows the computation time using R_1 or R_2 or R_3 as the selected ROI. The average computation time using R_1, R_2, and R_3 are 18.05, 18.19, and 0.96 seconds, respectively. Therefore, selecting R_3 as the ROI for photoplethysmogram extraction is much faster than selecting R_1 or R_2. Such results are as expected



(a) Heart rate difference. (b) Acceptance rate.

Figure 4.14: Impact of ROI on computation time.

because R_1 and R_2 require much more computationally-expensive face trackers than that used by R_3.

The computation time of FaceHeart is comparable to the state of art. In particular, Li *et al.* reported an average time of 3.3 seconds for device movement (equivalent to the video length in FaceHeart) in [49] and did not explicitly evaluate the computation time for liveness detection. In [48], the authors mentioned that the average authentication time for video recording and also liveness detection is 2.8 seconds when successful and failed authentications are combined and 4.9 seconds when only successful authentications are considered. Given the video length of four seconds used in our evaluations, we believe that the computation time of FaceHeart is similar to the state-of-the-art, but FaceHeart is more secure and user-friendly.

4.5 Discussion

As the first system exploring photoplethysmogram for secure face authentication on mobile devices, FaceHeart certainly has limitations. In this section, we outline the possible ways to further improve FaceHeart.

4.5.1 Camera-based PPG

As the camera-based PPG method in [91] is adopted to extract photoplethysmograms, FaceHeart naturally inherits its limitations related to user movement and the environment illumination. More specifically, the user is required to keep her/his head as still as possible in order to extract more accurate photoplethysmograms. Meanwhile, as shown in Section 4.4.4, the performance of FaceHeart depends greatly on the illumination in the environment. Hence, there should be sufficient and stable illumination in the environment to guarantee the high performance of FaceHeart.

Advanced schemes have been explored to alleviate the requirements on user movement and the environment illumination. For example, researchers have proposed schemes to improve the estimation accuracy of the heart rate under adverse situations, such as when the user spontaneously moves his head a little bit [88] or the illumination in the environment is below normal [106]. Although such schemes are not directly applicable to FaceHeart, they indicate a promising direction worth exploring. Other minor issues inherited from camera-based PPG methods include the impact of facial occlusion, facial expression, and user skin tone, which we plan to fully investigate in our future work.

4.5.2 Authentication time

In FaceHeart, the authentication time for liveness detection can be broken into two parts, i.e., video length and computation time. Given the video length of four seconds and the computation time of 0.96 seconds with R_3 as the ROI, the total authentication time of FaceHeart is around 4.96 seconds. In [48], the authors reported that the authentication time of their liveness detection scheme is around 4.9 seconds, which is comparable to 4.3 seconds of credential-based authentication schemes. In

this regard, the authentication time of FaceHeart is acceptable and also comparable to the state-of-the-art.

Similar to [48, 49], the authentication time of FaceHeart is dominated by the required video length, which is four seconds in our system. A shorter video length may be adopted, however, at the cost of higher EERs. One possible way to shorten the required video length is to extract new features from extracted photoplethysmograms. For example, heart rate variability and the absolute delay between the two photoplethysmograms from face and fingertip videos are very promising candidates. These two features can be useful only when the SNRs of the two photoplethysmograms are sufficiently high, which we plan to explore in the future.

4.6 Conclusion

In this chapter, we presented the design and evaluation of FaceHeart, a novel and practical scheme for liveness detection to secure face authentication on COTS mobile devices. FaceHeart relies on the non-forgeability of the photoplethysmograms extracted from two videos simultaneously taken through the front and rear cameras on a mobile device. Extensive user experiments confirmed that FaceHeart can effectively thwart photo-based and video-based forgery attacks on mobile face authentication systems.

Chapter 5

EYETELL: VIDEO-ASSISTED TOUCHSCREEN KEYSTROKE INFERENCE FROM EYE MOVEMENTS

5.1 Overview

In this chapter, we report the design and evaluation of EyeTell [107], a novel video-assisted keystroke inference attack that can infer a victim's keystrokes on his touchscreen device from a video capturing his eye movements. EyeTell is inspired by the observation that human eyes naturally focus on and follow the keys they type such that a typing sequence on a soft keyboard results in a unique gaze trace of continuous eye movements. Under EyeTell, the attacker records a video of the victim's eye movements during his typing process and then extracts a gaze trace. By analyzing the gaze trace, the attacker can infer the victim's input with high accuracy.

Although conceptually intuitive, EyeTell faces three main design challenges. First, it needs to extract a gaze trace from the recorded video without any prior information about the victim (e.g., what his eyes look like). Second, the gaze trace is usually very noisy, making it very difficult to recover the correct typing sequence. Third, the gaze trace does not tell the exact number of keystrokes on the soft keyboard. To tackle the first challenge, we explore a user-independent model-based gaze tracking method [108]. To deal with noisy gaze traces and accommodate unknown keystroke counts, we develop a novel decoding algorithm to rank all possible typing sequences and finally output the ones with high rank.

Our contributions are summarized as follows.

- We propose EyeTell, a novel video-based attack that can infer a victim's keystrokes on a touchscreen device from a video capturing his eye movements. In comparison with prior work [50, 51, 52, 53, 54, 55, 56, 57, 58], EyeTell requires neither the attacker to visually observe the victim's typing process nor the victim device to be placed on a static holder. Therefore, EyeTell is more practical, sneaky, and launchable from a large distance, thus posing a more serious threat to user privacy.

- We prototype and evaluate EyeTell through experiments on both iOS and Android devices, which involve the PIN, pattern-lock, and alphabetical soft keyboards. We show that EyeTell can identify the top-5, top-10, and top-50 likely PINs that must contain a target 4-digit PIN with probabilities up to 65%, 74%, and 90%, respectively. Similarly, EyeTell can output the top-5, top-10, and top-50 possible lock patterns that must include a target Android lock pattern with probabilities up to 70.3%, 75.3%, and 85.1%, respectively. In addition, EyeTell can identify the top-5, top-10, top-25, and top-50 likely words that must include a target word with probabilities up to 38.43%, 63.19%, 71.3%, and 72.45%, respectively.

- We point out future directions to improve EyeTell and also possible countermeasures. Although currently EyeTell works only under a short recording distance and a small recording angle, we believe that the adoption of better optics and eye tracking techniques can readily relieve such limitations.

5.2 Background on Video-Based Gaze Tracking

EyeTell is based on the intuition that a victim's gaze trace can reveal his typing sequence on a soft keyboard. Fig. 5.1 depicts the anterior segment of a human eye.

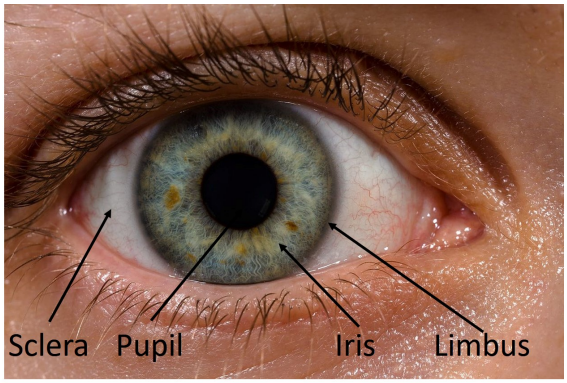


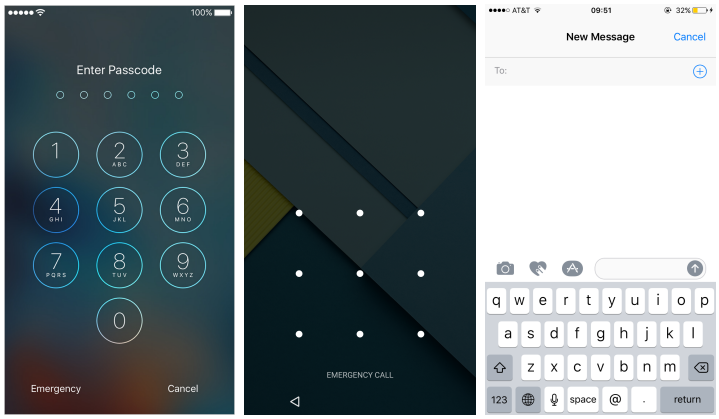
Figure 5.1: Anterior segment of a human eye [2].

According to the definition in [109], the gaze actually refers to the gaze direction. We now briefly introduce the background of video-based gaze tracking, which is used in EyeTell to extract gaze traces.

Gaze tracking refers to the techniques that determine the gaze direction of the eyes. Gaze tracking has numerous applications such as human attention analysis and gaze-based human-computer interfaces. So far video-based gaze tracking is most popular because it achieves high accuracy without requiring the target to wear any special device.

There are mainly two types of video-based gaze tracking methods: feature-based and appearance-based [109, 110]. Feature-based methods use local features such as contours, eye corners, and reflections from the eye image for gaze estimation. In contrast, appearance-based methods directly use the content of the eye image as input to estimate the gaze direction instead of extracting any local feature.

Feature-based methods can be further divided into interpolation-based and model-based methods according to how the features are used. Interpolation-based methods commonly assume that the mapping between the image features and gaze can be modeled as a parametric form such as a polynomial or nonparametric one like a neural network. In contrast, model-based methods directly calculate the gaze from



(a) PIN (b) Pattern lock (c) Alphabetical

Figure 5.2: Three representative soft keyboards.

the image features based on suitable geometric models of the human eye. Here, we adopt the model-based gaze tracking method in [108] due to its advantage that the attacker does not need to obtain any training data about the victim prior to the attack. Other model-based methods can be used in EyeTell as well if they require no training data.

5.3 Related Work

In this section, we discuss the prior work most related to EyeTell in two research directions: keystroke inference attacks and eye-tracking-related security implications.

5.3.1 Keystroke Inference Attacks

Prior keystroke inference attacks can be broadly classified into video-based, sensor-based, and WiFi-based attacks.

Video-based attacks

In this category, the attacker uses a recorded video to infer keystrokes. Early work targets physical keyboards. For instance, Backes *et al.* [50, 51] recovered the content on a computer screen from its reflections on nearby objects such as glasses and tea pots. As another example, Balzarotti *et al.* [52] inferred the keystrokes by characterizing the light diffusion around the keyboard in the video recording. This work [52] requires the attacker to directly video-record the victim's finger typings on the physical keyboard.

More recent research along this line targets soft keyboards on ubiquitous touchscreen mobile devices. In [53], Maggi *et al.* tried to recover keystrokes from key magnifications on the touchscreen. In [54], Raguram *et al.* inferred keystrokes from the touchscreen's reflection on the victim's sunglasses. In [55], Xu *et al.* extended the attack in [54] to recover keystrokes from double reflections of the touchscreen. In [56], Yue *et al.* inferred keystrokes by exploiting the homographic relationship between captured images and a reference image of a soft keyboard. Similar homographic relationship was also used in [57] by matching finger movements. In [58], Sun *et al.* showed that the keystrokes can be actually inferred from the motion of a tablet's backside. All these attacks require the attacker to record a video capturing at least part of the victim's typing process or device backside, so they do not work if no such video is available. For example, the surrounding environment may prevent the attacker from having an unobstructed, stealthy view of the victim's typing process.

In contrast, EyeTell requires no unobstructed view of the victim's device or typing process and only needs the attacker to record the victim's eye movements during the typing process. When a user types, he usually holds his device in one hand or places it on a table or his knee. This means that his eyes are normally at much higher positions

than his device during the typing process. So it is much easier and more sneaky to video-record the user's eye movements from a distance than ti video-record his device motion or typing process. EyeTell is thus applicable to much wider contexts.

Sensor-based attacks

In this category, the attacker uses on-board sensor data to infer a victim's keystrokes. In [25, 21], it was shown that the accelerometer data of a mobile device can be used to infer the victim's password. Subsequently, keystrokes were inferred in [111, 24] by combining both accelerometer and gyroscope data. In [112, 113], the authors exploited microphones and front cameras for keystroke inference. In comparison with video-based attacks (including EyeTell), these attacks require the attacker to acquire sensor data from the victim device through either malware infection or unprotected data transmissions. Such assumptions may not always hold in reality.

There is also work on using device sensors as the side channels to infer keystrokes of nearby physical keyboards. In [114, 115, 116], keystrokes on a physical keyboard were recovered through analyzing the acoustic emanations of the keyboard recorded by a nearby malicious microphone. In [117, 118], keystrokes were inferred by analyzing the time difference of arrival of acoustic signal recordings. In [119], Marquardt *et al.* used the accelerometer on a smartphone to measure the vibration induced by a nearby physical keyboard for keystroke inference. In [120], Liu *et al.* inferred keystrokes by exploiting the accelerometer data of a smartwatch worn by the victim while he typed. Similar to sensor-based attacks [25, 21, 111, 24, 112, 113], these schemes [114, 115, 116, 117, 118, 119, 120] assume that the attacker can obtain sensor data from the victim device or that sensor data can be collected by other devices close to the physical keyboard. By comparison, EyeTell has no such restriction and can be launched from a larger distance.

WiFi-based attacks

In this category, the attacker infers a victim's keystrokes from recorded channel state information (CSI). The idea is that different keystrokes lead to distinct changes in wireless channels and the corresponding CSI. It has been shown that CSI information can be exploited to infer a victim's keystrokes on a physical keyboard [121], or a soft keyboard [122], or a pattern lock keyboard [123]. All these attacks are user-dependent and require the attacker to first obtain the victim's data with known labels to train a classifier. In addition, they cannot tolerate any change in the surrounding environment other than the victim's hand or finger movement. Furthermore, the distance between the WiFi transmitter and receiver, the orientation of the victim device, and the victim's typing gestures were all fixed in the experiments. These shortcomings limit the applicability of WiFi-based keystroke inference attacks in practice.

5.3.2 Eye-Tracking-Related Security Implications

Considering eye tracking as an input method for user-device interaction, researchers have proposed to use it for user authentication and inferring user input.

User authentication

In the early days, researchers tried to use eye movement as a biometric identifier for user authentication. In [124], the authors put forward this idea and evaluated the identification rate among users. In [125, 126], the authors proposed novel features extracted from eye movements and designed specific stimulus to enhance the performance.

More recent research in this line mainly focuses on designing novel challenge-and-response schemes for user authentication in a contactless manner. The key motivation is that eye tracking as an input method is more secure against shoulder-surfing attacks,

besides novel two-factor authentication [40] and anti device-theft [127] schemes. For example, the authentication systems in [128, 129, 130, 131] ask a user to follow moving objects on the screen, draw pre-selected shapes, perform eye gestures to input PIN passwords, etc.

Inferring user input

There are few efforts to work on inferring user inputs on device touchscreen by exploiting eye tracking as a side channel. In [132], the authors pointed out that the victim's eyes would follow his finger movements on the touchscreen of mobile device, which may leak his inputs. To show such feasibility, they mannually analyzed the images taken by the front camera of the victim's device to infer his input digits. Through a small scale of experiments (three participants and nine trials in total), they obtained an accuracy result of around 67% on PIN keyboard.

Compared with the above work, EyeTell exhibits two main differences. First of all, EyeTell works on a more challenging scenario of inferring user keystrokes on mobile device touchscreen, while most user authentication schemes based on eye tracking aim at much larger screens such as TV. Furthermore, these schemes were engineered in a way that their eye tracking module can obtain a user's eye trace easily and effectively. On the contrary, EyeTell can only obtain a much noisier eye trace due to two reasons: the attacker does the video recording from a distance, and the victim's eye movements on a mobile touchscreen is much more subtle. Secondly, EyeTell involves a set of tools to infer user inputs and comprehensive investigations on different types of soft keyboards to better evaluate its security and privacy impacts.

5.4 Adversary Model

We consider a victim using a mobile touchscreen device such as a smartphone or tablet. Assume that the victim holds the device right in front of himself and types on the touchscreen soft keyboard. Such scenarios are very common in practice. For example, the victim may use his mobile device at his workplace or wait in line at a coffee shop. We assume that the victim is alert to conventional shoulder-surfing attacks in the sense that the attacker cannot get too close to the victim when he types on the device.

We consider an attacker who aims to infer the typed sequence on the victim device, which could be PINs, lock patterns, words, or sentences. We assume that the attacker can use a COTS smartphone, digital camera, or camcorder to record the victim's eyes during his typing process, possibly from a long distance. However, the attacker cannot obtain any IMU sensor (accelerometer, gyroscope, microphone, etc.) data by installing malware such as Trojans or malicious web scripts on the victim device. Different from prior work, we assume that the attacker can see neither the touchscreen or backside of the victim device nor the victim's hand movements during his typing process. Under these assumptions, existing video-based [50, 51, 52, 53, 54, 55, 56, 57, 58, 133] and sensor-based [25, 21, 111, 24, 112, 113, 115, 116, 117, 119] keystroke inference attacks no longer work.

5.5 EyeTell Design

In this section, we give an overview of EyeTell and then detail its design. For convenience only, we assume the victim device to be a smartphone throughout the illustration, though EyeTell can work with any mobile touchscreen device.

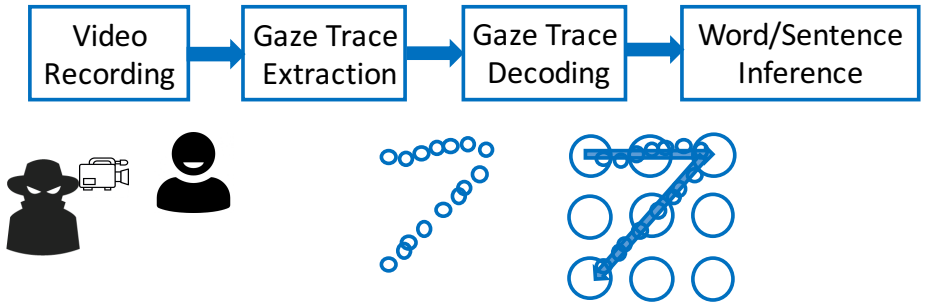


Figure 5.3: Workflow of EyeTell.

5.5.1 Overview

EyeTell is designed to infer the sensitive inputs on the soft keyboard from the video of the victim's eye movements while he types. The high-level design of EyeTell is shown in Fig. 5.3, which consists of the following four steps.

(1) Video Recording. We first record a video capturing the victim's eyes during his inputting process using a COTS camcorder. As mentioned in Section 5.4, we assume that neither the touchscreen nor the victim's hand movement can be directly seen from the video. In addition, we do not assume that the smartphone is fixed on a device holder or that the video can capture its backside.

(2) Gaze Trace Extraction. We adapt user-independent gaze tracking [108] to extract the gaze direction from each frame of the recorded video and then combine the directions to obtain a complete gaze trace. In particular, we detect the two eyes in each video frame and then the limbus for each eye, from which we finally estimate the corresponding gaze direction. Due to the noisy and unstable nature of the extracted gaze trace, we further apply outlier detection and low-pass filtering to obtain a cleaner gaze trace.

(3) Trace Decoding. In this step, we design a novel decoding algorithm to match the gaze trace extracted in Step 2 into a set of candidate typing sequences on the soft

(a) Side view (b) Attacker's view

Figure 5.4: Typical setup for video recording.

keyboard. Fig. 5.2 shows the soft keyboards we investigate, including the pattern-lock keyboard on Android and the PIN and alphabetical keyboards on iOS. For PIN or pattern-lock inference, each candidate typing sequence corresponds to one or several PINs or lock patterns. For word or sentence inference, an additional step is taken to select meaningful results with the assistance of a dictionary. The decoding algorithm must adapt to different inference scenarios where the attacker's prior information may vary a lot. For example, the attacker knows that a PIN must consist of four or six digits, but he knows very little to none about which word the victim is likely to input before doing word inference.

(4) Word/Sentence Inference. Finally, we select the possible words by considering meaningful alphabetical combinations using a dictionary. We also explore the linguistic relationship between adjacent English words to further infer sentences.

We detail each step above in what follows.

5.5.2 Video Recording

In this step, we want to obtain a video of the victim's eyes when he types on the soft keyboard of the smartphone. Fig. 5.4 shows a typical setting of video recording in our experiments. We ask the participants to sit on the chair and input on a smartphone. A Panasonic HCV7000 camcorder is used to record videos. Using a COTS camcorder can show that EyeTell is low-cost, convenient, and stealthy to launch. In our studies, we find that the following factors affect the result of our gaze tracking algorithm.

Image resolution. The resolution of the recorded video affects eye and limbus detection and therefore the extracted gaze. In the experiments, we always stick to the highest resolution of the camcorder, i.e., each video frame is of 1920×1080 pixels.

Video frame rate. Due to the noisy and instable nature of the extracted gaze trace, we need to collect more sudden changes of the user's eye movement and thus desire a higher video frame rate. In the experiments, we choose the frame rate as 60 fps, which is the highest frame rate supported by our camcorder. Our attack can be more effective if a camcorder supporting higher frame rates is available.

Light condition. The light condition in the environment may also affect the inference result, as the imaging sensor of the camcorder generates larger noise in low-illumination environments and thus produces a polluted gaze trace.

Recording angle. We define the recording angle as the angle between the plane formed by the victim and his smartphone and the plane formed by the victim and the attacker's camcorder. Our current EyeTell implementation requires that the camcorder be placed in the same plane as the victim and his smartphone, typically as shown in Fig. 5.4. Therefore, our default recording angle is zero degree. We believe that this assumption is fairly easy to achieve in practice with advanced camcorders and can be relieved if more sophisticated gaze tracking algorithms are available.

After video-recording the victim's eye movement, we manually crop the beginning and ending part of the video such that the remaining part contains only the typing process. For example, the video only contains the process of the victim inputting four digits or drawing a pattern on the smartphone.

5.5.3 Gaze Trace Extraction

There are three steps in gaze trace extraction: eye localization, limbus detection, and gaze trace estimation.

Eye detection

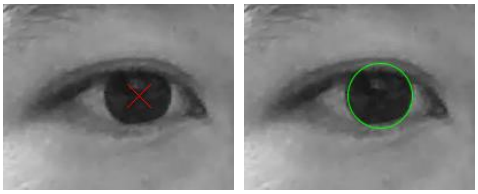
EyeTell detects the victim's eyes in each frame through a two-step approach. We first search for a pair of eyes within the entire frame in a coarse-grained manner. Once a rough region is obtained, we further refine the detected eye region and then calculate the accurate eye positions.

In the first step, we use a Haar-like feature-based cascade classifier [92] to detect possible eye regions and always select the first output as the candidate eye region. We then segment the candidate eye region into two area-of-interests (AOIs), one for each eye. The cascade classifier [92] is very efficient and also user-independent, but it may still incur false positives that the candidate region is not the eye region. For example, a rectangular area enclosing the user's clothes may be misclassified as the eye region.

We use two tricks to reduce such false positives. First, we require that the size of the detected eye region be above a minimum threshold. In our implementation, we set the threshold to be 80×40 pixels, which has been shown valid for our video recording setting. Second, we calculate a similarity score between the detected eye region and a reference region, which is the eye region successfully detected in a different frame

of the same video. In particular, we resize the candidate eye region to the same size as the reference region and then normalize the pixel values of both regions. After normalization, we calculate a pixel-level similarity score for the same pixel in the two regions, which is the ratio between the absolute difference of the two pixel values and their sum. After that, the similarity score of the two eye regions is calculated as the average pixel-level similarity score across the entire eye region. The smaller the similarity score, the more similar the two eye regions. In our implementation, we use an empirical threshold of 0.8 to filter out possible false positives of eye regions and manually check them. The threshold needs to be adjusted in practice: a small threshold may result in many possible false positives and thus increase the demand for manual checking, and vice versa. If a detected candidate eye region is indeed a false positive, we manually assign a correct rectangular region enclosing both eyes as the input to the cascade classifier, which leads to correct eye detection in practice.

In the second step, EyeTell uses a shape-based approach to refine the two AOIs by exploring the predicted dark circular appearance of eye features [134]. Specifically, we define the center of a circular pattern as the point where most image gradient vectors intersect. Then we search for the optimal point by using an objective function that measures how well the gradient vectors and the eye center displacement vectors are aligned. Moreover, considering the fact that the eye center usually appears darker than other areas in the eye, we attach each point with a weight of its inverse intensity in the objective function. Once the optimal points of the two AOIs (i.e., the two eye centers) are located, we refine the positions of two AOIs in the frame and then resize them to a fixed ratio. The resizing operation can minimize the areas of the two AOIs while maintaining important eye features within them. The red cross in Fig. 5.5a denotes the detected eye center.



(a) Eye center (b) Fitted limbus

Figure 5.5: Examples of our detected eye center and limbus.

Limbus detection

In this step, EyeTell determines the elliptical outline of the limbus from each identified AOI by first identifying a set of possible limbus edge points and then fitting an ellipse model from those edge points [108]. In contrast to other popular limbus detection methods [135, 136], this method does not rely on any pre-defined threshold, which allows EyeTell to reliably detect the limbus regardless of eye appearance, users, and lighting conditions. Moreover, it can detect the limbus from out-of-focus images because it does not depend on the existence of very strong edge. We illustrate this process in what follows.

Since limbus edge points are part of the edge, we search for them by analyzing the radial derivatives within each AOI. Specifically, we transform a given AOI into the polar form and then calculate the vertical derivative of each pixel. In our implementation, we select the pixel with the largest radial derivative in each column as the limbus edge point.

Special attention is paid to non-edge points that are incorrectly detected as edge points, which occurs if the radial derivatives of non-edge points are larger than those of true limbus edge points. According to our experimental observations, we use the following process to filter out as many such non-edge points as possible. First, we notice that nearby light sources can leave specularities on the cornea. The pixels

within these specularities can have very large radial derivatives and thus be incorrectly identified as limbus edge points. To deal with this case, we compare each pixel value with a threshold, e.g., 150 (the pixel value is between 0 and 255), to identify a set of possible specularities and then inpaint these small connected regions. The effective threshold depends on the recording environment, which we choose empirically. Second, we observe that the upper eyelid may cover part of the iris and therefore lead to incorrect limbus edge points. To cope with this case, we use three points, two eye corners and the iris-eyelid boundary point right above the given eye center, to fit a parabola to approximate the upper eyelid and then discard the points that fall outside the parabola.

Finally, we fit an ellipse model from the set of edge points using the iterative method in [137]. In each iteration, a minimum number of edge points are randomly selected from available ones to fit an ellipse model through a least-square approach. Then a support function is calculated to evaluate how fit the model is to the entire set. We use the support function in [137] that measures how well the geometric gradients of the fitted ellipse model align with the image gradients. Fig. 5.5b denotes a detected limbus in our experiment.

Gaze trace estimation

In this step, we estimate one gaze point from each frame to obtain a complete gaze trace from the entire video. To do so, we use the detected eye centers and limbus in the 2D domain to recover the corresponding 3D eye centers and optical axes. We then estimate the gaze point as the intersection between the optical axes and the virtual 3D screen. We further refer to the gaze point as point-of-gaze (PoG), which can be simply denoted by a vector $[x, y]^T$. Here x and y correspond to the coordinates of the PoG along x and y axis on the screen, respectively. For the benefit of better

readability, here we omit the detailed mathematical deduction to calculate a PoG. Below we detail this process.

First, we calculate the 3D center and optical axis of each eye from the eye center and limbus obtained from limbus detection. Denote the coordinate of the eye center on the 2D image plane by (e_x, e_y) and the fitted ellipse of limbus by $E(x, y) = Ax^2 + Bxy + Cy^2 + Dx + Ey + F$. The 3D center of an eye, denoted by $\mathbf{c} = [c_x, c_y, c_z]^T$, can be calculated as

$$c_x = c_z \frac{(e_x - \mu_0)}{f_x}, c_y = c_z \frac{(e_y - \upsilon_0)}{f_y}, c_z = \frac{f_x + f_y}{2} \cdot \frac{r_0}{r_{\max}}, \tag{5.1}$$

where f_x and f_y are the focal lengths in pixel along horizontal and vertical axis, respectively, (μ_0, υ_0) is the coordinate of the principal point on the 2D image plane, $r_{\max}$ is the semi-major axis of the fitted ellipse $E(x, y)$ on the 2D image plane, and r_0 is the actual size of human limbus. By definition, the line determined by the focal point and the principal point is perpendicular to the 2D image plane, which allows us to calculate the principal point from the focal point. In practice, f_x, f_y, μ_0, and υ_0 can be obtained by one-time camera calibration. In addition, parameters e_x, e_y, and $r_{\max}$ can be computed from $E(x, y)$, and r_0 is set to 6 mm in our implementation.

The optical axis of an eye, denoted by $\mathbf{k}$, can be written as $\mathbf{k} = \mathbf{c} + m\mathbf{n}$. Here $\mathbf{n}$ is the unit normal vector of the supporting plane of the limbal circle, and m is a constant. In the coordinate system of the eye, $\mathbf{n}$ is equal to $[0, 0, 1]^T$. Next, we obtain its corresponding form in the coordinate system of the camera by the rotation matrix between the two coordinate systems through the following equation [138],

$$\mathbf{n} = \begin{bmatrix} \mathbf{v_1} & \mathbf{v_2} & \mathbf{v_3} \end{bmatrix} \begin{bmatrix} h \\ 0 \\ g \end{bmatrix}, \tag{5.2}$$

where $\mathbf{v_1}$, $\mathbf{v_2}$, and $\mathbf{v_3}$ are three eigenvectors of $\mathbf{Q_e}$ defined as

$$\mathbf{Q_e} = \begin{bmatrix} A & \frac{B}{2} & -\frac{D}{f_x+f_y} \\ \frac{B}{2} & C & -\frac{E}{f_x+f_y} \\ -\frac{D}{f_x+f_y} & -\frac{E}{f_x+f_y} & \frac{4F}{(f_x+f_y)^2} \end{bmatrix}, \tag{5.3}$$

$$g = \sqrt{\frac{\lambda_2 - \lambda_3}{\lambda_1 - \lambda_3}}, h = \sqrt{\frac{\lambda_1 - \lambda_2}{\lambda_1 - \lambda_3}}, \tag{5.4}$$

and λ_1, λ_2, and λ_3 are the eigenvalues corresponding to $\mathbf{v_1}$, $\mathbf{v_2}$, and $\mathbf{v_3}$, respectively.

After obtaining the optical axis of each eye, we calculate the PoG as

$$\text{PoG} = \begin{bmatrix} x \\ y \\ 0 \end{bmatrix} = \begin{bmatrix} c_x \\ c_y \\ c_z \end{bmatrix} + m \begin{bmatrix} n_x \\ n_y \\ n_z \end{bmatrix}. \tag{5.5}$$

It follows that

$$m = -\frac{c_z}{n_z} \quad \text{and} \quad \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} c_x + mn_x \\ c_y + mn_y \end{bmatrix}, \tag{5.6}$$

where $[x, y]^T$ is the estimated PoG of a video frame.

By calculating the PoG for each eye in each frame, we obtain two complete gaze traces from the recorded video, denoted by $\Psi^l = (\text{PoG}_1^l, \ldots, \text{PoG}_{n_f}^l)$ and $\Psi^r = (\text{PoG}_1^r, \ldots, \text{PoG}_{n_f}^r)$ for the left and right eyes, respectively, where n_f is the number of frames in the video.

Since the extracted gaze traces are usually noisy and unstable, we apply outlier detection and filtering to enhance their quality. To detect possible outliers, we check the distance between the two estimated eye centers in each frame. If the distance in the ith frame is larger than an anatomical threshold, e.g., 80 mm, we consider that at least one PoG between PoG_i^l and PoG_i^r is an outlier. In this case, we replace the PoG that yields a larger PoG change between adjacent frames with the one that leads to a smaller change.

In the subsequent filtering step, we first obtain a raw gaze trace $\Psi = (\text{PoG}_1, \ldots, \text{PoG}_{n_f})$ by taking the average of the left and right gaze traces, where

$$\text{PoG}_i = \frac{\text{PoG}_i^l + \text{PoG}_i^r}{2} , \tag{5.7}$$

for all $i \in [1, n_f]$. We then apply a triangular kernel [139] to Ψ, which assigns linear weights to each PoG in the time order. Specifically, for each $j \in [1, n_f]$, we calculate

$$\overline{\text{PoG}}_j = \frac{\sum_{i=j-N_1+1}^{j} i \times \text{PoG}_i}{\sum_{i=1}^{j} i}, \tag{5.8}$$

where N_1 is empirically set to 5 in our implementation. The final gaze trace for keystroke inference is $\overline{\Psi} = (\overline{\text{PoG}}_1, \ldots, \overline{\text{PoG}}_{n_f})$. For convenience, we call each element in $\overline{\Psi}$ a PoG as well.

5.5.4 Trace Decoding

In this step, EyeTell decodes the gaze trace $\overline{\Psi}$ to obtain some candidate input sequences on the touchscreen. Depending on the soft keyboard the victim types on, the candidate input sequence may correspond to a lock pattern, a PIN, a word, or a sentence. Generally speaking, trace decoding is done in four steps. First, we identify the turning points in a gaze trace and then divide the whole trace into a sequence of segments, each corresponding to a sudden change in the PoG. Second, we convert each segment into a small set of candidate vectors. Third, given the sequence of segments and their corresponding candidate vector sets, we enumerate all possible combinations of candidate vectors. For each possible combination of candidate vectors, we traverse the soft keyboard to check whether or not the combination can be mapped into a valid input sequence. Finally, we rank all the valid input sequences according to certain heuristic rules and generate a final set of candidate input sequences for a given gaze trace. In what follows, we use the pattern-lock keyboard as the example to illustrate

trace decoding and then point out the difference when applying EyeTell to PIN and alphabetical keyboards.

Trace segmentation

We first apply a moving average filter to further smooth the gaze trace extracted in the last step, as it does not exhibit any clear pattern for segmentation. The length of the moving window has a direct impact on the segmentation performance. On the one hand, if the window is too short, the filtered gaze trace is not sufficiently smooth. On the other hand, if the window is too long, some sudden changes may be buried, resulting in some undetectable turning points. We empirically set the moving-window length to 10 based on analyzing our experiment data.

We then segment the smoothed trace by identifying the turning points that separate adjacent segments. For simplicity, we abuse the notation by letting $\overline{\Psi} = (\overline{\text{PoG}}_1, \ldots, \overline{\text{PoG}}_{n_f})$ denote the smoothed trace as well. Suppose that $\overline{\Psi}$ consists of two segments as an example. In the ideal case, the points in each segment lie in a straight line, and the intersection of the two lines is the turning point between two segments. Based on this observation, we first estimate the moving direction of each PoG (or element) in $\overline{\Psi}$. Let $\overrightarrow{\text{PoG}}_{i,j} = \overline{\text{PoG}}_j - \overline{\text{PoG}}_i$ be the vector for $\forall i, j \in [1, n_f]$. For each $\overline{\text{PoG}}_i$ and the next N_2 PoGs (i.e., $\{\overline{\text{PoG}}_j\}_{j=i+1}^{i+N_2-1}$), we compute N_2 vectors $\{\overrightarrow{\text{PoG}}_{j,i}\}_{j=i+1}^{i+N_2-1}$, where N_2 is a system parameter empirically set to 5 in our experiment. We further calculate

$$\overrightarrow{\text{PoG}}_i = \frac{\sum_{j=i+1}^{i+N_2-1} \overrightarrow{\text{PoG}}_{i+1,i}}{N_2} \tag{5.9}$$

as the moving direction of $\overline{\text{PoG}}_i$. Let $\theta_i \in [-\pi, \pi)$ denote the angle of $\overrightarrow{\text{PoG}}_i$. We can then obtain a sequence of angles $\theta_1, \ldots, \theta_{n_f-N_2+1}$) for the gaze trace $\overline{\Psi}$. For every N_3 adjacent PoGs such as $\{\text{PoG}_j\}_{j=i}^{i+N_3-1}$, we consider them in the same segment if and

only if

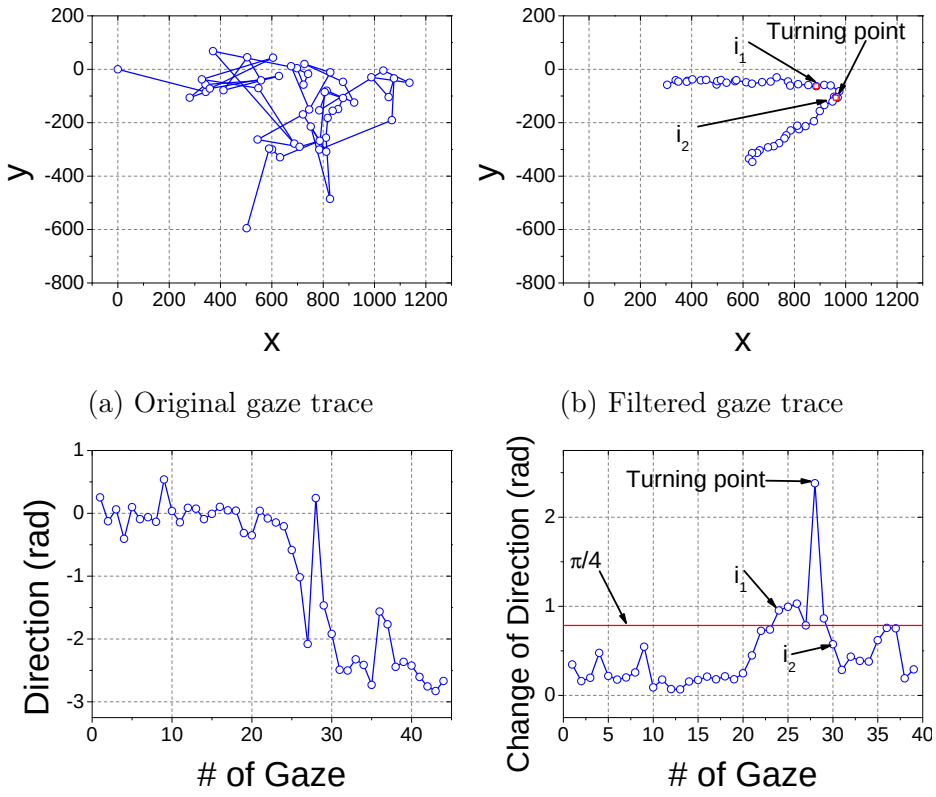
$$\frac{\sum_{j=i}^{i+N_3-1}|\theta_{j+1}-\theta_j|}{N_3} \leq \phi_1 \ ,$$

where N_3 and ϕ_1 are both system parameters that are empirically set to 5 and $\frac{\pi}{4}$ in our experiment, respectively.

We then search for turning points as follows. Starting from $i = 1$, we find the smallest i' such that $\frac{\sum_{j=i'}^{i'+N_3-1}|\theta_{j+1}-\theta_j|}{N_3} > \phi_1$ and then regard $\overline{\text{PoG}}_{i'}$ as the ending point of the first segment. Starting from i', we proceed to find the smallest i'' such that $\frac{\sum_{j=i''}^{i''+N_3-1}|\theta_{j+1}-\theta_j|}{N_3} \leq \phi_1$ and then consider $\overline{\text{PoG}}_{i''}$ as the starting point of the second segment. After determining i' and i'', we search between i' and i'' to find i_1 with the largest $\frac{\sum_{j=i_1}^{i_1+N_3-1}|\theta_{j+1}-\theta_j|}{N_3}$ and consider $\overline{\text{PoG}}_{i_1}$ as the turning point between the first two segments.

Repeating the above process, we can identify all the turning points in the gaze trace. Suppose that n_t turning points are found in total. Combined with the first and last PoGs of the gaze trace, the total n_t+2 points correspond to n_t+1 segments. Denote the $n_t + 2$ points by $\{\text{TP}_i\}_{i=1}^{n_t+2}$, where TP_1 and TP_{n_t+2} correspond to the first and last PoGs of the gaze trace, respectively, and TP_i ($\forall i \in [2, n_t + 1]$) are the turning points. In the remainder of the chapter, we denote the number of segments by n_s. Therefore, $n_s = n_t + 1$. The final output of trace segmentation comprises n_s segments, each of which can be represented by its length and angle. Specifically, assuming $\text{TP}_i = [x_i, y_i]^T$, the i-th segment can be characterized by $[x_{i+1}-x_i, y_{i+1}-y_i]^T$ for all $i \in [1, n_s]$.

We use the example in Fig. 5.6 to shed more light on trace segmentation. Specifically, Fig. 5.6a shows a two-segment gaze trace to decode; Fig. 5.6b shows the gaze trace after applying the moving average filter; Fig. 5.6c shows the angles of the PoGs on the trace; and Fig. 5.6d shows the ending point of the first segment and the starting point of the second segment.



(a) Original gaze trace

(b) Filtered gaze trace

(c) Direction

(d) Change of direction

Figure 5.6: An illustration for trace dividing.

N_2, N_3, and ϕ_1 depend on the factors such as frame rate, signal-to-noise ratio (SNR) of the video, etc. For example, N_2 and N_3 increase with the frame rate and decrease with SNR generally. In our system, we choose these parameters empirically by experimenting them with a small portion of data and observing the results of segmentation. In practice, we believe that they need to be adjusted or trained in different scenarios.

Decoding segment

We observe that only a limited number of gaze segments are permissible on any typical soft keyboard (PIN, pattern lock, and alphabetic), which are referred to as

Table 5.1: Mapping between alphabetical and quasi-PIN keyboards depicted in Fig. 5.8a.

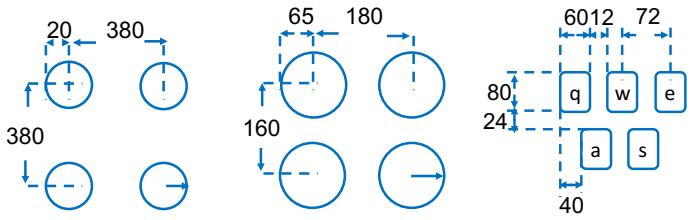
1	q,w,e,r	2	t,y	3	u,i,o,p
4	a,s,d	5	f,g,h	6	j,k,l
7	z,x	8	c,v,b	9	n,m

Table 5.2: All possible segments of pattern-lock keyboard.

Index	Length	Angle	Index	Length	Angle	Index	Length	Angle	Index	Length	Angle
①	1	0	⑦	1	$\frac{\pi}{2}$	⑬	1	π	⑲	1	$-\frac{\pi}{2}$
②	2	0	⑧	2	$\frac{\pi}{2}$	⑭	2	π	⑳	2	$-\frac{\pi}{2}$
③	$\sqrt{5}$	0.464	⑨	$\sqrt{5}$	2.03	⑮	$\sqrt{5}$	-2.68	㉑	$\sqrt{5}$	-1.11
④	$\sqrt{2}$	$\frac{\pi}{4}$	⑩	$\sqrt{2}$	$\frac{3\pi}{4}$	⑯	$\sqrt{2}$	$-\frac{3\pi}{4}$	㉒	$\sqrt{2}$	$-\frac{\pi}{4}$
⑤	$2\sqrt{2}$	$\frac{\pi}{4}$	⑪	$2\sqrt{2}$	$\frac{3\pi}{4}$	⑰	$2\sqrt{2}$	$-\frac{3\pi}{4}$	㉓	$2\sqrt{2}$	$-\frac{\pi}{4}$
⑥	$\sqrt{5}$	1.11	⑫	$\sqrt{5}$	2.68	⑱	$\sqrt{5}$	-2.03	㉔	$\sqrt{5}$	-0.464

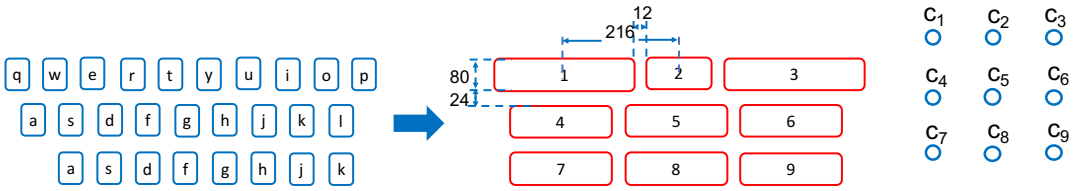
Table 5.3: Soft keyboard dimensions in pixel illustrated in Fig. 5.7 and Fig. 5.8a.

Keyboard	Radius	Width	Height	Horizontal Gap	Vertical Gap
PIN	65	N/A	N/A	50	30
Pattern lock	20	N/A	N/A	340	340
Alphabetical	N/A	60	80	12	24
Quasi-PIN	N/A	216	80	12	24



(a) Pattern-lock (b) PIN (c) Alphabetical

Figure 5.7: Measurement of the three keyboards. The unit is pixel.

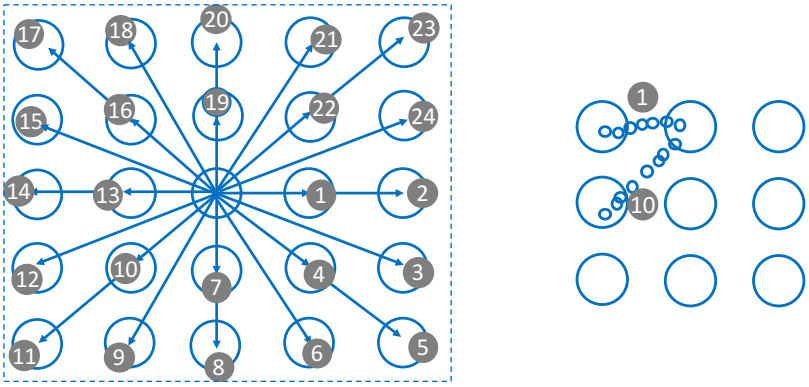


(a) Transform alphabetical keyboard to quasi-PIN keyboard. (b) Denotions in Table 5.4.

Figure 5.8: Quasi-PIN keyboard.

legitimate segments hereafter. In this step, we decode a given gaze segment into a small set of candidate legitimate segments on the pattern-lock keyboard. This is done by calculating the Euclidean distances between the given segment and all legitimate ones and then selecting those with shorter distances as the candidates.

Let us first look into more details of the pattern-lock keyboard and its corresponding legitimate segments. Fig. 5.7 depicts the dimensions of the pattern-lock keyboard layout on a Google Nexus 6 smartphone with Android 5.1.1, including the radius of nine white circles, the horizontal gap between two neighboring circles, and the vertical gap between two neighboring circles. All these dimensions are also listed in Table 5.3.. We further plot all the 24 possible segments on the pattern-lock keyboard



(a) All possible segments. (b) A pattern and its segments.

Figure 5.9: Segments on pattern-lock keyboard.

in Fig. 5.9 and then calculate their lengths and angles. The 24 segments lead to five lengths and 16 angles in total.

We first normalize the segment length to facilitate segment decoding. As we can see from Fig. 5.9, the minimum segment length is 1, so we try to make the minimum normalized segment length be 1 as well via the following approach. First, we sort the segments in the ascending order of their lengths. Let $L_{\max}$ denote the longest segment length. Then we select the shortest segment and calculate the ratio ρ between $L_{\max}$ and its length. According to Table 5.2, the length ratio between any two legitimate segments is no larger than $2\sqrt{2}$. Therefore, we compare ρ to a threshold $\rho_{\max}$. If $\rho \leq \rho_{\max}$, the currently selected segment is used for normalizing all the segments. Otherwise, we select the next shortest segment, calculate a new ρ, and compare it to $\rho_{\max}$. This process ends until $\rho \leq \rho_{\max}$. The currently selected segment is the one used for length normalization, and the normalized segment lengths smaller than 1 are all set to 1. $\rho_{\max}$ should be larger than $2\sqrt{2}$ to accommodate the noisy and instable nature of the gaze trace.

Next, we compute the Euclidean distance between each normalized segment and each legitimate segment in Fig. 5.9. Suppose that we look for η candidate legitimate segments for each normalized segment. Those leading to the top-η shortest Euclidean distances are selected as the candidates. Intuitively speaking, the larger η, the more likely that the correct legitimate segment is included in the candidate set, the less pinpointing capability the attacker has, and vice versa.

The final output in this step corresponds to n_s candidate sets, each corresponding to a gaze trace segment. We denote the candidate set for the i-th trace segment by $\mathcal{N}_i$ ($\forall i \in [1, n_s]$), which contains η legitimate segments in the ascending order of their Euclidean distances with the i-th trace segment.

Candidate lock patterns

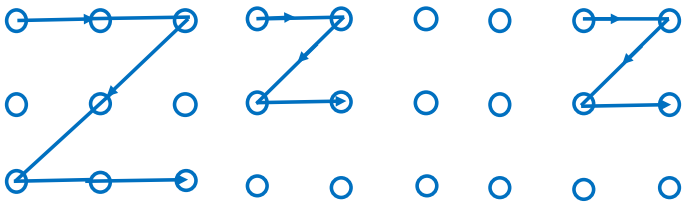
Now we generate the candidate lock patterns for a gaze trace. Let $c_1, \ldots, c_9$ denote the nine white circles of a pattern lock keyboard, as shown in Fig. 5.8b. By setting the center coordinate of c_1 to $(0, 0)$, we derive the center coordinates of other circles and list them in Table 5.4. Since the gaze trace comprises n_s segments with each having η candidate legitimate segments, a candidate lock pattern can be represented by a row vector $p = [p_1, \ldots, p_{n_s+1}]$, where p_i refers to the i-th point that corresponds to one of $c_1, \ldots, c_9$.

Table 5.4: Coordinates of pattern-lock keyboard depicited in Fig. 5.8b.

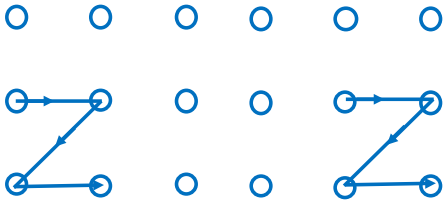
c_1	(0,0)	c_2	(1,0)	c_3	(2,0)
c_4	(1,0)	c_5	(1,1)	c_6	(2,1)
c_7	(2,0)	c_8	(2,1)	c_9	(2,2)

We generate the candidate lock patterns by considering each possible combination of n_s legitimate segments and then checking its feasibility by traversing on the pattern-

lock keyboard. In each round, we select a random segment S_i among the η segments in $\mathcal{N}_i$ ($\forall i \in [1, n_t]$) to form a legitimate segment sequence $\{S_1, \ldots, S_{n_s}\}$. There are totally η^{n_s} rounds, each with a unique legitimate segment sequence. Assuming that the length and angle of S_i are l and α, respectively, we rewrite $S_i = (l\cos(\alpha), l\sin(\alpha))$. Given $\{S_1, \ldots, S_{n_s}\}$ and an arbitrary starting point $p_s \in \{c_i\}_{i=1}^{9}$, we can obtain a candidate lock pattern p, where $p_1 = p_s$ and $p_i = p_{i-1} + S_{i-1}$ ($\forall i \in [2, n_s + 1]$). We say that p is *feasible* if $p_i \in \{c_1, \ldots, c_9\}, \forall i \in [1, n_s + 1]$. There are nine possible choices for p_s, each corresponding to a candidate lock pattern. All the feasible lock patterns are then recorded.



(a) c_1, c_3, c_7, c_9. (b) c_1, c_2, c_4, c_5. (c) c_2, c_3, c_5, c_6.



(d) c_4, c_5, c_7, c_8. (e) c_5, c_6, c_8, c_9.

Figure 5.10: Ambuguities due to normalization.

An undesirable consequence of length normalization is that larger lock pattens may be mis-recognized as their shrunken versions. The example in Fig. 5.10 illustrates this aspect. The correct pattern in the example is $[c_1, c_3, c_7, c_9]$, and the gaze trace segments after length normalization are $\{(1, 0), (-1, 1), (1, 0)\}$. So the candidate lock patterns are $[c_1, c_2, c_4, c_5]$, $[c_2, c_3, c_5, c_6]$, $[c_4, c_5, c_7, c_8]$, and $[c_5, c_6, c_8, c_9]$, illus-

trated in Fig. 5.10. Our remedy for the issue is that if a legitimate segment sequence $\{S_1, \ldots, S_{n_s}\}$ can generate a feasible lock pattern, we double the length of each segment there and then check if the new sequence $\{\tilde{S}_1, \ldots, \tilde{S}_{n_s}\}$ can generate a feasible lock pattern or not, where $\tilde{S}_i = (2l\cos(\alpha), 2l\sin(\alpha))$. All such feasible lock patterns are recorded as well.

Ranking candidate lock patterns

The final step is to rank candidate lock patterns with three heuristics as follows.

First, we introduce a row vector $r = (r_1, \ldots, r_{n_s})$, where $r_i \in [1, \eta]$ means that the r_i-th segment is chosen from $\mathcal{N}_i$ ($\forall i \in [1, n_s]$). Then we generate the η^{n_s} legitimate segment sequences based on r in the following order: $[1, 1, \ldots, 1, 1], [1, 1, \ldots, 1, 2], \ldots,$ $[1, 1, \ldots, 1, \eta], [1, 1, \ldots, 2, 1], [1, 1, \ldots, 2, 2], \ldots [1, 1, \ldots, 2, \eta], [1, 1, \ldots, 3, 1], \ldots, [\eta, \ldots, \eta]$. Recall that the earlier segments in each $\mathcal{N}_i$ have smaller Euclidean distances to the corresponding gaze trace segment than those of the later segments. Earlier legitimate segment sequences can thus produce higher ranked candidate lock patterns than later ones.

Second, the candidate lock patterns generated from the same legitimate segment sequence are ranked according to their starting points in the order of $c_1 > c_4 > c_7 > c_2 > c_5 > c_8 > c_3 > c_6 > c_9$. Such a heuristics is also adopted in [133].

Finally, the candidate lock patterns generated from an enlarged legitimate segment sequence have higher ranks than those from the original sequence. The intuition is that normal users tend to draw larger patterns.

Table 5.5: Hidden keys on PIN keyboard.

(1,3)	2	(4,6)	5	(7,9)	8	(1,7)	4	(2,8)	5
(3,9)	6	(1,9)	5	(3,7)	5	(5,0)	8	(2,0)	5,8

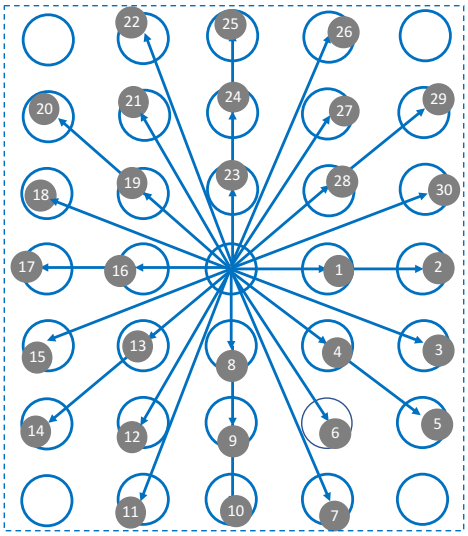


Figure 5.11: All possible segments of a PIN keyboard.

PIN keyboard

Now we discuss how EyeTell applies to the PIN soft keyboard. Fig. 5.7 and Table 5.3 show the dimensions of the PIN keyboard layout on a Google Nexus 6 with Android 5.1.1, including the radius of each key, the horizontal gap, and the vertical gap. We plot the 30 legitimate segments in Fig. 5.11. Note that users slide on the pattern-lock keyboard to draw a pattern but touch the keys on the PIN keyboard to input a PIN. A user may input the same key multiple times on the PIN keyboard, in which case there is little displacement in the corresponding gaze trace. Furthermore, a user may input three keys along the same direction sequentially, in which case the attacker does not know how many keys are touched. For example, the gaze traces for two different PINs (e.g., $[1, 4, 7, 9]$ and $[1, 7, 8, 9]$) can be very similar.

We then modify the process in Section 5.5.4 to generate candidate 4-digit PINs.

- If there are three trace segments, EyeTell directly generates candidate 4-digit PINs as in Section 5.5.4.

- If there are two trace segments, EyeTell first follows the process in Section 5.5.4 to generate candidate 3-digit PINs. We abuse the notation by letting a candidate PIN be denoted by a row vector p, in which each element is a key on the PIN keyboard. We have $p = [p_1, p_2, p_3]$ initially and then generate candidate 4-digit PINs as follows. First, we generate and record $[p_1, p_1, p_2, p_3]$, $[p_1, p_2, p_2, p_3]$, and $[p_1, p_2, p_3, p_3]$, as the user may type any key twice. Second, we consider the possible hidden keys between any two original keys. For example, if a possible hidden key p_h lies between p_1 and p_2, we consider and record $[p_1, p_h, p_2, p_3]$ as a candidate PIN as well. Table 5.5 shows the possible hidden keys on the PIN keyboard corresponding to each pair of original keys.

- If there is only one trace segment, EyeTell first follows the process in Section 5.5.4 to generate a 2-digit PIN denoted by $p = [p_1, p_2]$. Then we generate and record the candidate PINs as $[p_1, p_1, p_1, p_2]$, and $[p_1, p_2, p_2, p_2]$.

The above process can be easily extended to 6-digit PINs and omitted here for lack of space.

Alphabetical keyboard

Here we discuss how to adapt our algorithm to attack the alphabetical keyboard whose layout dimensions are given in Fig. 5.7 and Table 5.3. In contrast to the PIN keyboard, the alphabetical keyboard has more keys (26 instead of 10) and a smaller area (about 48% smaller), which poses a great challenge to keystroke inference. We tackle this challenge by first transforming the alphabetical keyboard into a quasi-PIN keyboard, as shown in Table 5.1 and depicted in Fig. 5.8a. Then we generate candidate PINs on the quasi-PIN keyboard as in Section 5.5.4. Next, we produce a

list of candidate words from candidate PINs and then use a dictionary to filter out non-existing words.

Keystroke inference on the quasi-PIN keyboard is even harder than that on the PIN keyboard. Specifically, the user may type the same key multiple times or hit some hidden keys on a segment, which is difficult for the attacker to identify. In addition, the attacker knows that the PIN to infer corresponds to 4 or 6 keys on the PIN keyboard, while he has no idea how many keys are contained in a PIN on the quasi-PIN keyboard because the corresponding word to infer may include an arbitrary number of letters. The situation becomes even worse because the same key or a hidden key may be typed multiple times. For example, the combinations of "ty", "er", "gh", and "ui" are quite common in English words.

We amend the process in Section 5.5.4 to increase the accuracy of inferring English words on the quasi-PIN keyboard and thus the alphabetical keyboard.

- We add a $(0, 0)$ segment into the set of legitimate segments. If a $(0, 0)$ segment is selected, the gaze trace stays on the same key, corresponding to the case that the victim inputs the same key repeatedly.

- Since it is unrealistic to consider all the possible lengths of the typed word, we only consider candidate words of $n_s + 1$ or $n_s + 2$ letters long for a given gaze trace of n_s segments.

As in [119, 120, 58], we refine the candidate words with the popular "corn-cob" dictionary [140] which is an on-line word list of 58,110 common English words.

Given a candidate PIN on the quasi-PIN keyboard, we generate a list of candidate words for the extracted gaze trace in the following two steps. First, we enumerate all the possible combinations of letters of the given PIN. Second, we search in the dictionary and add discoverable combinations into the list of candidate words. The

complexity of such a process can be very high. For example, in our experiments, the number of possible PINs for a 13-letter word is in the order of 10^4, the number of possible combinations is in the order of 10^6 ($3^{13} = 1594323$), and the number of strings in the dictionary is 58,110. All these add up to a complexity of 10^{15}. To reduce the search complexity, we build a prefix tree of the "corn-cob" dictionary using trie structure [141] such that the search complexity within the dictionary is $\mathcal{O}(L)$, where L is the length of the given string.

5.6 Performance Evaluation

5.6.1 Experiment Setup

User enrollment

We recruited 22 participants for the experiments, including 5 females and 17 males. Our experiment protocol was approved by the Institutional Review Board (IRB) at our institution and strictly followed in the experiments. Since the participants were only asked to input on smartphones, the experiments did not affect either them or people nearby at all. We only obtained the participants' oral consent because the IRB approved a waiver of the requirement to obtain written consent. All the participants were either graduate students in our lab or others we know in the same university. We did not reward them with any monetary compensation and only treated them to free snacks. Finally, all the recorded videos are stored in password-protected lab computers. As shown in Table 5.6, the number of participants in our evaluation is larger than those in our closely related work.

Table 5.6: Number of participants in related system evaluations.

System	[116]	[119]	[120]	[58]	[133]	EyeTell
Number of participants	N/A	N/A	5	4	10	22

Data collection

We used a Panasonic HCV700 camcorder for video recording in our experiment. This camcorder has a 21× zoom len and can record 1080p60 HD videos. Two smartphone models were used in the experiments: Apple iPhone 6s with a 10.5cm × 5.8cm screen size and Google Nexus 6 with a 12.3cm × 7.5cm screen size.

A typical data collection process is as follows. A participant was asked to sit on a chair (illustrated in Fig. 5.4), hold a smartphone in front of herself/himself, and input on the touchscrren. The input can be a PIN on the PIN keyboard, a pattern on the pattern-lock keyboard, or an English word on the alphabetical keyboard. The participant was asked to input in her/his normal typing/drawing speed. We observed that the participant almost always kept her/his head relatively steady during each inputting process which was very short and less than 5 s in our experiments. Such relatively steady head positions are explored by almost all existing gaze tracking methods, including the one used in EyeTell. The following default settings were used, unless noted otherwise. The distance between the participant and camcorder was around 2 m. The participant, smartphone, and camcorder lay in the same plane. The resolution and frame rate of the camcorder were set as 1920 × 1080 and 60 fps, respectively. We also adjusted the zoom of the camcorder such that the captured face of the participant was focused and larger than 500 × 500 pixels.

In general, we conducted two sets of experiments: one without task randomization and the other with task randomization. The former involved 12 participants, each of

whom performed experiments sequentially from one session to the next. For example, a participant first performed all experiments on inferring a single lock-pattern segment, then complete lock patterns, and so on. In contrast, the latter involved 10 participants, each of who was given randomly permuted tasks. As an example, a participant performed one trial on inferring a single lock-pattern segment, then two trials on complete lock pattens, then three trials on 4-digit PINs, and so on.

We use the experiment on inferring a single segment on the pattern-lock keyboard to examplify how we reduced the impact of fatigue. For this experiment, a participant was asked to draw each segment in Table 5.2 on the pattern-lock keyboard. For a given segment, she/he was asked to draw it five times. To counteract the impact of possible fatigue, the participant was asked to take pauses between two consecutive inputs. Before the experiment, we informed all the participants that they could stop the ongoing experiment freely whenever they felt a need to rest. Finally, we purposely asked each participant to stop and rest for one or two minutes about every ten minutes.

For the set of experiments without task randomization, we designed multiple sessions to fully evaluate EyeTell. In the following sessions (from Section 5.6.3 to Section 5.6.7), we will describe the details of these experiments (e.g., the number of participants, the experiment requirements, etc.) and the corresponding results. The same participant took part in multiple sessions, resulting in a total time between two and three hours. To further reduce the impact of possible fatigue, we collected the data of the same participant on different days. As a result, the total time of doing experiments for each participant was less than one hour on the same day.

For the set of experiments with task randomization, we also designed different experiments for the same participant. Particularly, task randomization was done in two steps. First, we prepared all the experiments (tasks) for the same participant,

assembled them together, and assigned each of them an order number. In our evaluation, a participant was assigned 24 single segments, 10 lock patterns, 10 4-digit PINs, and 10 6-digit PINs. Therefore, the order numbers are from 1 to 54 (i.e., $24 + 10 + 10 + 10 = 54$), which we use a vector $[1, 2, \ldots, 54]$ to denote. Second, we permuted the order vector randomly and obtained a new randomized one for each individual participant. Finally, each participant performed experiments according to her/his given vector.

As can be imagined, our experiments required a participant to look at the touchscreen of a mobile device and input on it repeatedly, which can result in fatigue. There are mainly two factors leading to fatigue in our experiments: experimental time and task similarity. Intuitively, if the experimental time is longer with very similar tasks, participants may easily suffer from fatigue. As mentioned above, we adopted two methods to reduce the impact of passible fatigue as much as possible. On the one hand, we asked the participants to take sufficient pauses during the experiments and stop the experiments freely, and controlled the duration of data collection on the same day. On the other hand, we conducted two sets of experiments, with and without task randomization.

For experiments with randomization, each participant was assigned 54 ordered tasks, of which the order was indicated by her/his given vector. A task can be inputting a single segment, a lock pattern, a 4-digit PIN, or a 6-digit PIN. Each participant was asked to repeat the same task for five times. To reduce the impact of fatigue as much as possible, besides following the above instruction, the participants were told to stop their experiments at any time they wished. Also, we collected the data of the same participant on different days. For each participant, the experimental time on the same day was less than half an hour. The total experimental time for a participant ranged from one and a half to three hours.

5.6.2 Performance Metrics

We use top-k inference accuracy as the main performance metric, as in [116, 119, 120, 58, 133]. Specifically, EyeTell generates a set of ranked candidate inputs (PINs, lock patterns, or letters) for each trial. We claim that a trial succeeds if the true input appears in the top-k candidate inputs. Top-k inference accuracy is defined as the percentage of successful trials. We compare the inference accuracy of EyeTell with that in [116, 119, 120, 58, 133]. Specifically, we compare EyeTell with [133] on inferring lock patterns and with [116, 119, 120, 58] on inferring English words.

5.6.3 Experiments on Pattern-Lock Keyboard

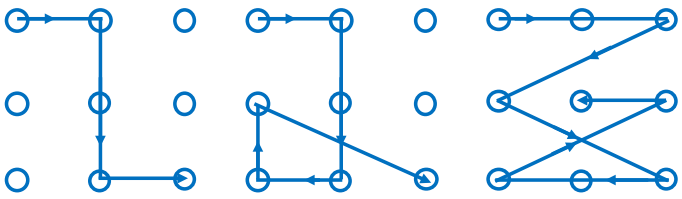
We first evaluate how accurately EyeTell can infer a single segment on the pattern-lock keyboard. Considering that inferring a single segment is the simplest task for EyeTell and the basis for more complicated ones, we want to see how well it performs. For this experiment, we asked each participant to draw each segment in Table 5.2 on a Nexus 6 for five times. Recall that Table 5.2 consists of all the possible single segments on a pattern-lock keyboard. For the segments with multiple possible starting points (e.g., segment ① can start from any point in $\{c_1, c_2, c_4, c_5, c_7, c_8\}$), the participants had the freedom to pick any starting point. Since there is only one segment in the resulting gaze trace, EyeTell can only calculate its angle but not its length. The output length is always 1 due to normalization. Therefore, we group the segments with the same angle together and obtain Table 5.7 from Table 5.2. Therefore, both segment ① and ② in Table 5.2 correspond to segment ① in Table 5.7. Here we ignore the impact of the segment length, which is reported in later evaluations. As we can see in Table 5.8, EyeTell can infer the angle of a single segment on the pattern-lock

keyboard with top-1, top-2, and top-3 inference accuracy up to 87.76%, 98.65%, and 99.74%, respectively.

Table 5.7: Angles of a single segment on the pattern-lock keyboard. Derived from Table 5.2.

Index	Angle	Index	Angle	Index	Angle	Index	Angle
①	0	⑤	$\frac{\pi}{2}$	⑨	π	⑬	$-\frac{\pi}{2}$
②	0.464	⑥	2.03	⑩	-2.68	⑭	-1.11
③	$\frac{\pi}{4}$	⑦	$\frac{3\pi}{4}$	⑪	$-\frac{3\pi}{4}$	⑮	$-\frac{\pi}{4}$
④	1.11	⑧	2.68	⑫	-2.03	⑯	-0.464

For these experiments with task randomization, each participant input each segment in Table 5.2 on a Nexus 6 for five times under task randomization. As we can see in Table 5.9, EyeTell can infer the angle of a single finger movement on the pattern-lock keyboard under task randomization with top-1, top-2, and top-3 inference accuracy up to 87.19%, 97.10%, and 99.62%, respectively.



(a) Simple (b) Medium (c) Complex

Figure 5.12: Examples of simple, medium, and complex lock patterns.

Then we evaluate the performance of EyeTell inferring lock patterns. We used the same set of lock patterns as those in [133], which includes 120 lock patterns in total [142]. In [133], the authors assigned a lock pattern to one of three categories,

Table 5.8: Inference accuracy on a single segment of pattern-lock keyboard.

Index of segment	top-1	top-2	top-3	top-4	top-5
①	87.78%	100%	100%	100%	100%
②	82.5%	90.83%	100%	100%	100%
③	96.67%	100%	100%	100%	100%
④	95%	100%	100%	100%	100%
⑤	80%	100%	100%	100%	100%
⑥	92.22%	100%	100%	100%	100%
⑦	85%	96.67%	100%	100%	100%
⑧	93.33%	100%	100%	100%	100%
⑨	90%	100%	100%	100%	100%
⑩	93.33%	100%	100%	100%	100%
⑪	93.33%	100%	100%	100%	100%
⑫	60%	100%	100%	100%	100%
⑬	80%	92.5%	95.83%	100%	100%
⑭	88.33%	98.33%	100%	100%	100%
⑮	100%	100%	100%	100%	100%
⑯	87.67%	100%	100%	100%	100%
Average	87.76%	98.65%	99.74%	100%	100%

Table 5.9: Inference accuracy on a single segment of pattern-lock keyboard.

Index of segment	top-1	top-2	top-3	top-4	top-5
①	82.5%	100%	100%	100%	100%
②	82.5%	92.5%	100%	100%	100%
③	92.5%	96.7%	100%	100%	100%
④	96.67%	100%	100%	100%	100%
⑤	75%	90.8%	100%	100%	100%
⑥	91.3%	100%	100%	100%	100%
⑦	83%	94.3%	100%	100%	100%
⑧	92.8%	98.2%	100%	100%	100%
⑨	90.3%	100%	100%	100%	100%
⑩	95.5%	100%	100%	100%	100%
⑪	93%	98%	100%	100%	100%
⑫	72%	100%	100%	100%	100%
⑬	84%	91%	94%	100%	100%
⑭	82%	100%	100%	100%	100%
⑮	97%	100%	100%	100%	100%
⑯	85%	92%	100%	100%	100%
Average	87.19%	97.10%	99.62%	100%	100%

i.e., simple, medium, and complex, according to its complexity score. Specifically, the complexity score CS_P of an arbitrary lock pattern P is estimated as

$$\mathrm{CS}_P = n_P \times \log_2(L_P + I_P + O_P), \tag{5.10}$$

where n_P denotes the number of connecting dots, L_P is the length of P, I_P denotes the number of intersections, and O_P is the number of overlapping linear segments. Based on the complexity score, P can then be categorized according to the following rule. If $\mathrm{CS}_P < 19$, P is simple; if $19 \leq \mathrm{CS}_P < 33$, P is medium; and if $\mathrm{CS}_P \geq 33$, P is complex. Fig. 5.12 gives an example for each pattern category. We use the simple pattern in Fig. 5.12(a) to explain the calculation of CS_P, for which we have $n_P = 5, L_P = 4, I_P = 0, O_P = 0$, and $\mathrm{CS}_P = 10$. Each participant was assigned with four simple lock patterns, three medium ones, and three complex ones. The assignment of lock patterns was generated randomly. Besides, each lock pattern was drawn five times on a Nexus 6.

As shown in Table 5.10, the average top-1, top-5, top-10, and top-50 accuracy of EyeTell inferring pattern locks are 57.5%, 70.3%, 75.3%, and 85.1%, respectively. In [133], the authors reported average top-5 accuracy more than 95%, which is much higher than what EyeTell can achieve. But such high accuracy in [133] was achieved based on the strong assumption that the attacker can directly capture how the victim drew her/his lock pattern on the screen. In contrast, EyeTell assumes that the attacker can only capture the victim's eyes (possibly from a large distance), which is much more realistic. We can also see that the inference accuracy increases with the complexity score of a lock pattern, which is consistent with the observation in [133].The reason is that higher pattern complexity helps reduce the number of candidate patterns.

Table 5.10: Inference accuracy on pattern-lock keyboard.

Pattern category	top-1	top-5	top-10	top-20	top-50
Simple	47.75%	69.5%	74.5%	79.5%	88.75%
Medium	59.3%	70%	75%	77%	83%
Complex	65%	71%	76%	78%	83%
Average	57.5%	70.3%	75.3%	78.3%	85.1%

For these experiments with task randomization, each participant input four simple patterns, three medium patterns, and three complex patterns from [142] on a Nexus 6 under task randomization. The patterns were randomly selected when preparing all the tasks for each participant. As shown in Table 5.11, the average top-1, top-5, top-10, and top-50 accuracy of EyeTell inferring pattern locks under task randomization are 55.8%, 70.1%, 75.1%, and 84.1%, respectively.

Table 5.11: Inference accuracy on pattern-lock keyboard.

Pattern category	top-1	top-5	top-10	top-20	top-50
Simple	45.4%	70.4%	75.4%	77.2%	85.6%
Medium	58.6%	69.6%	74.0%	78.0%	83.2%
Complex	63.4%	70.2%	75.8%	77.6%	83.4%
Average	55.8%	70.1%	75.1%	77.6%	84.1%

5.6.4 Experiment on PIN Keyboard

We asked each participant to input 10 4-digit PINs and 10 6-digit PINs on the PIN keyboard on an iPhone 6s. Each PIN was input five times. All the PINs were randomly generated and then assigned to the participants. We showed the results in

Table 5.12. As we can see, EyeTell can infer 4-digit PINs with average top-1, top-5, top-10, and top-50 accuracy up to 39%, 65%, 74%, and 90%, respectively. In addition, the average top-1, top-5, top-10, and top-50 accuracy on 6-digit PINs are 39%, 70%, 80%, and 90%, respectively. As for pattern locks, the inference accuracy for 6-digit PINs is slighter higher than that for 4-digit PINs, as 6-digit PINs are longer, more complex, and thus easier to infer.

Table 5.12: Inference accuracy on PIN keyboard.

# of digits	top-1	top-5	top-10	top-20	top-50
4-digit	39%	65%	74%	81%	90%
6-digit	39%	70%	80%	85%	90%

For these experiments with task randomization, a participant input 10 4-digit PINs and 10 6-digit PINs on an iPhone 6s under task randomization. The PINs were randomly generated when preparing all the tasks for each participant. As shown in Table 5.13, EyeTell can infer 4-digit PINs with average top-1, top-5, top-10, and top-50 accuracy up to 37.5%, 67.2%, 78.0%, and 92.0%, respectively. In addition, the average top-1, top-5, top-10, and top-50 accuracy on 6-digit PINs are 38.8%, 68.9%, 81.3%, and 91.0%, respectively.

Table 5.13: Inference accuracy on PIN keyboard.

# of digits	top-1	top-5	top-10	top-20	top-50
4-digit	37.5%	67.2%	78.0%	81.2%	92.0%
6-digit	38.8%	68.9%	81.3%	84.6%	91.0%

Table 5.14: Words for inference.

Length	Words
7	between, spanish, nuclear
8	identity, emirates, platinum, homeland, security
9	institute, extremely, sacrament, dangerous
10	difference, wristwatch, processing, unphysical
11	inquisition, pomegranate, feasibility, polytechnic, obfuscating
13	paediatrician, interceptions, abbreviations, impersonating, soulsearching, hydromagnetic

5.6.5 Experiment on Word Inference

We used the 27 English words in Table 5.14 from the corn-cob dictionary to evaluate the performance of EyeTell for word inference. The same words were also used in [116, 119, 120, 58]. The length of the 27 words ranges from 7 to 13 letters. We asked each participant to input each word five times on the alphabetical keyboard of an iPhone 6s.

Table 5.15 compares the word-inference performance of EyeTell with some existing schemes. As we can see, the average top-5, top-10, and top-50 accuracy on inferring English words are 38.43%, 63.19%, and 72.45%, respectively. EyeTell has comparable performance to the attacks in [116, 119, 120, 58] but with weaker assumptions. For example, they assume that the attacker can obtain the exact length of the typed word, while EyeTell does not rely on this assumption. In addition, as detailed in Section 5.3, they require that the attacker obtain on-board sensor data of the victim device [116, 119, 120] or that the victim device be placed on a static holder.

Table 5.15: Word-inference accuracy.

System	top-5	top-10	top-25	top-50	top-100
EyeTell	38.43%	63.19%	71.3%	72.45%	73.38%
[116]	N/A	43%	61%	73%	87%
[119]	N/A	43%	50%	57%	60%
[120]	54.80%	63%	75%	82.40%	86%
[58]	48%	63%	78%	93%	N/A

5.6.6 Experiment on Sentence Inference

EyeTell infers a complete sentence in two steps. In the first step, we generate a candidate set for each typed word. In the second step, we use the linguistic relationships between English words to manually select the best candidate for each typed word. Essentially, inferring a complete sentence is based on inferring each individual word (in Section 5.6.5). Therefore, for this experiment, we only involved four participants to demonstrate the feasibility of our approach. Each participant was asked to input two sentences twice on the alphabetical keyboard of an iPhone 6s. The same sentences were also used for evaluation in [58]. We can see that the results for different participants are comparable.

Table 5.16, Table 5.17, Table 5.18, and Table 5.19 show the results. If a typed word does not appear in the candidate set generated by EyeTell, we use a $*$ to denote it. The words in italic form are those EyeTell infers successfully. We also show the number of candidates for each word (including itself). We can see that EyeTell can recover a large portion of the two sentences with the aid of post-inference human interpretation. We believe that we can further improve the performance on sentence

inference by predicting unknown words using advanced linguistic models such as [143].

Table 5.16: Sentence-inference result for the first participant.

Input	our	friends	at	the	university	of	texas	are	planning
Output	our	∗	*at*	*the*	*university*	*of*	*texas*	*are*	*planning*
# **of candi.**	33	N/A	6	3	1	16	6	78	2
Input	a	conference	on	energy	economics	and	finance	in	february
Output	*a*	∗	on	*energy*	∗	and	finance	in	∗
# of candi.	N/A	N/A	5	3	N/A	54	N/A	8	N/A
Input	of	next	year	we	discuss	the	major	factors	underlying
Output	of	next	year	*we*	*discuss*	*the*	major	∗	*underlying*
# **of candi.**	16	30	15	7	8	5	44	N/A	1
Input	the	exceptionally	high	volatility	of	electricity	prices		
Output	*the*	∗	high	∗	*of*	*electricity*	*prices*		
# **of candi.**	5	N/A	85	N/A	16	2	26		

5.6.7 Influence Factors

In this section, we evaluate the impact of multiple factors on EyeTell for inferring 4-digit PINs on the PIN keyboard of an iPhone 6s, including the number of candidates (η) for segment decoding, the number of eyes used for extracting a gaze trace, the frame rate of the camcorder, the lighting condition for video recording, the distance

Table 5.17: Sentence-inference result for the second participant.

Input	our	friends	at	the	university	of	texas	are	planning
Output	our	$*$	*at*	*the*	*university*	*of*	*texas*	*are*	*planning*
# **of candi.**	27	N/A	8	3	1	14	6	90	2
Input	a	conference	on	energy	economics	and	finance	in	february
Output	*a*	$*$	on	*energy*	$*$	and	*finance*	in	*february*
# of candi.	N/A	N/A	5	3	N/A	30	2	8	5
Input	of	next	year	we	discuss	the	major	factors	underlying
Output	of	next	year	*we*	*discuss*	*the*	*major*	$*$	*underlying*
# **of candi.**	12	18	18	3	5	5	8	N/A	1
Input	the	exceptionally	high	volatility	of	electricity	prices		
Output	*the*	$*$	high	$*$	*of*	*electricity*	*prices*		
# **of candi.**	8	N/A	20	N/A	23	1	14		

between the victim and camcorder, and the recording angle. The following default setting was adopted, unless noted otherwise: $\eta = 5$, both eyes used for extracting a gaze trace, a frame rate of 60 fps, indoor normal lighting, 2 m between the victim and camcorder, and a zero-degree recording angle.

Among the 12 participants, only two of them do not wear glasses while the others do. Wearing glasses has little effect on the performance of our system. The reason is that we employ an image inpainting step to eliminate possible specularities within

Table 5.18: Sentence-inference result for the third participant.

Input	our	friends	at	the	university	of	texas	are	planning
Output	our	∗	*at*	*the*	*university*	*of*	*texas*	*are*	*planning*
# **of candi.**	20	N/A	16	3	1	6	6	53	2
Input	a	conference	on	energy	economics	and	finance	in	february
Output	*a*	*conference*	on	*energy*	∗	and	finance	in	february
# of candi.	N/A	1	15	6	N/A	54	N/A	8	10
Input	of	next	year	we	discuss	the	major	factors	underlying
Output	of	next	year	*we*	*discuss*	*the*	major	∗	*underlying*
# **of candi.**	7	25	21	3	5	3	60	N/A	1
Input	the	exceptionally	high	volatility	of	electricity	prices		
Output	*the*	∗	high	∗	*of*	*electricity*	*prices*		
# **of candi.**	5	N/A	100	N/A	18	1	10		

the eye region for limbus detection, as mentioned in Section 5.5.3. As a result, we do not distinguish participants with glasses from those without glasses.

Impact of η

Fig. 5.13a shows the top-5, top-20, and top-100 inference accuracy of EyeTell for $\eta = 3, 4,$ or 5. As we can see, the inference accuracy increases with η, and the top-100 accuracy exhibits the largest increase. Such results are as expected because larger η leads to more enumerations in Section 5.5.4 so that the probability of the typed

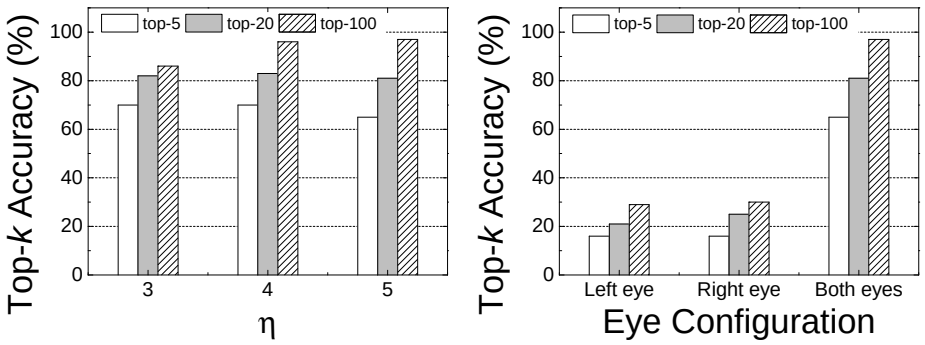
Table 5.19: Sentence-inference result for the fourth participant.

Input	our	friends	at	the	university	of	texas	are	planning
Output	our	∗	*at*	*the*	*university*	*of*	*texas*	*are*	*planning*
# **of candi.**	40	N/A	8	2	2	11	6	63	2
Input	a	conference	on	energy	economics	and	finance	in	february
Output	*a*	∗	on	*energy*	∗	and	finance	in	*february*
# of candi.	N/A	N/A	7	3	N/A	42	N/A	8	2
Input	of	next	year	we	discuss	the	major	factors	underlying
Output	of	next	year	*we*	*discuss*	*the*	major	∗	*underlying*
# **of candi.**	14	18	12	5	12	8	32	N/A	1
Input	the	exceptionally	high	volatility	of	electricity	prices		
Output	*the*	∗	high	∗	*of*	*electricity*	*prices*		
# **of candi.**	5	N/A	91	N/A	12	1	16		

PIN falling into its candidate set increases. In our experiment, we found that when $\eta = 5$, most PINs and lock patterns were included in their respective candidate sets. Though a larger η always leads to higher accuracy, we set $\eta = 5$ by default to reduce computation time.

Impact of eyes

Here we compare the inference accuracy when the gaze trace from only one eye (left or right) or from both eyes are used for PIN inference. The result is shown in Fig. 5.13b.



(a) η (b) Eye configuration

Figure 5.13: Impact of η (left) and eye configuration (right).

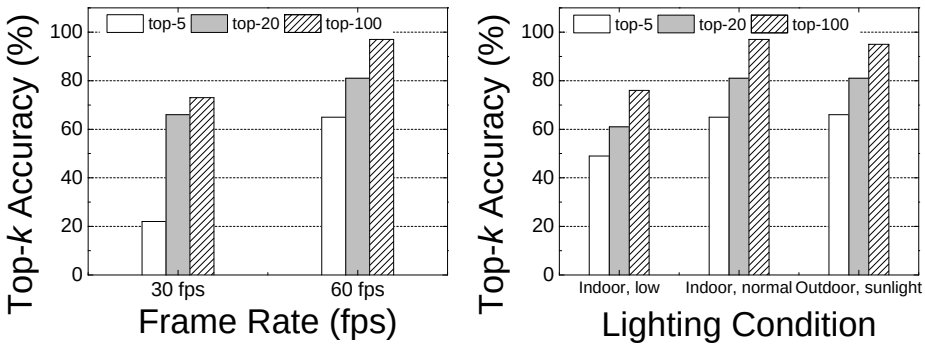
It is not surprising to see that EyeTell achieves much higher inference accuracy when the gaze traces of both eyes are used. The reason is that the gaze trace from one eye exhibits large noise due to the nature of human eyes while the gaze trace averaged from both eyes is much less noisy.

Impact of frame rate

Now we compare the inference accuracy of EyeTell under two frame rates for video recording, 30 fps and 60 fps. Since the default frame rate is 60 fps in our experiment, we down-sampled Ψ^l and Ψ^r in Section 5.5.3 by half to simulate the gaze trace obtained from 30-fps videos. As shown in Fig. 5.14a, EyeTell can yield better inference results under a higher frame rate. The reason is that the gaze traces from videos of higher frame rates are more accurate than those of lower frame rates, thus resulting in higher accuracy.

Impact of lighting conditions

In this experiment, we evaluate the impact of environmental lighting conditions on EyeTell. Three types of environments are investigated, including indoor normal light-



(a) Frame rate (b) Lighting condition

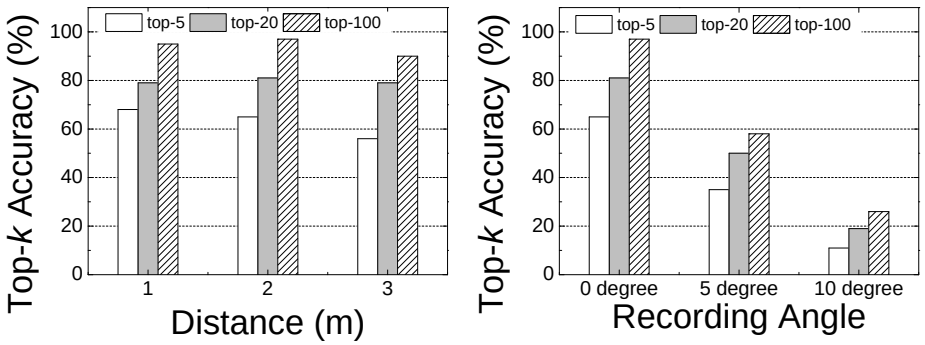
Figure 5.14: Impact of frame rate (left) and lighting condition (right).

ing with 300-360 lux illumination, indoor low lighting with 60-100 lux illumination, and outdoor daytime sunlight with around 1200 lux illumination. In each environment, each participant was asked to input 10 4-digit PINs on an iPhone 6s, and each PIN was input five times. As mentioned above, the PINs were generated randomly and then assigned to the participants. Fig. 5.14b summarizes the result for this experiment. EyeTell exhibits similar performance under indoor normal lighting and outdoor daytime sunlight conditions. However, the performance becomes worse in indoor low lighting environments. The reason is that low illumination in the shooting environment causes more noise in detected eye regions, thus degrading the accuracy of ellipse fitting for limbus and later gaze trace extraction.

Impact of recording distance

In this experiment, we evaluate EyeTell when the recording distance is 1m, 2m, and 3m, respectively. In each scenario, each participant was asked to input 10 4-digit PINs on an iPhone 6s, and each PIN was input five times. The PINs were generated randomly and then assigned to the participants. We show the result in Fig. 5.15a. As we can see, EyeTell has similar performance when the distance is 1m or 2m. The

slight performance degradation when the distance is 3m can be attributed to the larger zoom-in setting from a longer shooting distance. As a result, the captured video may be more sensitive to small head movements of the victim. However, we believe that the impact of the recording distance can be very limited if the attacker has more advanced camcorders.



(a) Recording distance (b) Recording angle

Figure 5.15: Impact of recording distance (left) and angle (right).

Impact of recording angle

In this experiment, we study the performance of EyeTell when the recording angle is 0°(the default), 5°, or 10°, respectively. In each scenario, each participant was asked to input 10 4-digit PINs on an iPhone 6s. Each PIN was input five times. The PINs were generated randomly and then assigned to the participants. Fig. 5.15b shows the results. As expected, the inference accuracy quickly decreases as the recording angle increases. This is mainly due to two reasons. First, the gaze tracking method [108] EyeTell adopts assumes that the recording angle is zero. Second, when the recording angle increases, the recorded video may not be able to capture the limbus of both eyes. Accurate gaze trace extraction under arbitrary recording angles (or equivalently arbitrary head postures) is very challenging and requires more advanced

gaze tracking methods. We plan to look further into this issue in our future work. Note that the attacker with an advanced camcorder may not have much difficulty achieving a near-zero recording angle in practice from a long distance to the victim.

5.6.8 Computational Time

We implemented EyeTell in two components. The first one is for gaze trace extraction implemented in C++, and the second for trace decoding implemented in Matlab. We run the experiments on a DELL desktop with 2.67 GHz CPU, 9 GB memory, and Windows 10 64-bit Professional. In the experiments, it takes less than 40s to generate a gaze trace from an input video. For trace decoding, the most time-consuming part is to generate the candidate set in Section 5.5.4, which is jointly determined by the number of segments and the number of candidates for each segment. Most PINs and lock patterns are associated with a few segments. For example, it takes less than 1s to generate the candidate set for a 4-digit PIN. In contrast, it takes about 40min for an English word with 13 letters. Overall, the computational time incurred by EyeTell is quite affordable for a determined adversary.

5.7 Discussion

In this section, we discuss the limitations of EyeTell and point out possible countermeasures.

5.7.1 Limitations

First, the inference accuracy of EyeTell is slighter lower than that of other video-based inference attacks [58, 133], especially for the alphabetical keyboard. There are two main reasons. First, other attacks use more direct observations about the keystrokes, such as the device's backside motion [58] and the victim's finger movement

[133]. In contrast, the gaze trace that EyeTell exploits only contains indirect keystroke information which is much more noisy and instable. Second, the efficacy of EyeTell on the alphabetical keyboard is largely limited by the uncertain number of keystrokes. We plan to explore extra side information such as eye fixation time in our future work to have more accurate estimation of the number of keystrokes and thus improve the inference accuracy of Eyetell.

Second, EyeTell currently requires the video to be recorded within a small recording angle, e.g., less than 5°based on our experiments. While such small recording angles make EyeTell detectable by vigilant users in uncrowded space, EyeTell is likely to succeed in crowded areas. This limitation can be alleviated by using more advanced camcorders or employing more advanced gaze tracking methods that are less sensitive to the victim's head posture. With better optics, the attacker can record the video from a longer distance. In addition, Gaze tracking based on machine learning [144] has shown to be effective even under different recording angles. We intend to explore this direction in our future work.

Finally, our experiment scale is comparable to that in the most recent work [133] but still limited. Though costly, larger-scale experiments may further evidence the efficacy of EyeTell.

5.7.2 Countermeasures

Since the only information EyeTell uses for keystroke inference is a video of the victim's eyes, mobile users should be alert when they input important sensitive information on their touchscreen devices. The following countermeasures can be adopted to thwart EyeTell. The most effective way against EyeTell is to prevent the attacker from video-recording the victim's eyes. For example, the user can wear sunglasses with dark colors to hide his gaze trace. In addition, users can input keystrokes with-

out looking at the keys so that the gaze trace extracted by EyeTell is irrelevant to keystrokes. However, this method may be practical only when the user incurs a small number of keystrokes, e.g., 4-digit PINs. Finally, sophisticated users can increase their typing speed on the touchscreen. In case that the frame rate of the attacker's camcorder is not high enough, the extracted gaze trace should be much less accurate and noisy, therefore degrading the inference result.

5.8 Conclusion and Future Work

In this chapter, we introduced EyeTell, a video-based keystroke inference attack framework to infer the victim's typed input from a video capturing his eyes. We adopted a user-independent model-based gaze tracking method to obtain a gaze trace of the victim's eyes and designed novel decoding algorithms to infer the typed input. We confirmed the high efficacy of EyeTell via extensive experiments on iOS and Android devices under various circumstances.

We plan to improve EyeTell in three directions in the future. First, we intend to develop novel gaze tracking methods that are less sensitive to the victim's head posture, which will greatly enhance EyeTell's applicability. Second, we will investigate novel methods to determine the number of keystrokes in order to improve the inference accuracy of EyeTell on alphabetical keyboards. Finally, we plan to evaluate EyeTell in a larger scale.

Chapter 6

WEARAUTH: SECURE AND USABLE WEARABLE AUTHENTICATION VIA ACOUSTIC SENSING

6.1 Overview

In this chapter, we propose WearAuth, a secure and usable technique to authenticate wearable devices via acoustic sensing. WearAuth targets wearable devices with an embedded speaker and one or more microphones. To authenticate a user, WearAuth invokes the speaker on the wearable to transmit inaudible acoustic signals. In the meantime, the user moves his finger to draw a pre-defined password pattern near the wearable device either in the air or on a solid surface. By analyzing the acoustic signals received by the microphone(s) on the wearable device, WearAuth can extract a 1D or 2D trace from the user's finger movement. If the extracted trace is classified legitimate by a pre-trained classifier, the user is considered authentic and otherwise denied access to the wearable device.

WearAuth has significant advantages over current authentication techniques for wearable devices. First, WearAuth explores tiny and cheap microphones and speakers that have been widely available or can be easily embedded onto wearable devices of different form factors, so it is widely applicable to a wide range of wearable devices with or without a touchscreen. Second, WearAuth allows a user to draw a self-chosen password pattern in a less-confined manner, e.g., in the air, on a desk, on the back of his hand, or even from a distance (say, 0.5 meter), so it is highly usable to a large range of users including senior citizens, children, visually impaired users, etc.

We conducted comprehensive experiments on a Samsung Galaxy S5 to demonstrate the feasibilty of WearAuth, involving six participants acting as both users and attackers. In Section 6.4, we name WearAuth as WearAuth-R when 1D traces are extracted and as WearAuth-S when 2D ones are extracted. Our experimental results show that WearAuth-R achieves 93% true-positive rate (TPR) and 97% true-negative rate (TNR) at the same time while WearAuth-S 91% TPR and 97% TNR. The equal-error rates (EER) for WearAuth-R drops from 7% to 9.4% under shoulder-surfing attack while that for WearAuth-S from 9.5% to 12.2%. Therefore, we believe that WearAuth achieves both high security and usability and shows strong robustness under shoulder-surfing attacks.

The rest of the chapter is organized as follows. Section 6.3 introduces the acoustic sensing technique behind our system. Section 6.4 details the WearAuth design. Section 6.5 presents the experimental evaluation. Section 6.2 presents the related work. Section 6.6 concludes this chapter.

6.2 Related Work

In this section, we discuss the prior work most related to our system in two directions: wearable device authentication and acoustic sensing on mobile device.

6.2.1 Wearable Device Authentication

Essentially, wearable devices fall into the general scope of mobile devices. There exist a large body of existing authentication methods developed for smartphones and tablets. Among these methods, some use touchscreen as the input interface, such as PIN and pattern lock. These touchscreen-based methods can be potentially applied to wearable devices with touchscreens. More recent methods, e.g., Touch ID [145] and Face ID [146] on iOS platform, use advanced sensors like fingerprint sensor and

camera for user input. Due to the high cost and large footprint of these sensors, these biometrics-based authentication methods are not suitable for most COTS wearable devices.

In the following, we focus on recent wearable device authentication methods which require only low-cost and small-footprint sensors rather than expensive or large-footprint ones. Examples of the former sensors are accelerometer, gyroscope, microphone, and small touchscreen while examples of the latter ones are fingerprint sensor and camera.

Researchers proposed to use accelerometer and gyroscope to extract a user's hand movement (i.e., gesture) for authentication. In most cases, these systems are designed for wearable devices on wrist or finger. In [147], Wang et al. proposed to extract free-form hand movements as a user's behavioral biometrics compared to extracting template-based hand movement in most previous work. In [148, 149, 150, 151], the authors confirmed the security and usability of hand-gesture-based authentication methods by implementation and evaluation through extensive experiments. In [152], Roshandel et al. presented a system that utilized their self-built finger ring to extract a 3D handwriting trace for user authentication. In [151], Liu et al. investigated the influence of 13 machine learning models on the overall performance of gesture-based authentication systems on smartphones or tablets. To sum up, gesture-based authentication schemes are applicable to wearable devices, taking both security and usability into consideration. They are especially suitable for those worn on wrist or finger. The advantages of these schemes are that they are convient to use for most users and robust to popular attacks such as shoulder-surfing attack and imitation attack. Meanwhile, one disadvantage is that it may be a little awkward for a user to perform her/his gesture in public places.

Speaker and microphone have also been used for authentication of wearable devices. In [153], Gong et al. presented a new protocol to authenticate an IoT device by measuring its distance from a known trusted mobile device. In their protocol, distance measurement was achieved by measuring the propogation time of acoustic signals. In [154], Chauhan et al. found that BreathPrint, which they referred to as the frequency spectrum of a user's breath, could be used as a new behavioral biometrics for authentication on mobile devices. In their following work [155], they adopted Recurrent Neural Networks (RNN) for their system and tested the performance on different devices including smartphone and smartwatch.

In [156], Hutchins et al. showed that a user's rhythmic tapping on the touchscreen of a smartwatch could be used to authenticate the user. Compared to conventional touchscreen-based methods like PIN and pattern lock, their method is easier to use (i.e., the true positive rate is high) and more secure under shoulder-surfing attack and imitation attack.

6.2.2 Acoustic Sensing on Mobile Device

Acoustic sensing has been actively researched during the past five years. A strong motivation behind this is that the speaker and microphone for acoustic sensing are affordable for most mobile devices, even low-cost IoT devices while acoustic sensing is very promising in many application scenarios such as device pairing, networking, device-user interaction, gaming. In the following, we focus on the acoustic sensing systems which track a user's hand or finger movements in a device-free manner, i.e., the user does not need to hold or wear any device on/with her/his moving hand or finger.

Hand or finger movement tracking are essentially measuring the distance between a mobile device and a moving object. FingerIO in [157] aims to find the echo of

an OFDM symbol corresponding to the finger position. By doing this, FingerIO can translate the echo into the distance of a user's moving finger. LLAP in [158] makes use of coherent detection to recover phase change of received acoustic signal. Since phase change is directly caused by a targeted moving object (in their case, a user's finger), LLAP can calculate the corresponding distance from the phase change. Finally, Strata in [3] applies a similar technique as LLAP, however, with an important difference. LLAP assumes that there is no moving object except the user's finger in the environment while Strata does not. Strata tackles the challenge of unstatic environment by estimating the channel impulse response (CIR) of the acoustic channel and selecting the exact tap corresponding to the user's moving finger.

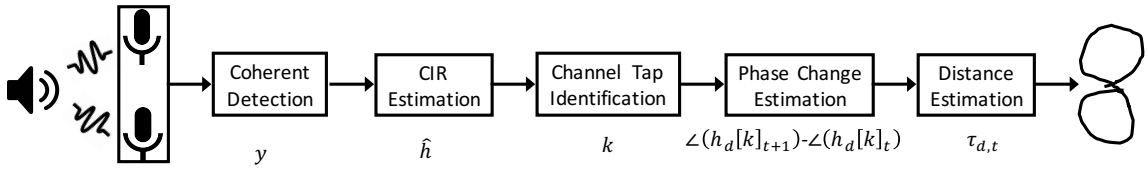
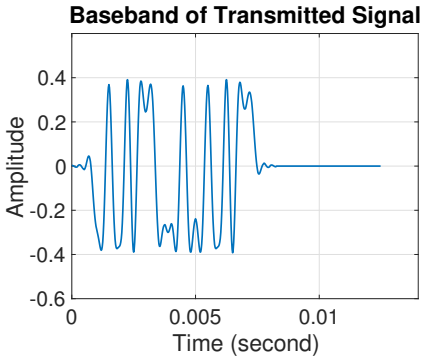
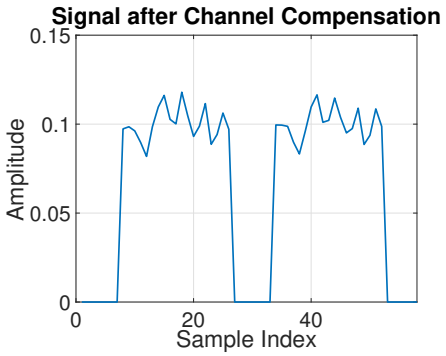


Figure 6.1: Flow chart of acoustic sensing in [3].

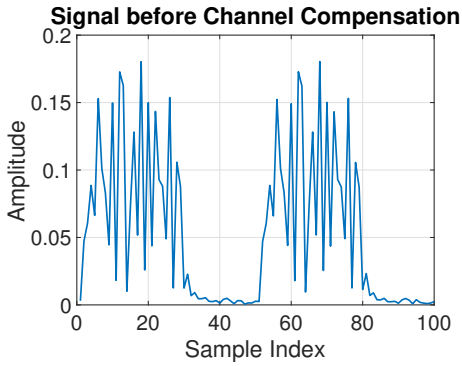
Figure 6.2: Speaker and microphones on Galaxy S5.



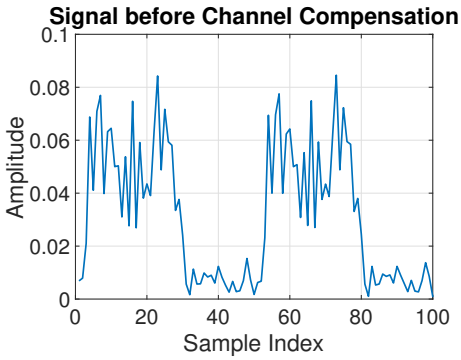
(a) Transmitted bassband signal.



(b) Decoded baseband signal.



(c) Received signal of bottom microphone.



(d) Received signal of top microphone.

Figure 6.3: Transmitted and received signals.

6.3 Acoustic Sensing for Motion Sensing

WearAuth uses the speaker and microphone(s) on a wearable device to track a user's finger movement. In this section, we briefly introduce the underlying motion-tracking technique (called Strata [3]) for self-containment and the minor changes we made for our context.

Strata uses the speaker and microphone on the same mobile device for motion tracking. Assume that the user's finger is the nearest moving object to the mobile device. The speaker repeatedly transmits a known acoustic sequence, and the microphone collects acoustic signals. We can estimate the channel impulse response (CIR)

from the transmitted and received signals, which can characterize the multipath effect in the environment [159]. Suppose that the acoustic channel between the speaker and microphone has L paths. Let τ_i and a_i denote the delay and attenuation coefficient of the i-th path, respectively. CIR can be denoted in a discrete form as

$$h[j] = \sum_{i=1}^{L} a_i e^{-j2\pi f_c \tau_i} \operatorname{sinc}(j - \tau_i W), j \in [1, N], \tag{6.1}$$

where f_c is the center frequency, W is the bandwidth, $\operatorname{sinc}(t) = \frac{\sin(\pi t)}{\pi t}$, and N is the number of taps. In our implementation, $f_c = 20\text{kHz}, W = 4\text{kHz}, N = 10$. Therefore, the frequency range of the acoustic signal is between 18kHz and 22kHz. Each $h[j]$ $(j = 1, 2, \ldots, N)$ is referred to as a tap of CIR. Note that N has nothing to do with L. From Equation (6.1) and the property of the sinc function, $h[j]$ is determined by a few (normally two or three) among the L paths, each of which makes the terms $(j - \tau_i W)$ sufficiently small. We can image that the specific path of which the transmitted signal is reflected by the user's finger corresponds to a specific delay, which affects only one CIR tap. Therefore, we can find the corresponding CIR tap and then try to infer the finger movement from it.

Finger movement tracking is based on the intuition that the position change of the user's finger incurs a phase change in the corresponding CIR tap, which is assumed to be $h[k]$ and relate to the L-th path without loss of generality. Suppose we track the user's finger movement between $(t-1)$-th and t-th frames. In our implementation, the frame duration is 12.5ms. Considering that the typical finger movement speed of a user is no more than $0.05\,\mathrm{m\,s^{-1}}$, the difference of $\tau_i W$ between two frames is no more than $\frac{0.05 \times 12.5}{v} W = 0.0092$, where $v = 340\,\mathrm{m\,s^{-1}}$ is the propagation speed of sound in the air. Therefore, it is safe to expect that the $(j - \tau_i W)$ term almost does not change between two frames. For simplicity, we merge $\operatorname{sinc}(j - \tau_i W)$ into a_i in Equation (6.1) and reuse a_i to denote the new attenuation coefficient here after. As

a result, we have the following equation for CIR.

$$h[j] = \sum_{i=1}^{L} a_i e^{-j2\pi f_c \tau_i}, j \in [1, N], \tag{6.2}$$

We further denote $h[k]$ estimated from the $(t-1)$-th and t-th frames by $h[k]_{t-1}$ and $h[k]_t$, respectively, which are defined based on Equation (6.2) as

$$\begin{aligned} h[k]_{t-1} &= \sum_{i=1}^{L} a_i e^{-j2\pi f_c \tau_{i,t-1}} \\ &= \sum_{i=1}^{L-1} a_i e^{-j2\pi f_c \tau_{i,t-1}} + a_L e^{-j2\pi f_c \tau_{L,t-1}} \end{aligned} \tag{6.3}$$

and

$$\begin{aligned} h[k]_t &= \sum_{i=1}^{L} a_i e^{-j2\pi f_c \tau_{i,t}} \\ &= \sum_{i=1}^{L-1} a_i e^{-j2\pi f_c \tau_{i,t}} + a_L e^{-j2\pi f_c (\tau_{L,t-1} + \tau_{d,t})}. \end{aligned} \tag{6.4}$$

Here $\tau_{d,t}$ is the delay difference incurred by the user's finger movement between frame $t-1$ and frame t, i.e., $\tau_{d,t} = \tau_{L,t} - \tau_{L,t-1}$. If $\tau_{d,t}$ is known, the corresponding finger movement can be easily calculated as $d_t = \upsilon \tau_{d,t}$.Let $h_d[k]_t = h[k]_t - h[k]_{t-1}$. The phase of $h_d[k]_t$ can be deduced from Equation (6.3) and Equation (6.4) as

$$\angle(h_d[k]_t) = \angle(e^{-j2\pi f_c \tau_{L,t-1}}) + \frac{\angle(e^{-j2\pi f_c \tau_{d,t}})}{2} + \frac{\pi}{2}. \tag{6.5}$$

Similarly, we have the following result for frames t and $t+1$,

$$\angle(h_d[k]_{t+1}) = \angle(e^{-j2\pi f_c \tau_{L,t}}) + \frac{\angle(e^{-j2\pi f_c \tau_{d,t+1}})}{2} + \frac{\pi}{2}. \tag{6.6}$$

From the above two equations, we observe that the subtraction between $\angle(h_d[k]_{t+1})$ and $\angle(h_d[k]_t)$ reveals $\tau_{d,t}$, i.e.,

$$\begin{aligned} \angle(h_d[k]_{t+1}) - \angle(h_d[k]_t) &= \angle(e^{-j2\pi f_c \tau_{L,t}}) - \angle(e^{-j2\pi f_c \tau_{L,t-1}}) \\ &\quad + \frac{(\angle(e^{-j2\pi f_c \tau_{d,t+1}}) - \angle(e^{-j2\pi f_c \tau_{d,t}}))}{2} \\ &\approx \angle(e^{-j2\pi f_c \tau_{d,t}}). \end{aligned} \tag{6.7}$$

The above approximation assumes that $\tau_{d,t+1} = \tau_{d,t}$. Using the above equation, we can obtain $\tau_{d,t}$ as

$$\tau_{d,t} = \frac{\angle(e^{-j2\pi f_c \tau_{d,t}})}{2\pi f_c} \approx \frac{\angle(h_d[k]_{t+1}) - \angle(h_d[k]_t)}{2\pi f_c}. \tag{6.8}$$

The whole process of finger movement tracking through acoustic sensing on mobile devices includes the following three main steps. The flow chart is illustrated in Figure 6.1, and more details can be found in [3].

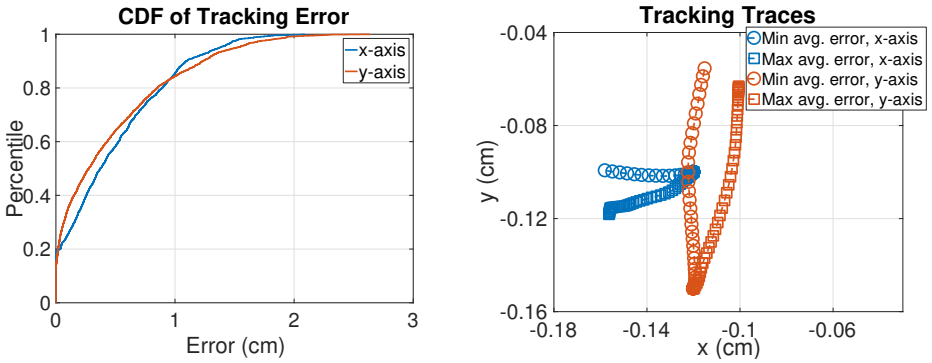
Step One: Acoustic Signal Transmission. The speaker transmits a 26-bit GSM training sequence repeatedly [160]. We achieve this by storing the sequence as a Waveform Audio (WAV) file in the format of 16-bit Pulse Coded Modulation (PCM) and playing it through the device's speaker continuously. Figure 6.3a shows the transmitted acoustic signal in the low-frequency baseband after pulse shaping and filtering.

Step Two: Channel Estimation. Our system uses the microphone(s) to collect the reflected multipath acoustic signals, converts them to baseband signals through demodulation, and uses Least-Square (LS) [159] for channel estimation. Specifically, given a converted baseband signal $\mathbf{y} = \{y_1, y_2, \ldots, y_{N+P}\}$ and the circulant training matrix $\mathbf{M} \in \mathcal{R}^{P \times N}$, the estimated channel can be derived as $\hat{\mathbf{h}} = (\mathbf{M}^H\mathbf{M})^{-1}\mathbf{M}^H\mathbf{y}_N$, where $\mathbf{M} = \begin{pmatrix} m_N & m_{N-1} & m_{N-2} & \cdots & m_1 \\ m_{N+1} & m_K & m_{N-1} & \cdots & m_2 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ m_{N+P} & m_{N+P-1} & m_{N+P-2} & \cdots & m_{P+1} \end{pmatrix}$, $\mathbf{y}_N = \{y_{N+1}, y_{N+2}, \ldots, y_{N+P}\}$, N is the number of taps to estimate (or the memory length), P is the reference length, and $N + P$ is the length of training sequence. In practical, N determines the maximum tracking distance of our acoustic sensing system. A larger N corresponds to a larger tracking distance, however, results in a less reliable CIR estimation as well. In our implementation, we choose $N = 10$ to strike the

balance between tracking distance and estiamtion reliability. $N = 10$ corresponds to a tracking distance up to 0.5 m, adequate in most day-to-day usage. Figure 6.3c and Figure 6.3d are the same baseband frames received by microphone A and B of Samsung Galaxy S5 illustrated in Figure 6.2, respectively. Obviously, the received signal strengths at the two microphones are very different because they are designed for different purposes. Figure 6.3b shows the same frames after channel compensation which can be further used to decode transmitted bits.

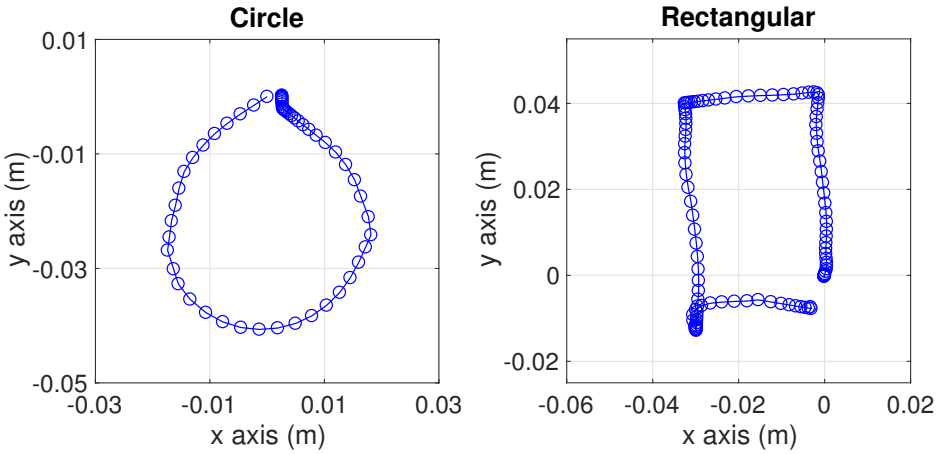
Step Three: Finger Movement Tracking. Assume that the initial position is the original point, i.e., $(0, 0)$ on a 2D XY plane. Using Equation (6.7), our system is able to calculate the distance change between two received frames, i.e., $\tau_{d,t}$. Specifically, in the case of one microphone used, say, microphone A, we have $\tau_{d,t}^{A}$ across different t while in the case of both microphone A and B, we have both $\tau_{d,t}^{A}$ and $\tau_{d,t}^{B}$. With the obtained $\tau_{d,t}$, our system is able to calculate a 1D or 2D trace.

Benchmark. Here we use simple experiments on a Samsung Galaxy S5 to showcase the performance of finger motion tracking in our system. In our experiments, we put the smartphone on a table and asked a user to draw a 1D line along the x or y axis, or a 2D shape on the table. For 1D lines, we collected 50 traces along the x axis and 50 along the y axis. The examples of 1D tracked traces are shown in Figure 6.4b. The traces with circle markers are the two with the minimum average tracking error while those with square markers with the maximum average tracking error. We also plot the cumulative distribution function (CDF) of the tracking errors for 1D lines in 6.4a. Our implementation achieves an average measurement error of around 0.4 cm which is similar to 0.3 cm in [3] for 1D lines. In addition, we randomly select two tracked 2D traces (circle and rectangular) and plot them in Figure 6.5. Their average tracking errors are 0.52 cm and 0.41 cm, respectively.



(a) CDF of tracking error. (b) Examples of tracking traces.

Figure 6.4: Performance of acoustic tracking system.



(a) Circle. (b) Square.

Figure 6.5: Examples of tracked shapes.

6.4 WearAuth Design

In this section, we give an overview of WearAuth and then detail each of its functional modules.

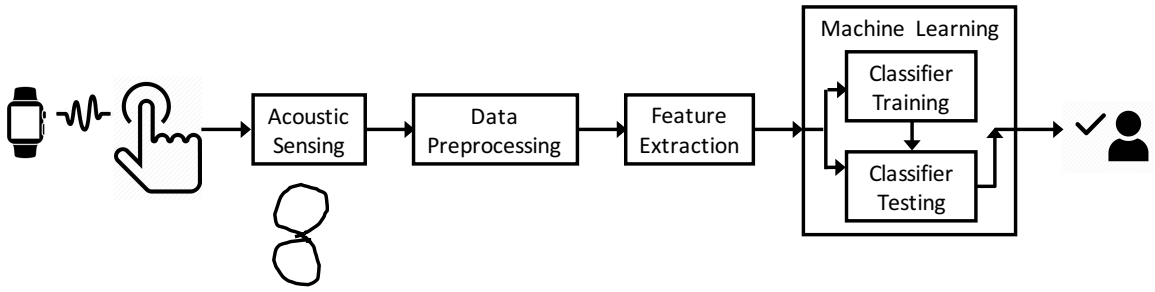


Figure 6.6: The flow chart of WearAuth.

6.4.1 Overview

We use a smartwatch as an example to explain how WearAuth works. Assume that a user wears his smartwatch on one wrist or puts it on a table. To unlock the smartwatch, he uses a finger on the wrist without the smartwatch to draw a password gesture near the smartwatch. WearAuth supports two types of gestures: a Type-I gesture refers to a sequence of 1D back-and-forth lines, and a Type-II gesture corresponds to a self-defined or fixed 2D shape. Which type of gestures to use depends on how many microphones a wearable device has and also the user's preference. In particular, if there is only one microphone, only Type-I gestures can be tracked and used; if there are two microphones, both Type-I and Type-II gestures can be tracked and used. Our system has little restriction on where the user performs his password gesture. For example, if he wears the smartwatch on his left wrist, he may do the drawing on the back of his left hand or even in the air. If he puts the smartwatch on a table instead, he may just draw his gesture right on the table.

As in most authentication systems, WearAuth involves two phases: an enrollment phase and a verification phase. The purpose of the enrollment phase is for the system to collect the legitimate user's data to train a supervised classifier. In this phase, the user decides his password gesture and then repeats it multiple times for the system

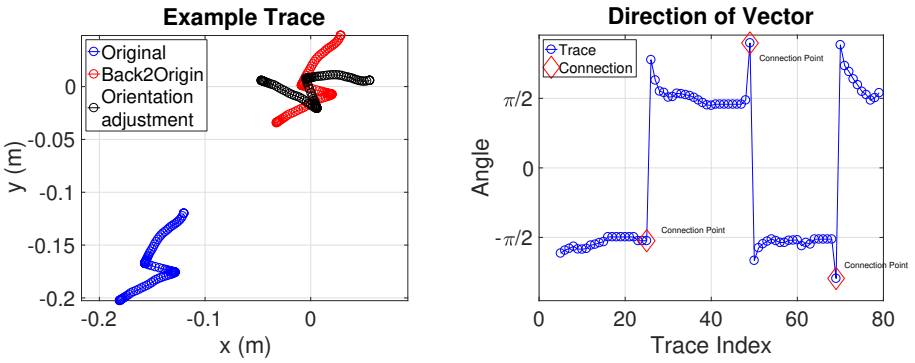
to collect sufficient data for classifier training. Specifically, when a Type-I gesture is used, the user chooses a familiar rhythm and follows it to draw back-and-forth lines. Such a rhythm can be from an existing song the user knows or even self-composed by himself. Similarly, when a Type-II gesture is used, the user chooses a 2D password shape either from a library of pre-selected patterns or self-defined by himself. During the verification phase, the user draws his password gesture to unlock the smartwatch.

Figure 6.6 depicts the flow chart of WearAuth. We briefly explain each functional module as follows. WearAuth collects acoustic signals reflected from a user's finger movement. The Acoustic Sensing (AS) module uses the motion tracking technique introduced in Section 6.3 to infer the user's finger movement trace. Then the Data Processing (DP) module applies averaging and normalization to the recovered finger trace. Next, the Feature Extraction (FE) module is invoked to extract a vector from each finger trace. Finally, the Machine Learning (ML) module uses the extracted feature vector(s) either to train a classifier in the enrollment phase or to determine whether the user is legitimate during the verification phase. Since the FE and ML modules for Type-I and Type-II gestures are significantly different, we name WearAuth for Type-I and Type-II gestures as WearAuth-R and WearAuth-S, respectively, where R and S are short for rhythm and shape, respectively.

6.4.2 Data Processing

The DP module is to reduce the impact of tracking errors in the AS module on the overall system performance. We accomplish this goal with three steps in sequel: trace smoothing, orientation adjustment, and normalization.

The first step is to smooth the finger trace produced by the AS module. The raw trace inevitably exhibits irregular ups and downs due to various factors. For example, the user's finger movement naturally includes tiny variations such as tiny



(a) An example trace. (b) The trace after processing.

Figure 6.7: Find "feature points" from a trace.

backward and forward displacements, which would affect the phase change of received signals; random environmental changes (such as nearby object movement) may affect the phase change as well; and the LS channel estimation is always left with residue errors [159]. For the other thing, we rely on the assumption that $\tau_{d,t+1} = \tau_{d,t}$ in Equation (6.7) in order to obtain the approximation that $\angle(h_d[k]_{t+1}) - \angle(h_d[k]_t) \approx \angle(e^{-j2\pi f_c \tau_{d,t}})$. Essentially, this assumes that the user maintains a similar finger moving speed when he is drawing 1D lines or 2D shapes. Such a requirement is easy to meet in typical finger tracking scenarios but more difficult for our user authentication. This is because WearAuth users usually have their own ways to draw lines or shapes and the finger movement speed usually changes across a trace. Another issue is that we need to identify the correct CIR tap(s) corresponding to the physical location of the user's finger in every frame. In practice, the acoustic sensing system may pick up a nearby tap by mistake, resulting in some abrupt points in the calculated trace. We further use a five-point slow-moving-average (SMA) filter as the first remedy to obtain a smoothed trace.

The second step is to adjust the orientation of the smoothed finger trace in order to loose the constraint on how the user draws his gesture. In this way, the user can

perform his password gesture in an arbitrary orientation, leading to higher usability. We achieve this goal as follows. First, we calculate the center point of a trace as the average of all points and then move the trace such that the center point becomes $(0, 0)$ in the 2D plane. Next, we calculate the angle between the first and last points and then rotate the trace around its center point, i.e., $(0, 0)$, until the angle becomes zero. Essentially, we apply a rotation matrix $\begin{bmatrix} \cos\alpha & \sin\alpha \\ -\sin\alpha & \cos\alpha \end{bmatrix}$, in which α is the above mentioned angle.

The third step is to normalize the adjusted finger trace so as to reduce the impact of varying gesture dimensions which may naturally occur for each different authentication instance. Intuitively, the traces drawn by the same user tend to be similar from one to another, but their sizes inevitably vary within a certain range. This issue can be mitigated by applying normalization to each finger trace. Assume that the minimum and maximum x-axis value of a trace is $x_{\min}$ and $x_{\max}$. We apply a $-x_{\min}$ displacement along the x-axis and then a $\frac{1}{x_{\max}-x_{\min}}$ magnification factor to the trace. As a result, the maximum difference along x-axis becomes 1.

WearAuth-S adopts all the three steps above, while WearAuth-R only uses the first step. As an example, Figure 6.7a shows a trace and its corresponding transformations after orientation adjustment, and Figure 6.7b shows the processed trace after normalization.

6.4.3 Feature Extraction

Case 1: WearAuth-R. Imagine that a user follows his chosen rhythm to draw 1D back-and-forth lines. For security concerns, our system enforces a minimum number of notes in a user's rhythm, say 4, each of which is a pitch lasting a certain duration. The number of notes and the duration of each note together determine the security

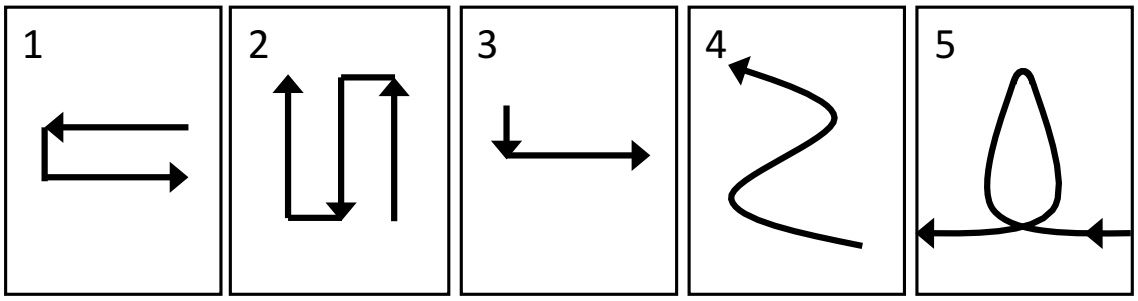
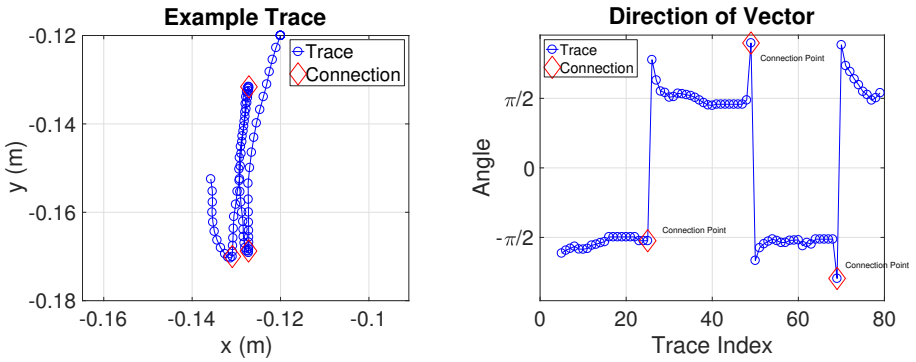


Figure 6.8: Gesture shapes in our library.

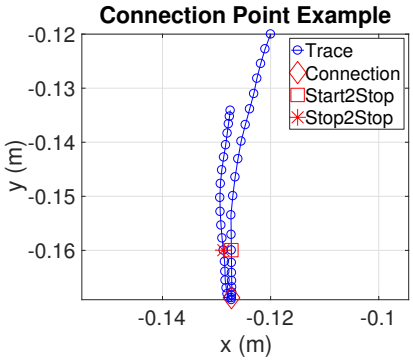
and usability of WearAuth-R. Intuitively, the more notes, the more difficult for an attacker to emulate the legitimate user (higher security), the longer time needed to finish the gesture (lower usability), and vice versa.

Now we describe how to infer the note durations from the finger trace. According to our observation, some users start drawing a line and stop at its end for sufficient time to finish the note, while others spend the whole duration drawing a line and have very short pause at its end. In either way, we can expect some very close points, if not completely overlapping, at the end of either a forward or backward line, which are referred to as "lingering" points. Then we want to find three types of "feature" points among these lingering points: start-to-linger, stop-to-linger, and connection points. The start-to-linger and stop-to-linger points correspond to the first and last of all lingering points, respectively, while the connection point is the one with the largest direction change. We regard all the points between the start-to-linger and stop-to-linger points in the same set of feature points as a "lingering segment". Any direction change in the finger trace—either from backward to forward or the reverse—leads to a new set of three features points. In practice, we first look for a connection point and later the corresponding start-to-linger and stop-to-linger point among the same group of lingering points because the connection point is the easiest to spot. More specifically, the connection point can be easily singled out by calculating the angle



(a) An example trace.

(b) Direction between two adjacent points.



(c) A zoom-in segment.

Figure 6.9: Find "feature points" from a trace.

between any two adjacent points in a trace and selecting the one with the largest angle change. After that, we start from the spotted connection point and search backward along the trace to find the start-to-linger point and forward to locate the stop-to-linger point. The last point in the backward search whose Cartesian distance from the connection point falls below a system threshold δ_p is the start-to-linger point, and the last point in the forward search satisfying the same Cartesian distance criterion is the stop-to-linger point. In our implementation, we use a $\delta_p = 0.75\,\text{cm}$.

Figure 6.9 exemplifies how to find feature points. In particular, Figure 6.9a is the example trace from which we need to identify three sets of feature points. Figure 6.9b

plots the angles between two adjacent points on the trace. We can easily identify three connection points, represented by red diamonds. Figure 6.9c plots a short segment obtained by zooming in the example trace. The points denoted by a red square and a red star are the start-to-linger and stop-to-linger point, respectively.

Next, we extract a feature vector from each finger trace. Each forward or backward line is called a "movement" which usually corresponds to a note in the chosen rhythm. Assume that the user trace consists of n movements (or equivalently n notes in the user's chosen rhythm) and N_p points. We denote the trace by $\{\mathbf{p}, \mathbf{t}\}$, where $\mathbf{p} = \{p_i = (x_i, y_i) | 1 \leq i \leq N_p\}$ is the set of point coordinates, and $\mathbf{t} = \{t_i | 1 \leq i \leq N_p\}$ is the set of timestamps corresponding to each point. Following the above procedure, we can obtain a set of point indexes $\{(\eta_{j,1}, \eta_{j,2}, \eta_{j,3}) | 1 \leq j \leq n+1\}$, where $\eta_{j,1}, \eta_{j,2}$, and $\eta_{j,3}$ are the indexes of the j-th start-to-linger, stop-to-linger, and connection points, respectively. For convenience only, we define $\eta_{1,1} = \eta_{1,2} = 1$ and $\eta_{n+1,2} = \eta_{n+1,3} = N_p$. Then the following time-domain features are extracted for WearAuth-R:

- $\{\beta_j = t_{\eta_{j,2}} - t_{\eta_{j,1}} | j \in [1, n+1]\}$: β_j is the duration of j-th lingering segment.
- $\{\alpha_j = t_{\eta_{j+1,3}} - t_{\eta_{j,3}} | j \in [1, n]\}$: α_j is the duration of j-th movement.
- $\{\gamma_j = \frac{\alpha_j}{\alpha_1} | j \in [1, n]\}$: γ_j is the ratio between the duration of j-th movement and that of the 1st one.

Case 2: WearAuth-S. In WearAuth-S, the user draws a 2D password shape to unlock his wearable device. He can either choose a shape from the preloaded system library or come up with his own one like some numeric digit, an English letter, or a signature. Figure 6.8 lists five recommended shapes in [39]. These shapes were found very easy to repeat by the same user. In addition, different users tend to draw the

same shape in diverse ways that can well distinguish different users. The following metrics are calculated from the finger trace output by the DP module.

- **Trace coordinates** $\mathbf{p} = \{(x_i, y_i) | i \in [1, N_p]\}$.

- **Curvature**. It is computed at point (x_i, y_i) by

$$\kappa_i = \frac{4\Psi_i^y \Delta_i^x - 4\Psi_i^x \Delta_i^y}{((\Delta_i^x)^2 + (\Delta_i^y)^2)^{3/2}}, \tag{6.9}$$

  where $\Delta_i^x = (x_{i-1} + x_{i+1})/2, \Delta_i^y = (y_{i-1} + y_{i+1})/2, \Psi_i^x = x_{i+1} - 2\overline{x}_i + x_{i-1}$, and $\Psi_i^y = y_{i+1} - 2\overline{y}_i + y_{i-1}$.

- **Velocity along the x-axis and y-axis.** We denote them by υ_x and υ_y, respectively, and compute them as

$$\upsilon_{x,i} = \frac{x_{i+1} - x_i}{t_{i+1} - t_i} \text{ and } \upsilon_{y,i} = \frac{y_{i+1} - y_i}{t_{i+1} - t_i}. \tag{6.10}$$

- **Acceleration along the x-axis and y-axis.** We denote them by a_x and a_y, respectively, and compute as

$$a_{x,i} = \frac{\upsilon_{x,i+1} - \upsilon_{x,i}}{t_{i+1} - t_i} \text{ and } a_{y,i} = \frac{\upsilon_{y,i+1} - \upsilon_{y,i}}{t_{i+1} - t_i}. \tag{6.11}$$

- **Direction.** It is defined as the angle of the vector between two adjacent points and calculated as

$$\theta_i = \arctan(\frac{y_{i+1} - y_i}{x_{i+1} - x_i}). \tag{6.12}$$

One challenge for Feature Extraction (FE) is that the number N_p of points varies from one trace to another even for the same user drawing the same gesture. As a result, the above metrics cannot be directly combined and fed into the ML module. We use a commonly-used solution to transform each metric above into a distance value in two steps. First, we want to select one trace from all training samples of the

same user as the template trace. For this purpose, we use the Dynamic Time Warping (DTW) algorithm [161] to compute the distance between every two traces for each of the eight metrics above. A smaller DTW distance suggests that the two training traces are similar with regard to the corresponding metric. Then we compute the average DTW distance for each training trace with all the other traces with regard to every metric, leading to eight distance values for each training trace. We further derive the average of its eight DTW distance values for each training trace, and the one with the minimum average is selected as the template trace. In the second step, we calculate eight DTW distance values between each non-template trace and the template trace in the form of $[d_x, d_y, d_\kappa, d_{v_x}, d_{v_y}, d_{a_x}, d_{a_y}, d_\theta]$.

6.4.4 Machine Learning

We adopt the popular support-vector machine (SVM) algorithm for classifier training and testing. In specific, there are two classes in our classifier: positive class and negative class. The samples of positive class come from the device user while those of negative class from other people, such as users of other devices or even attackers. After a classifier is built, we can then apply it to a new unlabelled sample to determine its legitimacy. If it is legitimate, the user is then authenticated. Vice versa. The details of our training and testing data set are in Section 6.5.3.

6.5 Performance Evaluation

This section evaluates the performance of WearAuth.

6.5.1 Adversary Model

We assume the following adversary model to evaluate the security and usability of WearAuth. The attacker has physical access to the victim's wearable device and

tries to unlock it with the full knowledge of how WearAuth works. We consider the following two specific attacks.

- **Random emulation**: The attacker knows how many notes are in the victim's rhythm for the 1D case or his password shape for the 2D case. But the attacker has no observation about how the victim actually performs the 1D or 2D gesture.
- **Shoulder surfing**: The attacker observes how the victim performs the 1D or 2D password gesture either in person or through a stealthy video recorder. He needs to guess the number of notes in the victim's rhythm for the 1D case or the corresponding password shape for the 2D case.

6.5.2 Performance Metrics

We use the following performance metrics.

- **Receiver Operating Characteristic (ROC) curve.** An ROC curve can be used to illustrate the performance of a binary classifier as its discrimination threshold changes. According to the definition in [104], we can obtain an ROC curve by plotting TPR (true-positive rate) with respect to FPR (false-positive rate) in various threshold settings.
- **Equal Error Rate.** When FPR (false-positive rate) is equal to FNR (false-negative rate), we call the corresponding error value as equal error rate (EER). It describes the best performance a classifier can achieve considering both FPR and FNR at the same time.

6.5.3 Experiment Setup

Implementation

We used a Samsung Galaxy S5 to demonstrate the feasibility of WearAuth. Although many COTS smartwatch models, e.g., Apple Watch 2 [162], Samsung Galaxy Gear [163], and Moto 360 [164], have two microphones, only one of them is programmable in most cases. Due to such a practical limitation, previous work on acoustic sensing either used a smartphone [158] or built a prototype [157] with the size similar to that of a smartwatch for feasibility demonstration. The main difference between smartwatches and smartphones when used for acoustic sensing is the device size, which nevertheless has little impact on the tracking results.

The following implementation details were used for acoustic sensing in our experiments. The passband for signal transmissions is 18 to 22 kHz, the center frequency is $f_c = 20$ kHz, and the total bandwidth is $W = 4$ kHz. We used BPSK modulation for acoustic signals, and the length of each BPSK symbol (i.e., 1 or -1) is 0.25 ms. The speaker repeatedly sends a known 26-bit GSM training sequence, which is also referred to as a frame. To avoid inter-frame interference, a fixed gap needs to be inserted. We achieved this by padding 24 zeros with the original sequence. Therefore, the final frame has 50 symbols and a duration of 12.5 ms. We also set $N = 10$ and $P = 16$ for LS estimation. Please refer to Strata in [3] for more implementation details.

Data collection

We recruited six participants in the experiments, including two females and four males. Their ages range between 20 and 30. The typical setup of our experiments is like this. We put the device on the table, with the screen facing upward. A participant

sat in front of the device and began to draw 1D lines or 2D shapes. She/he could do the drawing either on the table or in the air. Besides, they could choose the starting points of their drawing arbitrarily as long as the whole drawing was within 0.5m range. The experiments involved two steps. In the first, we asked each participant to draw lines in a forward and backward fashion following their own rhythms. Each participant used two rhythms and performed 25 line drawings (i.e., 1D patterns) for each rhythm. In the second step, we asked each participant to draw each shape in Figure 6.10 25 times.

We also collected some data to emulate the shoulder-surfing attack. For this purpose, we used a Google Nexus 6 to record a video capturing how each participant performed line or shape drawing during the experiments. We had one participant act as the attacker and the others as the victims. The attacker watched each victim's video for sufficiently long until confident of successful emulations. In addition, the attacker attempted each line or shape drawing for 10 times.

We eventually collected four datasets: $\mathcal{S}_{1D}$, $\mathcal{S}_{2D}$, $\mathcal{T}_{1D}$, and $\mathcal{T}_{2D}$. In particular, $\mathcal{S}_{1D}$ comprises the instances of 300 legitimate 1D drawings; $\mathcal{S}_{2D}$ consists of 750 instances of legitimate 2D drawings; $\mathcal{T}_{1D}$ and $\mathcal{T}_{2D}$ are composed of 600 1D and 1,500 2D instances of the shoulder-surfing attacks.

Training and testing procedure

The ratio between positive and negative samples in the training set is a critical factor in building a classifier. In most of our experiments, a training set usually consists of 15 positive samples and 35 negative ones, resulting in a 3 : 7 positive-negative ratio. All the remaining samples are used for testing. Consequently, a testing set usually consists of at least 10 positive samples and 90 negative ones. To obtain a classifier model, we ran 10-fold cross validation over a training set. Then we applied

the optimal one among the 10 folds to the corresponding testing set to derive TPR and FPR. For each selected rhythm (1D) or shape (2D) of a participant, we ran the above training-testing process 20 times to alleviate the impact of randomness in generating a training set. The ROC and EER results to report are the average over all experiment runs and all participants.

6.5.4 Experimental results

Number of positive samples

First, we examine the impact of the number of positive samples in a training set on EER. It is expected that EER drops with the increase of positive samples until it saturates to a certain level. We changed the number of positive samples from five to 15 while fixing the number of negative ones to 35 in a training set. The fold number for cross validation is five rather than 10. Figure 6.11 shows the relationship between EER and the number of positive samples. As we can see, EER drops from 14% to 7.2% when the number of positive samples increases from five to 15 for WearAuth-R. The result for WearAuth-S is similar where EER drops from 18% to 9.5%. We can also see that EER for WearAuth-R becomes saturated when the number of positive-samples is larger than 12. Similarly, the EER for WearAuth-S becomes saturated when the number of positive samples is larger than 14.

ROC performance

Now we show the ROC curves for the experimental process described in Section 6.5.3. Figure 6.12a shows the ROC curve of WearAuth-R. It is clear that WearAuth-R can achieve 93% TPR and 97% TNR simultaneously, which confirms its feasibility as a secure and usable authentication scheme. Figure 6.12b shows that WearAuth-S can achieve 91% TPR and 97% TNR simultaneously, which is comparable to the

performance of WearAuth-R. So both WearAuth-R and WearAuth-S can serve as a secure and convenient authentication method for wearable devices.

Performance under attacks

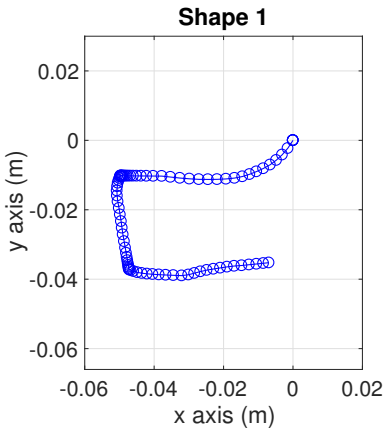
We first discuss the performance under random emulation attacks. According to the process of generating a training set for for a selected rhythm or shape in Section 6.5.3, each negative sample in both training and testing sets can also be regarded as a random-emulation attack. So the promising results reported in Section 6.5.4 have confirmed that WearAuth-R and WearAuth-S can both achieve high TPR and TNR under the random-emulation attack.

To evaluate the resilience to shoulder-surfing attacks, we labeled the attack samples as negative and added them to the testing set. Figure 6.13a and Figure 6.13b show the ROC curves of WearAuth-R and WearAuth-S under shoulder-surfing attacks, respectively. In contrast to the results in Figure 6.12a and Figure 6.12b, the performance of both WearAuth-R and WearAuth-S degrades a little, which is expected. More specifically, EER for WearAuth-R decreases from 7% to 9.4%,and EER for WearAuth-S from 9.5% to 12.2%. Given that conventional wearable authentication methods like PIN or pattern lock have almost no resilience to shoulder-surfing attacks, we believe that our highly usable system is sufficiently effective and raises the bar for shoulder-surfing attacks.

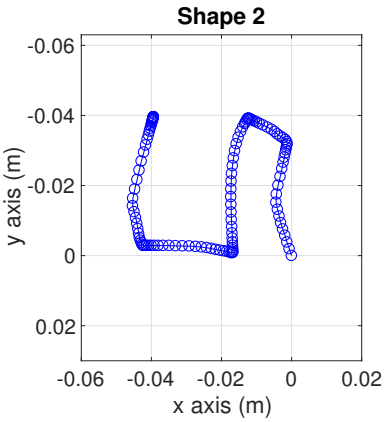
6.6 Conclusion

In this chapter, we presented the design and evaluation of WearAuth, a novel and practical system for user authentication on COTS wearable devices. WearAuth utilizes acoustic sensing to track a user's finger movement and explores user-specific features like a user's selective rhythm or drawing behavior for authentication. Exten-

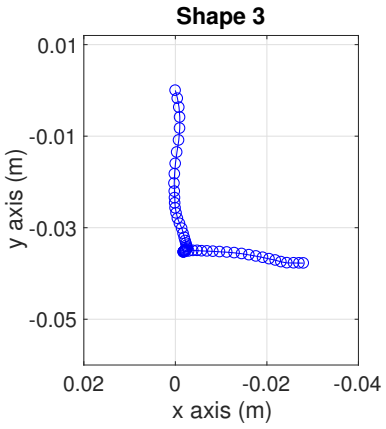
sive user experiments confirmed that WearAuth is highly usable and resilient to both random emulation and shoulder-surfing attacks.



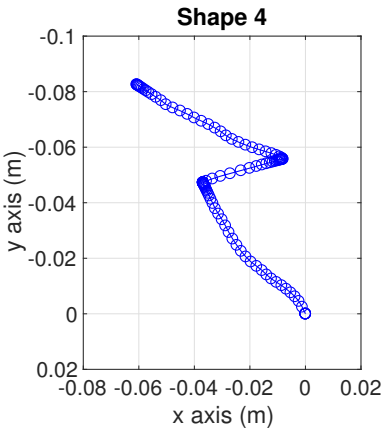
(a) Shape 1.



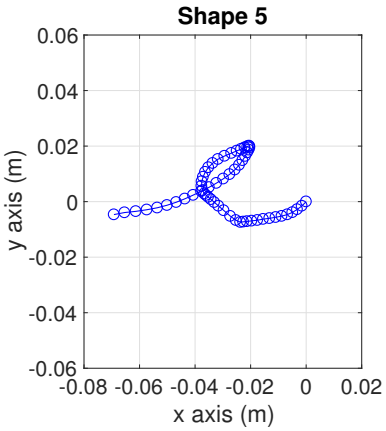
(b) Shape 2.



(c) Shape 3.



(d) Shape 4.



(e) Shape 5.

Figure 6.10: Shapes in Figure. 6.8.

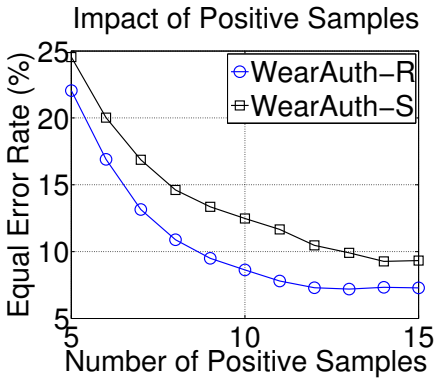
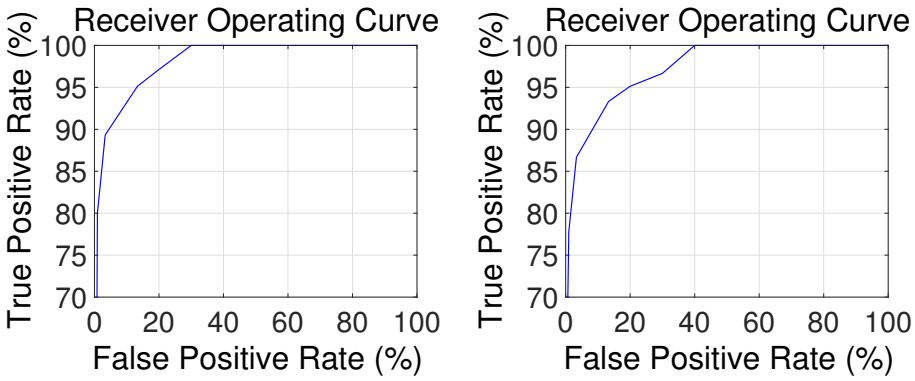


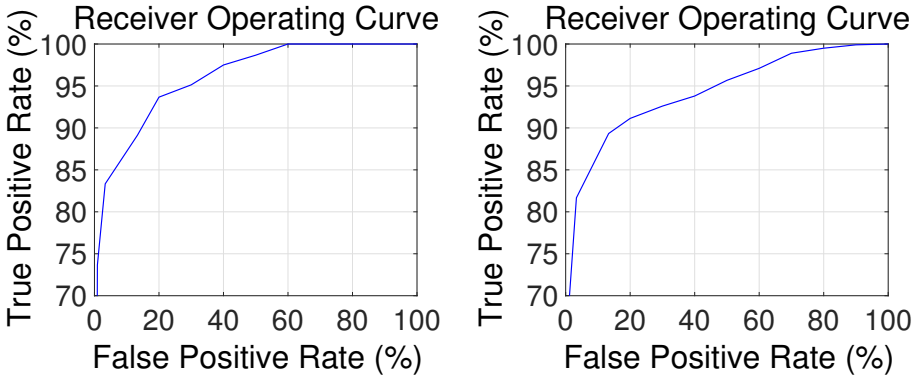
Figure 6.11: Impact of positive sample size on EER.



(a) ROC of WearAuth-R.

(b) ROC of WearAuth-S.

Figure 6.12: ROC without attacks.



(a) ROC of WearAuth-R.

(b) ROC of WearAuth-S.

Figure 6.13: ROC under shoulder-surfing attack.

Chapter 7

CONCLUSION AND FUTURE WORK

In this dissertation, we work on two lines related to security and privacy of mobile devices. Specifically, we focus on two novel attacks, i.e., mobile app fingerprinting and keystroke inference attack, and three novel countermeasures, i.e., rhythm-based authentication, liveness detection for mobile face authentication, and acoustic sensing for authentication on wearable devices. To achieve mobile app fingerprinting, we presented the design and evaluation of POWERFUL, a novel framework on Android mobile device which combines power analysis and machine learning for mobile app usage inference. POWERFUL exploits the app-specific characteristics of the power profiles without requiring user permission. Our extensive experiments demonstrated that POWERFUL is able to infer the app being used at a specific time with high accuracy, thus posing a realistic and serious threat to user privacy. To achieve keystroke inference on mobile devices, we presented the design and evaluation of EyeTell, a novel video-assisted attack that can infer a victim's keystrokes on his touchscreen device from a video capturing his eye movements. EyeTell explores the observation that human eyes naturally focus on and follow the keys they type, so a typing sequence on a soft keyboard results in a unique gaze trace of continuous eye movements. Comprehensive experiments on iOS and Android devices confirm the high efficacy of EyeTell. To achieve rhythm-based authentication, we presented the design and evaluation of RhyAuth, a novel two-factor rhythm-based authentication scheme for multi-touch mobile devices. RhyAuth is a two-factor authentication scheme that depends on a user-chosen rhythm and also the behavioral metrics for inputting the rhythm. Our experiments on Android devices confirm that RhyAuth is highly secure against vari-

ous attacks such as shoulder-surfing attack and emitation attack. To achieve liveness detection for mobile face authentication, we presented the design and evaluation of FaceHeart, a novel and practical scheme to secure face authentication on COTS mobile devices. FaceHeart relies on the non-forgeability of the photoplethysmograms extracted from two videos simultaneously taken through the front and rear cameras on a mobile device. Extensive user experiments confirm that FaceHeart can effectively thwart photo-based and video-based forgery attacks on mobile face authentication systems. To improve current authentication on wearable devices, we presented the design and evaluation of WearAuth, a novel and practical framework more favorable on wearables. WearAuth exploits acoustic sensing to track a user's finger movement and then extracts features from tracked finger trace for authentication. Our user experiments confirmed that WearAuth was secure against random emulation attack and shoulder-surfing attack, while preserving high usability.

Our current effort is still far from perfect. For rhythm-based mobile authentication, there have been some research results showing that user behavior on the touchscreen can be well-expected or easily-emitated with the help of robots. Such research imediately casts great shadow to user-biometric based authentication schemes. In order to defend against such heuristic attacks, we can explore the following two directions. First, assuming it is challenging for normal human attackers to emitate other user's biometric behaviors, we focus on detecting robot emitation by liveness detection of human fingers. Second, considering it is easier for either human attackers or robots to emitate other user's biometric behaviors with longer-duration data, we want to explort features extracted from touch events of shorter durations. For mobile app fingerprinting attack, there is much room for our system to improve. For example, currently we only use power profiles to infer the specific application a targeted victim was using. Due to the various factors affecting the power profiles, POWER-

FUL achieves acceptable inference accuracy only within a small set of apps. In fact, there are other public-available data on Android devices which may be beneficial for app fingerprinting. One example is CPU usage. CPU usage provides a new dimension to evaluate the app usage of the whole device. Therefore, we believe it may be combined with power profiles to improve the capability of POWERFUL. Another direction is to explore more advanced attacks based on our mobile app fingerprinting attack. One promising attack is to analyze the app preferences of a targeted victim to infer more private information, such as interests, living styles. For liveness detection of mobile face authentication, the main limitation is that FaceHeart requires a pair of face and fingertip video with at least four-second duration. Such a requirement may be impractical for frequent authentication, such as face unlock. One direction to relieve such requirement is to explore real-time matching of two PPGs. Therefore, our system needs to improve the estimation accuracy of PPG by better face detection and tracking and PPG extraction algorithms. For keystroke inference attacks on mobile devices, the main limitation is that the recording angle in the attack has to be sufficiently small to ensure an acceptable inference accuracy. Obiviously, this is a big constraint in real attack scenarios and also a big challenge to tackle. A possible solution is to look for gaze tracking algorithms which are less sensitive to user head postures. For authentication on wearable devices, one limitation is that most gesture-based schemes including ours can be subject to well-designed imitation attack. To counteract such potential attack, we forsee to use more advanced machine learning techniques like RNN to improve robustness of a trained model. Or we can simulate the imitation adversary using Generative Adversarial Network (GAN) to train with more advanced attack samples.

REFERENCES

[1] http://cloud.freehandmusic.netdna-cdn.com/preview/530x4/warner/amgrace.png.

[2] "Structure of human eye," https://en.wikipedia.org/wiki/Human_eye.

[3] S. Yun, Y. Chen, H. Zheng, L. Qiu, and W. Mao, "Strata: Fine-grained acoustic-based device-free tracking," in *ACM MobiCom*, Niagara Falls, NY, Jun. 2017.

[4] "Cisco visual networking index global mobile data traffic forecast update 2012-2017." [Online]. Available: http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-520862.html

[5] N. Sae-Bae, K. Ahmed, K. Isbister, and N. Memon, "Biometric-rich gestures: a novel approach to authentication on multi-touch devices," in *ACM CHI*, Austin, TX, May 2012.

[6] A. Aviv, K. Gibson, E. Mossop, M. Blaze, and J. Smith, "Smudge attacks on smartphone touch screens," in *WOOT*, Washington, DC, August 2010.

[7] http://www.who.int/blindness/en/.

[8] http://www.afb.org/section.aspx?SectionID=15.

[9] H. Bojinov and D. Boneh, "Mobile token-based authentication on a budget," in *HotMobile*, Phoenix, AZ, April 2011.

[10] T. Vu, A. Baid, S. Gao, M. Gruteser, R. Howard, J. Lindqvist, P. Spasojevic, and J. Walling, "Distinguishing users with capacitive touch communication," in *ACM MobiCom*, Istanbul, Turkey, August 2012.

[11] http://gizmodo.com/hackers-iphone-5s-fingerprint-security-is-not-secure-1367817697.

[12] M. Jakobsson, E. Shi, P. Golle, and R. Chow, "Implicit authentication for mobile devices," in *HotSec*, Montreal, Canada, August 2009.

[13] E. Shi, Y. Niu, M. Jakobsson, and R. Chow, "Implicit authentication through learning user behavior," in *ISC*, Boca Raton, FL, October 2010.

[14] J. Mantyjarvi, M. Lindholm, E. Vildjiounaite, S. Makela, and H. Ailisto, "Identifying users of portable devices from gait pattern with accelerometers," in *ICASSP*, Philadelphia, PA, March 2005.

[15] D. Gafurov, E. Snekkenes, and P. Bours, "Spoof attacks on gait authentication system," *IEEE Transactions on Information Forensics and Security*, vol. 2, no. 3, pp. 491–502, September 2007.

[16] A. Luca, A. Hang, F. Brudy, C. Lindner, and H. Hussman, "Touch me once and i know it's you! implicit authentication based on touch screen patterns," in *CHI*, Austin, TX, May 2012.

[17] F. Sandnes and X. Zhang, "User identification based on touch dynamics," in *UIC/ATC*, Fukuoka, Japan, September 2012.

[18] L. Li, X. Zhao, and G. Xue, "Unobservable re-authentication for smartphones," in *NDSS*, San Diego, USA, February 2013.

[19] "Nielsen mobile app analysis." [Online]. Available: http://www.nielsen.com/us/en/insights/news/2015/so-many-apps-so-much-more-time-for-entertainment.html

[20] "Alcatel-lucent motive security labs malware report - h1 2015." [Online]. Available: http://resources.alcatel-lucent.com/asset/189669

[21] E. Owusu, J. Han, S. Das, A. Perrig, and J. Zhang, "Accessory: password inference using accelerometers on smartphones," in *ACM HotMobile*, San Diego, CA, February 2012.

[22] L. Cai and H. Chen, "On the practicality of motion based keystroke inference attack." Springer Berlin Heidelberg, Jun. 2012.

[23] R. Schlegel, K. Zhang, X. Zhou, M. Intwala, A. Kapadia, and X. Wang, "Soundcomber: A stealthy and context-aware sound trojan for smartphones." in *NDSS*, San Diego, CA, February 2011.

[24] Z. Xu, K. Bai, and S. Zhu, "Taplogger: Inferring user inputs on smartphone touchscreens using on-board motion sensors," in *ACM WiSec*, Tucson, AZ, April 2012.

[25] L. Cai and H. Chen, "Touchlogger: Inferring keystrokes on touch screen from smartphone motion," in *USENIX HotSec*, San Francisco, CA, August 2011.

[26] J. Han, E. Owusu, L. Nguyen, A. Perrig, and J. Zhang, "Accomplice: Location inference using accelerometers on smartphones," in *IEEE COMSNETS*, Bangalore, India, January 2012.

[27] M. Azizyan, I. Constandache, and R. Choudhury, "Surroundsense: mobile phone localization via ambience fingerprinting," in *ACM MobiCom*, Beijing, China, September 2009.

[28] S. Jana and V. Shmatikov, "Memento: Learning secrets from process footprints," in *IEEE S&P*, San Francisco, CA, May 2012.

[29] K. Zhang and X. Wang, "Peeping tom in the neighborhood: keystroke eavesdropping on multi-user systems," in *USENIX Security*, Montreal, Canada, August 2009.

[30] X. Zhou, S. Demetriou, D. He, M. Naveed, X. Pan, X. Wang, C. Gunter, and K. Nahrstedt, "Identity, location, disease and more: Inferring your secrets from android public resources," in *ACM CCS*, berlin, Germany, November 2013.

[31] Q. Chen, Z. Qian, and Z. Mao, "Peeking into your app without actually seeing it: Ui state inference and novel android attacks," in *USENIX Security*, San Diego, CA, August 2014.

[32] T. Stöber, M. Frank, J. Schmitt, and I. Martinovic, "Who do you sync you are?: smartphone fingerprinting via application behaviour," in *ACM WiSec*, Budapest, Hungary, April 2013.

[33] Q. Xu, Y. Liao, S. Miskovic, Z. Mao, M. Baldi, A. Nucci, and T. Andrews, "Automatic generation of mobile app signatures from traffic observations," in *IEEE INFOCOM*, Hong Kong, April 2015.

[34] S. Miskovic, G. Lee, Y. Liao, and M. Baldi, "Appprint: Automatic fingerprinting of mobile applications in network traffic."

[35] S. Dai, A. Tongaonkar, X. Wang, A. Nucci, and D. Song, "Networkprofiler: Towards automatic fingerprinting of android apps," in *IEEE INFOCOM*, Turin, Italy, April 2013.

[36] N. Verde, G. Ateniese, E. Gabrielli, L. Mancini, and A. Spognardi, "No nat'd user left behind: Fingerprinting users behind nat from netflow records alone," in *IEEE ICDCS*, Madrid, Spain, Jul. 2014.

[37] Q. Wang, A. Yahyavi, B. Kemme, and W. He, "I know what you did on your smartphone: Inferring app usage over encrypted data traffic," in *IEEE CNS'15*, Florence, Italy, Sep. 2015.

[38] [Online]. Available: http://www.channelpronetwork.com/article/mobile-device-security-startling-statistics-data-loss-and-data-breachesl

[39] M. Shahzad, A. Liu, and A. Samuel, "Secure unlocking of mobile touch screen devices by simple gestures: You can see it but you can not do it," in *ACM MobiCom*, Miami, USA, Sep. 2013.

[40] J. Sun, X. Chen, J. Zhang, Y. Zhang, and J. Zhang, "TouchIn: Sightless two-factor authentication on multi-touch mobile devices," in *IEEE CNS*, San Francisco, CA, October 2014.

[41] Y. Chen, J. Sun, R. Zhang, and Y. Zhang, "Your song your way: Rhythm-based two-factor authentication for multi-touch mobile devices," in *IEEE INFOCOM*, Hong Kong, China, April 2015.

[42] T. Li, Y. Chen, J. Sun, X. Jin, and Y. Zhang, "iLock: Immediate and automatic locking of mobile devices against data theft," in *ACM CCS*, Vienna, Austria, October 2016.

[43] O. Kähm and N. Damer, "2d face liveness detection: An overview," in *IEEE BIOSIG*, Darmstadt, German, September 2012.

[44] K. Kollreider, H. Fronthaler, and J. Bigun, "Non-intrusive liveness detection by face images," *Image and Vision Computing*, vol. 27, no. 3, pp. 233–244, February 2009.

[45] R. Ghiass, O. Arandjelovic, H. Bendada, and X. Maldague, "Infrared face recognition: a literature review," in *IEEE IJCNN*, Dallas, TX, August 2013.

[46] J. Määttä, A. Hadid, and M. Pietikainen, "Face spoofing detection from single images using micro-texture analysis," in *IEEE IJCB*, Washington, DC, October 2011.

[47] X. Tan, Y. Li, J. Liu, and L. Jiang, "Face liveness detection from a single image with sparse low rank bilinear discriminative model," in *ECCV*, Crete, Greece, September 2010.

[48] S. Chen, A. Pande, and P. Mohapatra, "Sensor-assisted facial recognition: an enhanced biometric authentication system for smartphones," in *ACM MobiSys*, Bretton Woods, NH, Jun. 2014.

[49] Y. Li, Y. Li, Q. Yan, H. Kong, and R. Deng, "Seeing your face is not enough: An inertial sensor-based liveness detection for face authentication," in *ACM CCS*, Denver, CO, October 2015.

[50] M. Backes, M. Dürmuth, and D. Unruh, "Compromising reflections-or-how to read lcd monitors around the corner," in *IEEE Symposium on Security and Privacy*, Oakland, CA, USA, May 2008.

[51] M. Backes, T. Chen, M. Duermuth, H. Lensch, and M. Welk, "Tempest in a teapot: Compromising reflections revisited," in *IEEE Symposium on Security and Privacy*, Oakland, CA, USA, May 2009.

[52] D. Balzarotti, M. Cova, and G. Vigna, "ClearShot: Eavesdropping on keyboard input from video," in *IEEE Symposium on Security and Privacy*, Oakland, CA, USA, May 2008.

[53] F. Maggi, A. Volpatto, S. Gasparini, G. Boracchi, and S. Zanero, "A fast eavesdropping attack against touchscreens," in *Information Assurance and Security*, Melaka, Malaysia, December 2011.

[54] R. Raguram, A. White, D. Goswami, F. Monrose, and J.-M. Frahm, "iSpy: Automatic reconstruction of typed input from compromising reflections," in *ACM CCS*, Chicago, IL, USA, October 2011.

[55] Y. Xu, J. Heinly, A. White, F. Monrose, and J. Frahm, "Seeing double: Reconstructing obscured typed input from repeated compromising reflections," in *ACM CCS*, Berlin, Germany, October 2013.

[56] Q. Yue, Z. Ling, X. Fu, B. Liu, K. Ren, and W. Zhao, "Blind recognition of touched keys on mobile devices," in *ACM CCS*, Scottsdale, Arizona, November 2014.

[57] D. Shukla, R. Kumar, A. Serwadda, and V. Phoha, "Beware, your hands reveal your secrets!" in *ACM CCS*, Scottsdale, AZ, November 2014.

[58] J. Sun, X. Jin, Y. Chen, J. Zhang, R. Zhang, and Y. Zhang, "Visible: Video-assisted keystroke inference from tablet backside motion," in *NDSS*, San Diego, CA, Feb. 2016.

[59] J. O. Wobbrock, "Tapsongs: tapping rhythm-based passwords on a single binary sensor," in *ACM UIST*, Victoria, Canada, Oct. 2009.

[60] D. Marques, T. Guerreiro, L. Duarte, and L. Carriço, "Under the table: Tap authentication for smartphones," *ACM BCS-HCI*, Sep. 2013.

[61] F. X. Lin, D. Ashbrook, and S. White, "Rhythmlink: securely pairing i/o-constrained devices by tapping," in *ACM UIST*, Santa Barbara, CA, Oct. 2011.

[62] N. Zheng, K. Bai, H. Huang, and H. Wang, "You are how you tap: A two-factor authentication for smartphone users," in *IEEE ICNP*, North Carolina, October 2014.

[63] A. Serwadda and V. V. Phoha, "When kids' toys breach mobile phone security," in *ACM CCS*, Berlin, Germany, November 2013.

[64] http://en.wikipedia.org/wiki/Simple_moving_average.

[65] http://en.wikipedia.org/wiki/Dynamic_time_warping.

[66] C. Chang and C. Lin, "Libsvm: a library for support vector machines," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 2, no. 3, p. 27, April 2011.

[67] M. Sherman, G. Clark, Y. Yang, S. Sugrim, A. Modig, J. Lindqvist, A. Oulasvirta, and T. Roos, "User-generated free-form gestures for authentication: security and memorability," in *ACM MobiSys*, Bretton Woods, NH, Jun. 2014.

[68] S. Uellenbeck, M. Durmuth, C. Wolf, and T. Holz, "Quantifying the security of graphical passwords: The case of android unlock patterns," in *ACM CCS*, Berlin, Germany, November 2013.

[69] Y. Chen, X. Jin, J. Sun, R. Zhang, and Y. Zhang, "POWERFUL: Mobile app fingerprinting via power analysis," in *IEEE INFOCOM*, Atlanta, GA, April 2017.

[70] J. Sun, X. Jin, Y. Chen, J. Zhang, R. Zhang, and Y. Zhang, "Visible: Video-assisted keystroke inference from tablet backside motion," in *NDSS*, San Diego, CA, Feb. 2016.

[71] https://github.com/sonyxperiadev/ApkAnalyser.

[72] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. Dick, Z. Mao, and L. Yang, "Accurate online power estimation and automatic battery behavior based power model generation for smartphones," in *CODES/ISSS*, Scottsdale, AZ, 2010.

[73] L. Ardito, G. Procaccianti, M. Torchiano, and G. Migliore, "Profiling power consumption on mobile devices," 2013.

[74] A. Pathak, Y. Hu, M. Zhang, P. Bahl, and Y. Wang, "Fine-grained power modeling for smartphones using system call tracing," in *ACM EuroSys*, Salzburg, Austria, April 2011.

[75] A. Pathak, Y. Hu, and M. Zhang, "Where is the energy spent inside my app?: fine grained energy accounting on smartphones with eprof," in *ACM EuroSys*, Bern, Switzerland, April 2012.

[76] N. Brouwers, M. Zuniga, and K. Langendoen, "Neat: a novel energy analysis toolkit for free-roaming smartphones," in *ACM SenSys*, memphis, TN, November 2014.

[77] Y. Michalevsky, A. Schulman, G. Veerapandian, D. Boneh, and G. Nakibly, "Powerspy: Location tracking using mobile device power analysis," in *USENIX Security*, Washington, D.C., August 2015.

[78] http://www.ibtimes.co.uk/android-apps-one-ten-affected-malware-viruses-states-new-research-1459576.

[79] http://www.leviathansecurity.com/blog/zero-permission-android-applications.

[80] R. Quinlan, *C4.5: Programs for Machine Learning*. San Mateo, CA: Morgan Kaufmann Publishers, 1993.

[81] T. Ho, "The random subspace method for constructing decision forests," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, no. 8, pp. 832–844, 1998.

[82] J. Suykens and J. Vandewalle, "Least squares support vector machine classifiers," *Neural processing letters*, vol. 9, no. 3, pp. 293–300, 1999.

[83] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. Witten, "The weka data mining software: an update," *ACM SIGKDD explorations newsletter*, Nov 2009.

[84] D. Roobaert, G. Karakoulas, and N. Chawla, "Information gain, correlation and support vector machines," in *Feature Extraction*. Springer, 2006, pp. 463–470.

[85] T. M. Mitchell, *Machine Learning*. New York, NY, USA: McGraw-Hill, Inc., 1997.

[86] Y. Chen, J. Sun, X. Jin, T. Li, R. Zhang, and Y. Zhang, "Your face your heart: Secure mobile face authentication with photoplethysmograms," in *IEEE INFOCOM*, Atlanta, GA, April 2017.

[87] K. Shelley and S. Shelley, "Pulse oximeter waveform: photoelectric plethysmography," *Clinical Monitoring, Carol Lake, R. Hines, and C. Blitt, Eds.: WB Saunders Company*, pp. 420–428, 2001.

[88] M. Kumar, A. Veeraraghavan, and A. Sabharwal, "Distanceppg: Robust non-contact vital signs monitoring using a camera," *Biomedical optics express*, vol. 6, no. 5, pp. 1565–1588, May 2015.

[89] M. Lewandowska, J. Rumiński, T. Kocejko, and J. Nowak, "Measuring pulse rate with a webcamâĂŤa non-contact method for evaluating cardiac activity," in *IEEE FedCSIS*, Szczecin, Poland, September 2011.

[90] X. Li, J. Chen, G. Zhao, and M. Pietikainen, "Remote heart rate measurement from face videos under realistic situations," in *IEEE CVPR*, Columbus, OH, Jun. 2014.

[91] M.-Z. Poh, D. J. McDuff, and R. W. Picard, "Non-contact, automated cardiac pulse measurements using video imaging and blind source separation." *Optics express*, vol. 18, no. 10, pp. 10 762–10 774, May 2010.

[92] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *IEEE CVPR*, Kauai, HI, December 2001.

[93] B. D. Lucas, T. Kanade *et al.*, "An iterative image registration technique with an application to stereo vision." in *IJCAI*, 1981.

[94] C. Tomasi and T. Kanade, *Detection and tracking of point features*. School of Computer Science, Carnegie Mellon Univ. Pittsburgh, 1991.

[95] J. Shi and C. Tomasi, "Good features to track," in *IEEE CVPR*, Seattle, WA, Jun. 1994.

[96] A. Asthana, S. Zafeiriou, S. Cheng, and M. Pantic, "Incremental face alignment in the wild," in *IEEE CVPR*, Columbus, OH, Jun. 2014.

[97] W. Verkruysse, L. O. Svaasand, and J. S. Nelson, "Remote plethysmographic imaging using ambient light." *Optics express*, vol. 16, no. 26, pp. 21 434–21 445, 2008.

[98] A. Lam and Y. Kuno, "Robust heart rate measurement from video using select random patches," in *IEEE CVPR*, Santiago, Chile, December 2015.

[99] S. Haykin and B. Widrow, *Least-mean-square adaptive filters*. John Wiley & Sons, 2003, vol. 31.

[100] N. Friedman, D. Geiger, and M. Goldszmidt, "Bayesian network classifiers," *Machine learning*, vol. 29, no. 2-3, pp. 131–163, 1997.

[101] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006.

[102] S. Haykin and N. Network, "A comprehensive foundation," *Neural Networks*, vol. 2, no. 2004, 2004.

[103] M. J. Gregoski, M. Mueller, A. Vertegel, A. Shaporev, B. B. Jackson, R. M. Frenzel, S. M. Sprehn, and F. A. Treiber, "Development and validation of a smartphone heart rate acquisition application for health promotion and wellness telehealth applications," *International journal of telemedicine and applications*, vol. 2012, p. 1, 2012.

[104] J. A. Hanley and B. J. McNeil, "The meaning and use of the area under a receiver operating characteristic (roc) curve." *Radiology*, vol. 143, no. 1, pp. 29–36, 1982.

[105] https://i-msdn.sec.s-msft.com/dynimg/IC584331.png.

[106] S. Xu, L. Sun, and G. K. Rohde, "Robust efficient estimation of heart rate pulse from video," *Biomedical optics express*, vol. 5, no. 4, pp. 1124–1135, March 2014.

[107] Y. Chen, T. Li, R. Zhang, Y. Zhang, and T. Hedgpeth, "Eyetell: Video-assisted touchscreen keystroke inference from eye movements," in *IEEE Symposium on Security and Privacy*, San Franscico, CA, May 2018, conditionally accepted.

[108] E. Wood and A. Bulling, "Eyetab: Model-based gaze estimation on unmodified tablet computers," in *ACM ETRA*, Safety Harbor, FL, March 2014.

[109] D. Hansen and Q. Ji, "In the eye of the beholder: A survey of models for eyes and gaze," *IEEE Trans. PAMI*, vol. 32, no. 3, pp. 478–500, March 2010.

[110] Z. Zhu and Q. Ji, "Novel eye gaze tracking techniques under natural head movement," *IEEE Trans. BME*, vol. 54, no. 12, pp. 2246–2260, December 2007.

[111] E. Miluzzo, A. Varshavsky, S. Balakrishnan, and R. Choudhury, "Tapprints: your finger taps have fingerprints," in *ACM MobiSys*, Low Wood Bay, Lake District, UK, June 2012.

[112] L. Simon and R. Anderson, "PIN skimmer: Inferring pins through the camera and microphone," in *ACM SPSM*, Berlin, Germany, November 2013.

[113] S. Narain, A. Sanatinia, and G. Noubir, "Single-stroke language-agnostic keylogging using stereo-microphones and domain specific machine learning," in *ACM WiSec*, Oxford, United Kingdom, July 2014.

[114] D. Asonov and R. Agrawal, "Keyboard acoustic emanations," in *IEEE S&P*, Oakland, CA, May 2004.

[115] L. Zhuang, F. Zhou, and J. Tygar, "Keyboard acoustic emanations revisited," in *ACM CCS*, Alexandria, VA, November 2005.

[116] Y. Berger, A. Wool, and A. Yeredor, "Dictionary attacks using keyboard acoustic emanations," in *ACM CCS*, Alexandria, VA, November 2006.

[117] T. Zhu, Q. Ma, S. Zhang, and Y. Liu, "Context-free attacks using keyboard acoustic emanations," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communication Security*, Scottsdale, Arizona, USA, November 2014.

[118] J. Liu, Y. Wang, G. Kar, Y. Chen, J. Yang, and M. Gruteser, "Snooping keystrokes with mm-level audio ranging on a single phone," in *ACM MobiCom*, Paris, France, September 2015.

[119] P. Marquardt, A. Verma, H. Carter, and P. Traynor, "(sp)iphone: Decoding vibrations from nearby keyboards using mobile phone accelerometers," in *ACM CCS*, Chicago, IL, November 2011.

[120] X. Liu, Z. Zhou, W. Diao, Z. Li, and K. Zhang, "When good becomes evil: Keystroke inference with smartwatch," in *ACM CCS*, Denver, CO, October 2015.

[121] K. Ali, A. Liu, W. Wang, and M. Shahzad, "Keystroke recognition using wifi signals," in *ACM MobiCom*, Paris, France, September 2015.

[122] M. Li, Y. Meng, J. Liu, H. Zhu, X. Liang, Y. Liu, and N. Ruan, "When CSI meets public WiFi: Inferring your mobile phone password via WiFi signals," in *ACM CCS*, Vienna, Austria, October 2016.

[123] J. Zhang, X. Zheng, Z. Tang, T. Xing, X. Chen, D. Fang, R. Li, X. Gong, and F. Chen, "Privacy leakage in mobile sensing: Your unlock passwords can be leaked through wireless hotspot functionality," *Mobile Information Systems*, 2016.

[124] R. Bednarik, T. Kinnunen, A. Mihaila, and P. Fränti, "Eye-movements as a biometric," in *SCIA*, Copenhagen, Denmark, June 2005.

[125] O. Komogortsev, A. Karpov, and C. Holland, "CUE: counterfeit-resistant usable eye movement-based authentication via oculomotor plant characteristics and complex eye movement patterns," in *SPIE Defense, Security, and Sensing*, Baltimore, May 2012.

[126] C. Holland and O. Komogortsev, "Complex eye movement pattern biometrics: Analyzing fixations and saccades," in *IAPR ICB*, Madrid, Spain, June 2013.

[127] T. Li, Y. Chen, J. Sun, X. Jin, and Y. Zhang, "ilock: Immediate and automatic locking of mobile devices against data theft," in *ACM CCS*, Vienna, Austria, October 2016.

[128] A. D. Luca, R. Weiss, and H. Drewes, "Evaluation of eye-gaze interaction methods for security enhanced pin-entry," in *ACM OZCHI*, Adelaide, Australia, November 2007.

[129] A. D. Luca, M. Denzel, and H. Hussmann, "Look into my eyes!: Can you guess my password?" in *ACM SOUPS*, Mountain View, CA, July 2009.

[130] D. Liu, B. Dong, X. Gao, and H. Wang, "Exploiting eye tracking for smartphone authentication," in *ACNS*, New York, NY, June 2015.

[131] Z. Li, M. Li, P. Mohapatra, J. Han, and S. Chen, "iType: Using eye gaze to enhance typing privacy," in *IEEE INFOCOM*, Atlanta, GA, May 2017.

[132] A. Al-Haiqi, M. Ismail, and R. Nordin, "The eye as a new side channel threat on smartphones," in *IEEE SCORed*, Putrajaya, Malaysia, December 2013.

[133] G. Ye, Z. Tang, D. Fang, X. Chen, K. Kim, B. Taylor, and Z. Wang, "Cracking Android pattern lock in five attempts," in *ISOC NDSS*, San Diego, CA, February 2017.

[134] F. Timm and E. Barth, "Accurate eye centre localisation by means of gradients." in *VISAPP*, Algarve, Portugal, March 2011.

[135] J. Daugman, "High confidence visual recognition of persons by a test of statistical independence," *IEEE Trans. PAMI*, vol. 15, no. 11, pp. 1148–1161, November 1993.

[136] J. Wang, E. Sung, and R. Venkateswarlu, "Eye gaze estimation from a single image of one eye," in *IEEE ICCV*, Nice, France, October 2003.

[137] L. Świrski, A. Bulling, and N. Dodgson, "Robust real-time pupil tracking in highly off-axis images," in *ACM ETRA*, Santa Barbara, CA, March 2012.

[138] E. Wood, "Gaze tracking for commodity portable devices," Ph.D. dissertation, University of Cambridge, 2013.

[139] M. Kumar, J. Klingner, R. Puranik, T. Winograd, and A. Paepcke, "Improving the accuracy of gaze input for interaction," in *ACM ETRA*, Savannah, GA, March 2008.

[140] "corn-cob dictionary," http://www.mieliestronk.com/wordlist.html.

[141] "Trie data structure," https://en.wikipedia.org/wiki/Trie.

[142] "120 patterns for pattern lock keyboard," http://www.research.lancs.ac.uk/portal/files/138568011/Patterns.pdf.

[143] D. Ping, X. Sun, and B. Mao, "Textlogger: inferring longer inputs on touch screen using motion sensors," in *ACM WiSec*, New York, NY, June 2015.

[144] K. Krafka, A. Khosla, P. Kellnhofer, H. Kannan, S. Bhandarkar, W. Matusik, and A. Torralba, "Eye tracking for everyone," in *IEEE CVPR*, Las Vegas, NV, June 2016.

[145] https://support.apple.com/en-us/ht201371.

[146] https://support.apple.com/en-us/ht208108.

[147] Z. Wang, C. Shen, and Y. Chen, "Handwaving authentication: Unlocking your smartwatch through handwaving biometrics," in *Springer CCBR*, Shenzhen, China, Oct. 2017.

[148] J. Yang, Y. Li, and M. Xie, "Motionauth: Motion-based authentication for wrist worn smart devices," in *IEEE PerCom Workshops*, St. Louis, Missouri, Mar. 2015.

[149] Y. Li, M. Xie, and J. Bian, "Segauth: A segment-based approach to behavioral biometric authentication," in *IEEE CNS*, Philadelphia, PA, Oct. 2016.

[150] A. Lewis, Y. Li, and M. Xie, "Real time motion-based authentication for smartwatch," in *IEEE CNS*, Philadelphia, PA, Oct. 2016.

[151] Y. Li and M. Xie, "Understanding secure and usable gestures for realtime motion based authentication," in *IEEE INFOCOM Workshops*, Honolulu, HI, Apr. 2018.

[152] M. Roshandel, A. Munjal, P. Moghadam, S. Tajik, and H. Ketabdar, "Multisensor finger ring for authentication based on 3d signatures," in *Springer HCI*, Crete, Greece, Jun. 2014.

[153] N. Z. Gong, A. Ozen, Y. Wu, X. Cao, R. Shin, D. Song, H. Jin, and X. Bao, "Piano: Proximity-based user authentication on voice-powered internet-of-things devices," in *IEEE ICDCS*, Atlanta, GA, Jun. 2017.

[154] J. Chauhan, Y. Hu, S. Seneviratne, A. Misra, A. Seneviratne, and Y. Lee, "Breathprint: Breathing acoustics-based user authentication," in *ACM MobiSys*, Niagara Falls, NY, Jun. 2017.

[155] J. Chauhan, S. Seneviratne, Y. Hu, A. Misra, A. Seneviratne, and Y. Lee, "Breathing-based authentication on resource-constrained iot devices using recurrent neural networks," *IEEE Computer*, vol. 51, no. 5, pp. 60–67, May 2018.

[156] B. Hutchins, A. Reddy, W. Jin, M. Zhou, M. Li, and L. Yang, "Beat-pin: A user authentication mechanism for wearable devices through secret beats," in *ACM ASIACCS*, Incheon, Korea, Jun. 2018.

[157] R. Nandakumar, V. Iyer, D. Tan, and S. Gollakota, "Fingerio: Using active sonar for fine-grained finger tracking," in *ACM CHI*, San Jose, CA, May 2016.

[158] W. Wang, A. X. Liu, and K. Sun, "Device-free gesture tracking using acoustic signals," in *ACM MobiCom*, New York, NY, Oct. 2016.

[159] D. Tse and P. Viswanath, *Fundamentals of wireless communications*. Cambridge University Press, 2005.

[160] M. Pukkila, "Channel estimation modeling," *Nokia Research Center*, 2000.

[161] T. Rakthanmanon, B. Campana, A. Mueen, G. Batista, B. Westover, Q. Zhu, J. Zakaria, and E. Keogh, "Searching and mining trillions of time series subsequences under dynamic time warping," in *ACM SIGKDD*, Beijing, China, Aug. 2012.

[162] https://www.ifixit.com/Teardown/Apple+Watch+Series+2+Teardown/67385.

[163] https://en.wikipedia.org/wiki/Samsung_Galaxy_Gear.

[164] https://www.motorola.com.au/products/moto-360.